Sobrecarga de Operadores

Sobrecarga de Operadores

- Como vimos até agora, operações relacionadas a objetos são relizadas através da chamada de métodos
  - Porém, podemos utilizar os operadores da linguagem C++ para especificar operações comuns de manipulação de objetos;
  - Novos operadores não podem ser criados;
  - A adaptação dos operadores nativos da linguagem para nossas classes é chamada de **sobrecarga de operadores**
    - Similar à sobrecarga de funções e métodos;
    - Dependendo dos operandos envolvidos, o operador se comporta diferentemente;
    - Ou seja, se torna **sensível ao contexto**.

Sobrecarga de Operadores

- Por exemplo, os operadores ariméticos são sobrecarregados por padrão
  - Funcionam para quaisquer tipo de número (*int*, *float*, *double*, *long*, etc.);
  - Outro operador que sobrecarregado é o de atribuição.
- Qualquer operação realizada por um operador sobrecarregado também pode ser realizada por um método sobrecarregado
  - No entanto, a notação de operador é muito mais simples e natural.
- Um bom exemplo de classe com operadores sobrecarregados é a *string*.

Sobrecarga de Operadores

- A principal convenção a respeito da sobrecarga de operadores é a de que o comportamento do operador utilizado seja análogo ao original
  - Ou seja, **+** só deve ser utilizado para realizar adição.
- Para sobrecarregar um operador criamos um método ou função global cujo nome será ***operator***, seguido pelo símbolo do operador a ser sobrecarregado
  - Por exemplo, ***operator +()***;

Sobrecarga de Operadores

- **Atenção!**
  - Por padrão, não se sobrecarrega os operadores `=`, `,` e `&`;
  - Os operadores `.` ,`.*`, `?:` e `::` não podem ser sobrecarregados;
  - Não é possível sobrecarregar um operador para lidar com tipos primitivos
    - Soma de inteiros não pode ser alterada.
  - Não é possível alterar a precedência de um operador através de sobrecarga;
  - O número de operandos de um operador não pode ser alterado através de sobrecarga;
  - Sobrecarregar um operador não sobrecarrega os outros
    - Sobrecarregar **+** não sobrecarrega **+=**;

Sobrecarga de Operadores

- Métodos vs. Funções
  - Quando sobrecarregamos um operador usando um método, o operando da esquerda deve sempre ser um objeto
    - Somente neste caso será utilizado o operador sobrecarregado.
  - Se não possuirmos certeza sobre isto, devemos usar uma função
    - Permitirá a comutatividade.
    - Se for necessário acessar os atributos de um objeto, declaramos a função como **amiga**.
  - Os operadores (), [], -> e qualquer operador de atribuição devem ser sobrecarregados como métodos.

Sobrecarga de Operadores

- Podemos, por exemplo, sobrecarregar o operador **>>** utilizado para leitura de dados
  - Note que o operando do lado esquerdo é um objeto da classe ***istream***, da biblioteca padrão C++
    - Não podemos alterá-la para incluir a sobrecarga;
    - Mas podemos criar uma função que altere o operador do lado direito.
- Por exemplo, vamos sobrecarregar os operadores **>>** e **<<** para objetos que representam números de telefone
  - Criaremos uma classe para representar os número de telefone com **funções amigas** que sobrecarregam os operadores **>>** e **<<**.

NumeroTelefone.h

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

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

private:
   string DDD;      // código de área de 2 algarismos
   string inicio;   // linha de 4 algarismos
   string fim;      // linha de 4 algarismos
};
```

Sobrecarga de Operadores

```
friend ostream &operator<<( ostream &, const NumeroTelefone & );
```

- A sobrecarga do operador **<<** é implementada através de uma função amiga, que retorna uma referência a um objeto *ostream* (fluxo de saída) e recebe como parâmetros referências para um objeto *ostream* e para um objeto *NumeroTelefone*;

```
friend istream &operator>>( istream &, NumeroTelefone & );
```

- A sobrecarga do operador **>>** é implementada através de uma função amiga, que retorna uma referência a um objeto *istream* (fluxo de entrada) e recebe como parâmetros referências para um objeto *istream* e para um objeto *NumeroTelefone*.

Sobrecarga de Operadores

- O fato de ambas as funções retornarem um objeto, além de alterarem os atributos do objeto da classe *NumeroTelefone* permite construções do tipo:
  - cin>>a;
  - cin>>a>>b>>c;
  - cout<<a;
  - cout<<a<<b<<c;
- Caso contrário não seria possível realizar chamadas "em cascata".

NumeroTelefone.cpp

```
#include <iomanip>
using namespace std;
#include "NumeroTelefone.h"

ostream &operator<<( ostream &output, const NumeroTelefone &numero )
{
   output << "(" << numero.DDD << ") "
          << numero.inicio << "-" << numero.fim;
   return output; // permite cout << a << b << c;
}

istream &operator>>( istream &input, NumeroTelefone &numero )
{
   input.ignore(); // pula (
   input >> setw( 2 ) >> numero.DDD; // entrada do código de área
   input.ignore( 2 ); // pula ) e espaço
   input >> setw( 4 ) >> numero.inicio; // primeira parte
   input.ignore(); // pula traço (-)
   input >> setw( 4 ) >> numero.fim; // segunda parte
   return input; // permite cin >> a >> b >> c;
}
```

driverNumeroTelefone.cpp

```
#include <iostream>
using namespace std;
#include "NumeroTelefone.h"

int main()
{
   NumeroTelefone telefone; // cria objeto telefone

   cout << "Digite o numero no formato (12) 3456-7890:" << endl;

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

   cout << "O numero informado foi: ";

   // cout << telefone invoca operator<< emitindo implicitamente
   // chamada da função global operator<<( cout, telefone )
   cout << telefone << endl;
   return 0;
}
```

Sobrecarga de Operadores

- Quando a chamada "**cin >> telefone;**" é executada, na verdade chamamos "**operator>>(cin, telefone)**;";
- Nas funções de sobrecarga, *cin* e *cout* recebem outro nome, *input* e *output*, respectivamente;
- Note que na sobrecarga do operador << o objeto da classe *NumeroTelefone* é recebido como uma constante
  - Menor privilégio: se o método não alterará o objeto, então este deve ser uma constante;
  - Um programa só pode chamar os métodos **const** de um objeto constante.
- Note ainda que o programa pressupõe que o usuário digitará corretamente os dados
  - Como alterar isto?

Sobrecarga de Operadores

- Vamos alterar nosso exemplo para sobrecarregar o operador *
  - Ele será utilizado para atribuição
    - Não sobrecarregaremos o operador = porque este já é sobrecarregado.
  - Será implementado como um método;
- Este exemplo também ilustra uma sobrecarga que não retorna nada
  - Desta forma não será possível realizar chamadas do tipo

a * b* c;

Telefone.h

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

class NumeroTelefone
{
   friend ostream &operator<<( ostream &, const NumeroTelefone & );
   friend istream &operator>>( istream &, NumeroTelefone & );
public:
   void operator*(NumeroTelefone ); // sobrecarga do operador *
private:
   string DDD;
   string inicio;
   string fim;
};
```

Telefone.cpp

```
ostream &operator<<( ostream &output, const NumeroTelefone &numero )
{
   output << "(" << numero.DDD << ") "<< numero.inicio << "-" <<
numero.fim;
   return output;
}

istream &operator>>( istream &input, NumeroTelefone &numero )
{
   input.ignore();
   input >> setw( 2 ) >> numero.DDD;
   input.ignore( 2 );
   input >> setw( 4 ) >> numero.inicio;
   input.ignore();
   input >> setw( 4 ) >> numero.fim;
   return input;
}
```

```
void NumeroTelefone::operator*(NumeroTelefone ladoDireito)
{
      DDD = ladoDireito.DDD; // Realiza a cópia dos atributos
      inicio =  ladoDireito.inicio;
      fim =  ladoDireito.fim;
}
```

DriverTelefone.cpp

```
#include <iostream>
using namespace std;
#include "NumeroTelefone.h"

int main()
{
   NumeroTelefone telefone, celular; // cria objeto telefone

   cout << "Digite o numero no formato (12) 3456-7890:" << endl;
   cin >> telefone;
   cout << "O numero informado foi: ";

   celular * telefone; // chamada do método celular.*(telefone);

   cout << celular << endl;
   return 0;
}
```