

Linguagem de Programação C++

Exercício:

Agora, efetuaremos mais uma expansão na classe `DataSobrecargaOperador` sobrecarregando o operador binário - e operador unário -. Você deve possibilitar que o operador binário - possa ser utilizado para subtrair duas datas, gerando um inteiro que representa a quantidade de dias decorridos entre as datas. Já o operador - unário, ao ser aplicado sobre um objeto da classe `DataSobrecargaOperador` retorna o número de dias decorridos desde o início do ano contido na data.

Observação: considere os anos bissextos.

```

//conteúdo do arquivo dataSobrecargaOperador.h
#ifndef DATASOBRECARGAOPERADOR_H
#define DATASOBRECARGAOPERADOR_H
    #include <iostream>
    class DataSobrecargaOperador {
        friend DataSobrecargaOperador operator+ (int,
DataSobrecargaOperador &);
        friend DataSobrecargaOperador operator+
(DataSobrecargaOperador &, int);
        friend ostream &operator<<(ostream &, DataSobrecargaOperador &);
        friend istream &operator>>(istream &, DataSobrecargaOperador &);
    public:
        DataSobrecargaOperador(int=1, int=1, int=1900);
        void setDia(int);
        void setMes(int);
        void setAno(int);
        int getDia();
        int getMes();
        int getAno();
        int operator- ();
        int operator- (DataSobrecargaOperador &);
    private:
        int dia;
        int mes;
        int ano;
        int verificaDia(int); };
#endif

```

```
//conteúdo do arquivo dataSobrecargaOperador.cpp
```

```
#include <iostream>
```

```
using namespace std;
```

```
#include "dataSobrecargaOperador.h"
```

```
int DataSobrecargaOperador::operator- ()
```

```
{  
    DataSobrecargaOperador data(1,1,getAno());  
    return (*this)-data;  
}
```

```

int DataSobrecargaOperador::operator- (DataSobrecargaOperador &data)
{
    static const int diasPorMes[]={0,31,28,31,30,31,30,31,31,30,31,30,31};
    int dias1=0, dias2=0;
    for (int i=1900; i<getAno(); i++)
        dias1+=(i%400==0 || (i%4==0 && i%100!=0))?366:365;
    for (int i=1; i<getMes(); i++)
        dias1+=(i==2 && (getAno()%400==0 || (getAno()%4==0 &&
getAno()%100!=0)))?29:diasPorMes[i];
    dias1+=getDia();
    for (int i=1900; i<data.getAno(); i++)
        dias2+=(i%400==0 || (i%4==0 && i%100!=0))?366:365;
    for (int i=1; i<data.getMes(); i++)
        dias2+=(i==2 && (data.getAno()%400==0 || (data.getAno()%4==0 &&
data.getAno()%100!=0)))?29:diasPorMes[i];
    dias2+=data.getDia();
    return dias1-dias2;
}

```

```

//conteúdo do arquivo dataSobrecargaOperador.h
#ifndef DATASOBRECARGAOPERADOR_H
#define DATASOBRECARGAOPERADOR_H
    #include <iostream>
    class DataSobrecargaOperador {
        friend DataSobrecargaOperador operator+ (int,
DataSobrecargaOperador &);
        friend DataSobrecargaOperador operator+
(DataSobrecargaOperador &, int);
        friend ostream &operator<<(ostream &, DataSobrecargaOperador &);
        friend istream &operator>>(istream &, DataSobrecargaOperador &);
    public:
        DataSobrecargaOperador(int=1, int=1, int=1900);
        void setDia(int);
        void setMes(int);
        void setAno(int);
        int getDia();
        int getMes();
        int getAno();
        int operator- ();
        int operator- (DataSobrecargaOperador &);

```

```
private:
    int dia;
    int mes;
    int ano;
    int verificaDia(int);
    static const int numeroDeDiasNoMes[]; // vetor de dias por mês
    void incremento();
    bool fimDoMes( int );
    bool anoBissexto( int );
};
#endif
```

```
//conteúdo do arquivo dataSobrecargaOperador.cpp
```

```
#include <iostream>
```

```
using namespace std;
```

```
#include "dataSobrecargaOperador.h"
```

```
// inicializa membro static no escopo de arquivo
```

```
const int DataSobrecargaOperador::numeroDeDiasNoMes[] =  
    { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
```

```
int DataSobrecargaOperador::operator- ()
```

```
{  
    DataSobrecargaOperador data(1,1,getAno());  
    return (*this)-data;  
}
```

```

int DataSobrecargaOperador::operator- (DataSobrecargaOperador &data)
{
    int dias1=0, dias2=0;
    for (int i=1900; i<getAno(); i++)
        dias1+= anoBissextos(i)?366:365;
    for (int i=1; i<getMes(); i++)
        dias1+=(i==2 && (anoBissextos(getAno())))?29:numeroDeDiasNoMes[i];
    dias1+=getDia();
    for (int i=1900; i<data.getAno(); i++)
        dias2+= anoBissextos(i)?366:365;
    for (int i=1; i<data.getMes(); i++)
        dias2+=(i==2 && (anoBissextos(data.getAno())))?29:
numeroDeDiasNoMes[i];
    dias2+=data.getDia();
    return dias1-dias2;
}

```


Linguagem de Programação C++

Sobrecarga de operadores

Trataremos agora de um último detalhe sobre o tópico sobrecarga de operadores.

Vimos, que a escolha sobre qual implementação de um determinado operador sobrecarregado é feita com base na assinatura da função que o implementa. Contudo, como esta diferenciação se dará, por exemplo, no caso do operador ++ que poder ser pré ou pós-fixado?

A implementação prefixada se dará da mesma forma que fazemos a sobrecarga dos demais operadores unários. O problema reside na implementação pós-fixada.

Linguagem de Programação C++

Sobrecarga de operadores

Possíveis protótipos para o operador ++ prefixado seriam:

```
nomeDaClasse &operator++(); //função-membro
```

```
nomeDaClasse &operator++(nomeDaClasse &);
```

```
//função global
```

A solução adotada na linguagem C++, para possibilitar que o compilador diferencie a implementação pós-fixada da prefixada, foi a inserção de um parâmetro extra. Ou seja, gerando os seguintes protótipos:

```
nomeDaClasse &operator++(int); //função-membro
```

```
nomeDaClasse &operator++(nomeDaClasse &, int);
```

```
374 //função global
```

Linguagem de Programação C++

Para uma melhor fixação vamos analisar um exemplo de utilização da sobrecarga do operador ++ pós e prefixado.

Observação: Tudo que está sendo dito sobre o operador de incremento (++) vale para o operador de decremento (--).

Vamos imaginar a aplicação do operador de incremento sobre um objeto da classe DataSobrecargaOperador, o qual geraria o incremento de um dia na data em questão.

```

//conteúdo do arquivo dataSobrecargaOperador.h
#ifndef DATASOBRECARGAOPERADOR_H
#define DATASOBRECARGAOPERADOR_H
    #include <iostream>
    class DataSobrecargaOperador {
        friend DataSobrecargaOperador operator+ (int,
DataSobrecargaOperador &);
        friend DataSobrecargaOperador operator+
(DataSobrecargaOperador &, int);
        friend ostream &operator<<(ostream &, DataSobrecargaOperador &);
        friend istream &operator>>(istream &, DataSobrecargaOperador &);
    public:
        DataSobrecargaOperador(int=1, int=1, int=1900);
        void setDia(int);
        void setMes(int);
        void setAno(int);
        int getDia();
        int getMes();
        int getAno();
        int operator- ();
        int operator- (DataSobrecargaOperador &);
        // operador de incremento pré-fixado
        DataSobrecargaOperador &operator++();

```

```

// operador de incremento pós-fixado
DataSobrecargaOperador operator++( int );
private:
    int dia;
    int mes;
    int ano;
    int verificaDia(int);
    static const int numeroDeDiasNoMes[]; // vetor de dias por mês
    void incremento();
    bool fimDoMes( int );
    bool anoBissexto( int );
};
#endif

```

//conteúdo do arquivo dataSobrecargaOperador.cpp

```

...
// operador de incremento pré-fixado sobrecarregado
DataSobrecargaOperador &DataSobrecargaOperador::operator++()
{
    incremento(); // incrementa data
    return *this; // retorno de referência para criar um lvalue
}
377

```

```
// operador de incremento pós-fixado sobrecarregado;  
// observe que o parâmetro fictício do tipo inteiro não tem nem ao menos  
// um nome (identificador) de parâmetro  
DataSobrecargaOperador DataSobrecargaOperador::operator++( int )  
{  
    DataSobrecargaOperador temp = *this; // armazena o estado atual do  
objeto  
    incremento();  
    // retorna o objeto temporário, salvo, não-incrementado  
    return temp; // retorno de valor; não um retorno de referência  
}  
...
```

Linguagem de Programação C++

Com base no que estudamos é possível perceber que podemos definir uma classe String com inúmeras funcionalidades interessantes, com:

- possibilidade de atribuir uma string a outra com o operador =;
- possibilidade de comparar duas strings com o operador ==;
- possibilidade de concatenar duas strings com o operador +;
- possibilidade de alocar dinamicamente apenas memória necessária para armazenar a string;
- etc...

Linguagem de Programação C++

Podemos considerar a implementação desta classe String como um ótimo exercício de revisão.