OO – Engenharia Eletrônica

Orientação a Objetos
-
Programação em C++

Slides 8: Sobrecarga de métodos,
Const e Sobrecarga de operadores.

Prof. Jean Marcelo SIMÃO

Sobrecarga de métodos

```
#ifndef _PESSOA_H_
#define _PESSOA_H_

class Pessoa
{
protected:
   int diaP;
   int mesP;
   int anoP;
   int idadeP;
   char nomeP[30];

public:

   Pessoa(int diaNa, int mesNa, int anoNa, char* nome = " ");
   Pessoa();

   ~Pessoa();

   . . .
  // Sobrecarga de função NÃO é polimorfismo!

   void Calc_Idade(int diaAT, int mesAT, int anoAT);
   void Calc_Idade(int anoAT);

   int  informaIdade();

};
#endif
```

```
. . .
Pessoa::Pessoa(int diaNa, int mesNa, int anoNa, char* nome)
{
  Inicializa(diaNa, mesNa, anoNa, nome);
}

Pessoa::Pessoa()
{
}
. . .
void Pessoa::Calc_Idade(int diaAT, int mesAT, int anoAT)
{
   idadeP = anoAT - anoP;
    if ( mesP > mesAT )
    {
       idadeP = idadeP - 1;
    }
    else
    {
       if ( mesP == mesAT )
       {
           if ( diaP > diaAT )
           {
             idadeP = idadeP - 1;
           }
       }
    }
    printf("\n A idade da Pessoa %s é %d \n", nomeP, idadeP);
}

void Pessoa::Calc_Idade(int anoAT)
{
      idadeP = anoAT - anoP;
       printf("\n A idade da Pessoa %s é %d \n", nomeP, idadeP);
}
. . .
```

Const e Sobrecarga de Operadores

Classe *String*

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

int _tmain ( int argc, _TCHAR* argv[] )
{
    string s1 ( “bom dia! " ), s2;

    s2 = s1;                                // Atribui s1 a s2 com =

    cout << "S1: " << s1 << endl;
    cout << "S2: " << s2 << endl;
    cout << endl;

     s2[ 0 ] =  'B' ;                       // modifica S2

    cout << "S1: " << s1 << endl;
    cout << "S2: " << s2 << endl;
    cout << endl;

    int tam = s2.length();

    for (int x = 0; x < tam; ++x)
    {
        cout << s2 [ x ];                   // demonstrando o operador de colchetes
    }
    cout << endl;
}
```

Classe MinhaString

Versão 0

Classe MinhaString

Versão 0

Exemplo inspirado no livro dos Deitels:

- Deitel H. M., Deitel, P. J. "C++ Como Programar". 3a Edição Bookman, 2001.

```
#ifndef _MINHASTRING_H_
#define _MINHASTRING_H_

class MinhaString
{

public:
     MinhaString()
     {
        // falta inicializar atributos
     }

     MinhaString ( char s [ ] );

     ~MinhaString ( );

      char* getString ( );

private:

     int tamanho;

     char str[300]; // Vetor de caracteres.

};

#endif
```

```
#include "stdafx.h"
#include "MinhaString.h"

MinhaString::MinhaString ( char s [ ] )
{
    tamanho = strlen( s );

    strcpy ( str, s );

}

MinhaString::~MinhaString()
{
}

char* MinhaString::getString()
{
      return str;
}
```

```
// StringOperator.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include "MinhaString.h"

int _tmain ( int argc, _TCHAR* argv[] )
{



    MinhaString S1 ("Minha primeira string soh minha." );

    cout << S1.getString() << endl;

    return 0;

}
```

```
// StringOperator.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include "MinhaString.h"

int _tmain ( int argc, _TCHAR* argv[] )
{

   char Vet [100] = "Minha primeira string soh minha." ;

    MinhaString S1 ( Vet );

    cout << S1.getString() << endl;

    return 0;

}
```

```
// StringOperator.cpp : Defines the entry point for the console application.
//

#include "stdafx.h“   // aqui está incluído o “string.h”
#include "MinhaString.h"

int _tmain ( int argc, _TCHAR* argv[] )
{

    char Vet [100];

    strcpy( Vet, "Minha primeira string soh minha.");


    MinhaString S1 ( Vet );

    cout << S1.getString() << endl;

    return 0;

}
```

Classe MinhaString

Versão 1

Exemplo inspirado no livro dos Deitels:

- Deitel H. M., Deitel, P. J. "C++ Como Programar". 3a Edição Bookman, 2001.

```
#ifndef _MINHASTRING_H_
#define _MINHASTRING_H_

class MinhaString
{
public:
    MinhaString ( )
    {
        tamanho = 0;
        strcpy ( str, "" );
    }

    MinhaString ( const char s[] );

    ~MinhaString ( );

    char* getString ( );

private:

    int tamanho;

    char str[300]; // Vetor de caracteres.
};
#endif
```

```
#include "stdafx.h"
#include "MinhaString.h"

MinhaString::MinhaString (const char s[])
{

    tamanho = strlen( s );

    strcpy ( str, s );

}

MinhaString::~MinhaString()
{

}

char* MinhaString::getString()
{
     return str;
}
```

```
// StringOperator.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include "MinhaString.h"

int _tmain (int argc, _TCHAR* argv[])
{

    char Vet [100] = "Minha primeira string soh minha." ;

    MinhaString S1 ( Vet );

    cout << S1.getString() << endl;

    return 0;

}
```

Classe MinhaString

Versão 2

```
#ifndef _MINHASTRING_H_
#define _MINHASTRING_H_

class MinhaString
{
public:
     MinhaString ( )
     {
          tamanho = 0;
          strcpy ( str, "" );
     }

     MinhaString ( const char* s );

     ~MinhaString ( );

     char* getString ( );


private:

    int tamanho;

    char str[300]; // Vetor de caracteres.

};
#endif
```

```
#include "stdafx.h"
#include "MinhaString.h"

MinhaString::MinhaString (const char* s)
{

    tamanho = strlen( s );

    strcpy ( str, s );

}

MinhaString::~MinhaString()
{

}



char* MinhaString::getString()
{
     return str;
}
```

```
#ifndef _MINHASTRING_H_
#define _MINHASTRING_H_

class MinhaString
{
public:
     MinhaString ( )
     {
          tamanho = 0;
          strcpy ( str, "" );
     }

     MinhaString ( const char* s );

     ~MinhaString ( );

      char* getString ( );

private:

    int tamanho;

    char str[300]; // Vetor de caracteres.

};
#endif
```

```
#include "stdafx.h"
#include "MinhaString.h"

MinhaString::MinhaString (const char* s):
tamanho ( strlen( s ) )
{
   strcpy ( str, s );
}

MinhaString::~MinhaString()
{

}



char* MinhaString::getString()
{
     return str;
}
```

```
#ifndef _MINHASTRING_H_
#define _MINHASTRING_H_

class MinhaString
{
public:
     MinhaString ( )
     {
          tamanho = 0;
          strcpy ( str, "" );
     }

     MinhaString ( const char* s );

     ~MinhaString ( );

     char* getString ( );

private:

    int tamanho;

    char str[300]; // Vetor de caracteres.

};
#endif
```

```
#include "stdafx.h"
#include "MinhaString.h"

MinhaString::MinhaString (const char* s):
tamanho( strlen( s ) )
{
   strcpy ( str, s );
}

MinhaString::~MinhaString()
{

}


char* MinhaString::getString()
{
     return str;
}
```

```
// StringOperator.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include "MinhaString.h"

int _tmain (int argc, _TCHAR* argv[])
{

    MinhaString S1 ( "Minha primeira string soh minha." );

    cout << S1.getString() << endl;

    char*  pS = NULL;

    pS = S1.getString();

    pS[0] = 'T';   pS[1] = 'I'              // via operador de colchetes

    // *pS = 'T';  (*ps+1) = 'I';           // via aritmética de ponteiros.

    cout << S1.getString() << endl;

    return 0;

}
```

Classe MinhaString

Versão 3

```
#ifndef _MINHASTRING_H_
#define _MINHASTRING_H_

class MinhaString
{
public:
    MinhaString ( )
    {
        tamanho = 0;
        strcpy ( str, "" );
    }

    MinhaString ( const char* s );

    ~MinhaString ( );

    const char* getString ( );

private:

    int tamanho;

    char str[300];

};
#endif
```

```
#include "stdafx.h"
#include "MinhaString.h"

MinhaString::MinhaString (const char* s):
tamanho( strlen( s ) )
{
    strcpy ( str, s );
}

MinhaString::~MinhaString()
{

}

const char* MinhaString::getString()
{
    return str;
}
```

```
#ifndef _MINHASTRING_H_
#define _MINHASTRING_H_

class MinhaString
{
public:
     MinhaString ( )
     {
          tamanho = 0;
          strcpy ( str, "" );
     }

     MinhaString ( const char* s );

     ~MinhaString ( );

     const char* getString ( );

private:

    const int tamanho;

    char str[300];

};
#endif
```

```
#include "stdafx.h"
#include "MinhaString.h"

MinhaString::MinhaString (const char* s):
tamanho( strlen( s ) )
{
    strcpy ( str, s );
}

MinhaString::~MinhaString()
{

}

const char* MinhaString::getString()
{
     return str;
}
```

```
// StringOperator.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include "MinhaString.h"

int _tmain (int argc, _TCHAR* argv[])
{

    MinhaString S1 ( "Minha primeira string soh minha." );

    cout << S1.getString() << endl;

    const char*  pS;

    pS = S1.getString();

   // pS[0] = 'W';

    char vet2 [2000];
    strcpy ( vet2, pS );

    return 0;

}
```

Classe MinhaString

Versão 4

```
#ifndef _MINHASTRING_H_
#define _MINHASTRING_H_

class MinhaString
{
public:
     MinhaString ( )
     {
          tamanho = 0;
          strcpy ( str, "" );
     }

     MinhaString ( const char* s );

     ~MinhaString ( );

     const char* getString ( );

private:

     void setString( const char* s);

private:

     const int tamanho;

     char* pStr;   // Ponteiro para o ínicio do string.
};
#endif
```

```
#include "stdafx.h"
#include "MinhaString.h"

MinhaString::MinhaString (const char* s):
tamanho( strlen( s ) )
{
    setString ( s );
}

MinhaString::~MinhaString()
{
     delete [ ] pStr;
     pStr = NULL;
}

void MinhaString::setString ( const char* s )
{
      pStr = new char [ tamanho + 1 ];
      // pStr = malloc( (tamanho +1) * sizeof(char) )
      strcpy ( pStr, s );
}

const char* MinhaString::getString()
{
      return pStr;
}
```

```
// StringOperator.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include "MinhaString.h"

int _tmain (int argc, _TCHAR* argv[])
{

    MinhaString S1 ( "Minha primeira string soh minha." );

    cout << S1.getString() << endl;

    return 0;

}
```

Classe MinhaString

Versão 5

```
#ifndef _MINHASTRING_H_
#define _MINHASTRING_H_

class MinhaString
{
public:
   // MinhaString ( )
   // { tamanho = 0; strcpy ( str, "" ); }
    MinhaString (const char* s = "");
    ~MinhaString();
    const char* getString();

   // Sobre carga de operador
    void operator= (char* s);

private:
    void setString( const char* s );

private:
    int tamanho;
    char* pStr; // Ponteiro para o ínicio do string.
};

#endif
```

```
#include "stdafx.h"
#include "MinhaString.h"

MinhaString::MinhaString (const char* s):
tamanho ( strlen( s ) )
{
    setString ( s );
}

MinhaString::~MinhaString()
{
    delete [] pStr;
    pStr = NULL;
}

void MinhaString::setString (const char* s)
{
    pStr = new char[ tamanho + 1 ];
    strcpy ( pStr, s );
}

const char* MinhaString::getString()
{
    return pStr;
}

void MinhaString::operator= (char* s)
{
      delete [] pStr;
      tamanho = strlen ( s );
      setString ( s );
}
```

```
// StringOperator.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include "MinhaString.h"

int _tmain(int argc, _TCHAR* argv[])
{
    MinhaString S1 ( "Minha primeira string soh minha." );

    MinhaString S2;

    S2 = "Operador de atribuicao sobrecarregado é muito útil.";

    cout << S1.getString() << endl;

    cout << S2.getString() << endl;

    return 0;
}
```

```
// StringOperator.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include "MinhaString.h"

int _tmain(int argc, _TCHAR* argv[])
{
    MinhaString S1 ( "Minha primeira string soh minha." );

    MinhaString S2;

    S2.operator=("Operador de atribuicao sobrecarregado é muito útil.");

    cout << S1.getString() << endl;

    cout << S2.getString() << endl;

    return 0;
}
```

Classe MinhaString

Versao 6

```
#ifndef _MINHASTRING_H_
#define _MINHASTRING_H_
class MinhaString
{public:
    // MinhaString ( ) { }
    MinhaString (const char* s = "");
    ~MinhaString ( );
    const char* getString ( );

    // Sobre carga de operadores
    void operator = (const char* s);

    bool operator == (MinhaString s);

private:
    void setString ( const char* s );
private:
    int tamanho;
    char* pStr;
};
#endif
```

```
#include "stdafx.h"
#include "MinhaString.h"
MinhaString::MinhaString (const char* s) :
tamanho( strlen( s ) )
{
     setString( s );
}
MinhaString::~MinhaString()
{
     delete [] pStr;
    pStr = NULL;
}
```

```
void MinhaString::setString ( const char* s )
{
    pStr = new char [ tamanho + 1 ];
    strcpy ( pStr, s );
}

const char* MinhaString::getString ( )
 {
    return pStr;
}

void MinhaString::operator = (const char* s)
{
     if ( s != pStr )
    {
        delete [] pStr;
        tamanho = strlen (s);
        setString( s );
    }
}

bool MinhaString::operator == ( MinhaString s )
{
    if ( 0 == strcmp ( pStr, s.getString () ) )
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

```
// StringOperator.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include "MinhaString.h"

int _tmain(int argc, _TCHAR* argv[])
{
    MinhaString S1 ("Minha primeira string soh minha.");

    MinhaString S2A, S2B;

    S2A = "Operador de atribuicao sobrecarregado é muito útil.";
    S2B = "Operador de atribuicao sobrecarregado é muito útil.";


    cout << S1.getString() << endl;

    cout << S2A.getString() << endl;
    cout << S2B.getString() << endl;

     if ( S2A == S2B )
     {
                cout << "S2A eh igual a S2B!" << endl;
     }
     else
    {
                cout << "S2A NAO eh igual a S2B!" << endl;
    }

    return 0;
}
```

```
// StringOperator.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include "MinhaString.h"

int _tmain(int argc, _TCHAR* argv[])
{
   MinhaString S1 ("Minha primeira string soh minha.");

   MinhaString S2A, S2B;

   S2A = "Operador de atribuicao sobrecarregado é muito útil.";
   S2B = "Operador de atribuicao sobrecarregado é muito útil.";


   cout << S1.getString() << endl;

   cout << S2A.getString() << endl;
   cout << S2B.getString() << endl;

    if ( S2A.operator==(S2B ) )
    {
                cout << "S2A eh igual a S2B!" << endl;
    }
    else
   {
                cout << "S2A NAO eh igual a S2B!" << endl;
   }

   return 0;
}
```

Classe MinhaString

Versao 7

```
#ifndef _MINHASTRING_H_
#define _MINHASTRING_H_
class MinhaString
{public:
    //MinhaString ( ) { }
    MinhaString (const char* s = "");
    ~MinhaString ( );
    const char* getString ( );

    // Sobre carga de operadores
    void operator = (const char* s);

    bool operator == (const MinhaString s);

private:
    void setString ( const char* s );
private:
    int tamanho;
    char* pStr;
};
#endif
```

```
#include "stdafx.h"
#include "MinhaString.h"
MinhaString::MinhaString (const char* s) :
tamanho( strlen( s ) )
{
    setString( s );
}

MinhaString::~MinhaString()
{
    delete [] pStr;  pStr = NULL;
}
```

```
void MinhaString::setString ( const char* s )
{
    pStr = new char [ tamanho + 1 ];
    strcpy ( pStr, s );
}

const char* MinhaString::getString ( )
 {
    return pStr;
}

void MinhaString::operator = (const char* s)
{
   . . .
}

bool MinhaString::operator == (const MinhaString s )
{
    char* aux = new char [ strlen (s.getString()) + 1 ];
    strcpy ( aux, s.getString() );

    // if ( 0 == strcmp ( pStr, s.getString () ) )
    if ( 0 == strcmp ( pStr, aux ) )
    {
        return true;
    }
    else
    {
        return false;
    }

    delete [] aux;
}
```

```
// StringOperator.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include "MinhaString.h"

int _tmain(int argc, _TCHAR* argv[])
{
    MinhaString S1 ("Minha primeira string soh minha.");

    MinhaString S2A;
    const  MinhaString S2B("Operador de atribuicao sobrecarregado é muito útil);

    S2A = "Operador de atribuicao sobrecarregado é muito útil.";

    cout << S1.getString() << endl;

    cout << S2A.getString() << endl;
    cout << S2B.getString() << endl;

     if ( S2A == S2B )
     {
                cout << "S2A eh igual a S2B!" << endl;
     }
     else
    {
                cout << "S2A NAO eh igual a S2B!" << endl;
    }

    return 0;
}
```

Classe MinhaString

Versao 8 A

```
#ifndef _MINHASTRING_H_
#define _MINHASTRING_H_

class MinhaString
{
public:
    // MinhaString () { }
    MinhaString (const char* s = "");
    ~MinhaString();
    const char* getString();

    // Sobre carga de operadores
    void operator = (const char* s);
    void operator = (MinhaString  s);

    bool operator == (MinhaString s);

private:
     void setString( const char* s );

private:
     int tamanho;
     char* pStr;
};

#endif
```

```
. . .

void MinhaString::operator = (const char* s)
{
        if ( s != pStr )
        {
                delete [] pStr;
                tamanho = strlen (s);
                setString( s );
        }
}

void MinhaString::operator = (MinhaString s)
{
     operator = ( s.getString() );
}

bool MinhaString::operator == (MinhaString s)
{
     if ( 0 == strcmp ( pStr, s.getString() ) )
     {
                return true;
     }
     else
     {
                return false;
     }
}
```

// Cuidado, como *s* é uma cópia de outro objeto, ocorre também a cópia do ponteiro, o que leva a desalocar memória em s, afetando a string passada como parämetro. Isto porque, no final das contas, o ponteiro da cópia *s* aponta para o mesmo local que o ponteiro do objeto original

```
#ifndef _MINHASTRING_H_
#define _MINHASTRING_H_

class MinhaString
{
public:
    // MinhaString () { }
    MinhaString (const char* s = "");
    ~MinhaString();
    const char* getString();

    // Sobre carga de operadores
    void operator = (const char* s);
    void operator = (MinhaString  s);

    bool operator == (MinhaString s);

private:
     void setString( const char* s );

private:
     int tamanho;
     char* pStr;
};

#endif
```

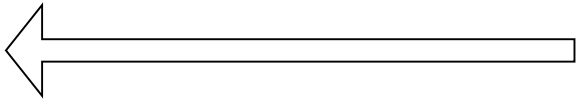
```
. . .

void MinhaString::operator = (const char* s)
{
        if ( s != pStr )
        {
                delete [] pStr;
                tamanho = strlen (s);
                setString( s );
        }
}

void MinhaString::operator = (MinhaString s)
{
      operator = ( s.getString() );
}

bool MinhaString::operator == (MinhaString s)
{
      if ( 0 == strcmp ( pStr, s.getString() ) )
      {
                return true;
      }
      else
     {
                return false;
     }
}
```

// Cuidado, como *s* é uma cópia de outro objeto, ocorre também a cópia do ponteiro, o que leva a desalocar memória em s, afetando a string passada como parâmetro. Isto porque, no final das contas, o ponteiro da cópia *s* aponta para o mesmo local que o ponteiro do objeto original

```
// StringOperator.cpp : Defines the entry point for the console application.

#include "stdafx.h"
#include "MinhaString.h"

int _tmain(int argc, _TCHAR* argv[])
{
    MinhaString S1 ( "Minha primeira string soh minha." );

    MinhaString S2A, S2B;

    S2A = "Operador de atribuicao sobrecarregado é muito útil.";
    // S2B = "Operador de atribuicao sobrecarregado é muito útil.";
    S2B = S2A;

    cout << S1.getString() << endl;

    cout << S2A.getString() << endl;
    cout << S2B.getString() << endl;

    if ( S2A == S2B )
    {
            cout << "S2A eh igual a S2B!" << endl;
    }
    else
    {
            cout << "S2A NAO eh igual a S2B!" << endl;
    }
    return 0;
}
```

```
// StringOperator.cpp : Defines the entry point for the console application.

#include "stdafx.h"
#include "MinhaString.h"

int _tmain(int argc, _TCHAR* argv[])
{
    MinhaString S1 ( "Minha primeira string soh minha." );

    MinhaString S2A, S2B;

    S2A = "Operador de atribuicao sobrecarregado é muito útil.";
    // S2B = "Operador de atribuicao sobrecarregado é muito útil.";
    S2B.operator=( S2A );

    cout << S1.getString() << endl;

    cout << S2A.getString() << endl;
    cout << S2B.getString() << endl;

    if ( S2A == S2B )
    {
            cout << "S2A eh igual a S2B!" << endl;
    }
    else
    {
            cout << "S2A NAO eh igual a S2B!" << endl;
    }
    return 0;
}
```

```
// StringOperator.cpp : Defines the entry point for the console application.

#include "stdafx.h"
#include "MinhaString.h"

int _tmain(int argc, _TCHAR* argv[])
{
    MinhaString S1 ( "Minha primeira string soh minha." );

    MinhaString S2A, S2B;

    S2A.operator= ("Operador de atribuicao sobrecarregado é muito útil.");
    // S2B = "Operador de atribuicao sobrecarregado é muito útil.";
    S2B.operator=( S2A );

    cout << S1.getString() << endl;

    cout << S2A.getString() << endl;
    cout << S2B.getString() << endl;

    if ( S2A.operator== ( S2B ) )
    {
            cout << "S2A eh igual a S2B!" << endl;
    }
    else
    {
            cout << "S2A NAO eh igual a S2B!" << endl;
    }
    return 0;
}
```



// Cuidado, como *s* é uma cópia de S2B, ocorre também a cópia do ponteiro, o que leva a desalocar memória em s, afetando S2B no tocante ao ponteiro. Isto porque, no final das contas, o ponteiro da cópia aponta para o mesmo local que o ponteiro do objeto original

A T E N Ç Ã O !

Ao utilizar o atributo pStr alocado dinamicamente, deve-se tomar cuidado para que nenhuma função receba a *string* em passagem por valor. Isto porque a variável estática criada irá copiar o endereço daquela passada como parâmetro e terá sua memória alocada dinâmicamente desalocada pela sua destrutora ao sair do seu escopo.

De uma forma mais simples:

Ao trabalhar com classes que possuem ponteiros como um de seus atributos (ou memória dinâmicamente alocada), **não utilize passagem por valor.**

Classe MinhaString

Versao 8 B

```
#ifndef _MINHASTRING_H_
#define _MINHASTRING_H_

class MinhaString
{
public:
    // MinhaString () { }
    MinhaString (const char* s = "");
    ~MinhaString();
    const char* getString();

    // Sobre carga de operadores
    void operator = (const char* s);
    void operator = (MinhaString* s);

    bool operator == (MinhaString* s);

private:
     void setString( const char* s );

private:
     int tamanho;
     char* pStr;
};

#endif
```

```
. . .

void MinhaString::operator = (const char* s)
{
        if ( s != pStr )
        {
                delete [] pStr;
                tamanho = strlen (s);
                setString( s );
        }
}

void MinhaString::operator = (MinhaString* s)
{
      operator = ( s->getString() );
}

bool MinhaString::operator == (MinhaString* s)
{
      if ( 0 == strcmp ( pStr, s->getString() ) )
      {
                return true;
      }
      else
     {
                return false;
     }
}
```

```
// StringOperator.cpp : Defines the entry point for the console application.

#include "stdafx.h"
#include "MinhaString.h"

int _tmain(int argc, _TCHAR* argv[])
{
    MinhaString S1 ( "Minha primeira string soh minha." );

    MinhaString S2A, S2B;

    S2A = "Operador de atribuicao sobrecarregado é muito útil.";
    // S2B = "Operador de atribuicao sobrecarregado é muito útil.";
    S2B = &S2A;

    cout << S1.getString() << endl;

    cout << S2A.getString() << endl;
    cout << S2B.getString() << endl;

    if ( S2A == &S2B )
    {
            cout << "S2A eh igual a S2B!" << endl;
    }
    else
    {
            cout << "S2A NAO eh igual a S2B!" << endl;
    }
    return 0;
}
```

```
// StringOperator.cpp : Defines the entry point for the console application.

#include "stdafx.h"
#include "MinhaString.h"

int _tmain(int argc, _TCHAR* argv[])
{
    MinhaString S1 ( "Minha primeira string soh minha." );

    MinhaString S2A, S2B;

    S2A = "Operador de atribuicao sobrecarregado é muito útil.";
    // S2B = "Operador de atribuicao sobrecarregado é muito útil.";
    S2B.operator=( &S2A );

    cout << S1.getString() << endl;

    cout << S2A.getString() << endl;
    cout << S2B.getString() << endl;

    if ( S2A.operator=( &S2B ) )
    {
            cout << "S2A eh igual a S2B!" << endl;
    }
    else
    {
            cout << "S2A NAO eh igual a S2B!" << endl;
    }
    return 0;
}
```

Classe MinhaString

Versao 8 C

```
#ifndef _MINHASTRING_H_
#define _MINHASTRING_H_

class MinhaString
{
public:
    // MinhaString () { }
    MinhaString (const char* s = "");
    ~MinhaString();
    const char* getString();

    // Sobre carga de operadores
    void operator = (const char* s);
    void operator = (MinhaString& s);

    bool operator == (MinhaString& s);

private:
     void setString( const char* s );

private:
     int tamanho;
     char* pStr;
};

#endif
```

```
. . .

void MinhaString::operator = (const char* s)
{
        if ( s != pStr )
        {
                delete [] pStr;
                tamanho = strlen (s);
                setString( s );
        }
}

void MinhaString::operator = (MinhaString& s)
{
      operator = ( s.getString() );
}

bool MinhaString::operator == (MinhaString& s)
{
      if ( 0 == strcmp ( pStr, s.getString() ) )
      {
                return true;
      }
      else
     {
                return false;
     }
}
```

```
// StringOperator.cpp : Defines the entry point for the console application.

#include "stdafx.h"
#include "MinhaString.h"

int _tmain(int argc, _TCHAR* argv[])
{
    MinhaString S1 ( "Minha primeira string soh minha." );

    MinhaString S2A, S2B;

    S2A = "Operador de atribuicao sobrecarregado é muito útil.";
    // S2B = "Operador de atribuicao sobrecarregado é muito útil.";
    S2B = S2A;

    cout << S1.getString() << endl;

    cout << S2A.getString() << endl;
    cout << S2B.getString() << endl;

    if ( S2A == S2B )
    {
            cout << "S2A eh igual a S2B!" << endl;
    }
    else
    {
            cout << "S2A NAO eh igual a S2B!" << endl;
    }
    return 0;
}
```

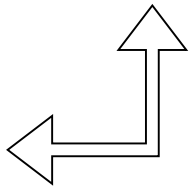
```
// StringOperator.cpp : Defines the entry point for the console application.

#include "stdafx.h"
#include "MinhaString.h"

int _tmain(int argc, _TCHAR* argv[])
{
    MinhaString S1 ( "Minha primeira string soh minha." );

    MinhaString S2A, S2B;

    S2A = "Operador de atribuicao sobrecarregado é muito útil.";
    // S2B = "Operador de atribuicao sobrecarregado é muito útil.";
    S2B.operator=( S2A );

    cout << S1.getString() << endl;

    cout << S2A.getString() << endl;
    cout << S2B.getString() << endl;

    if ( S2A.operator== ( S2B ) )
    {
            cout << "S2A eh igual a S2B!" << endl;
    }
    else
    {
            cout << "S2A NAO eh igual a S2B!" << endl;
    }
    return 0;
}
```

Classe MinhaString

Versao 9

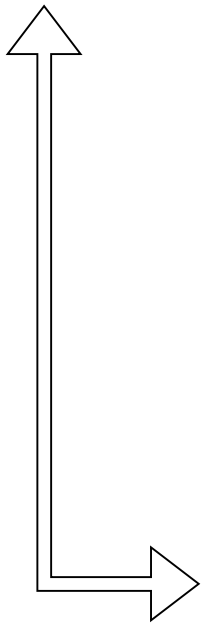
```
#ifndef _MINHASTRING_H_
#define _MINHASTRING_H_

class MinhaString
{
public:
    // MinhaString () { }
    MinhaString (const char* s = "");
    ~MinhaString ( );
    const char* getString();

     // Sobre carga de operadores
     void operator = (const char* s);
     void operator = (MinhaString& s);
     bool operator == (MinhaString& s);

private:
     void setString ( const char* s );

private:
     int tamanho;
     char* pStr;
};

ostream&  operator<< (ostream& saida, MinhaString& s);

#endif
```

```
int main ( )
{
    string s1 ( “bom dia! " )

    int i1 = 1;

    float f2  = 2.1;


    cout << "vars:" << s1 << i1 << f2 << endl;


    return 0;
}
```

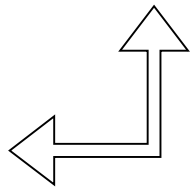


ostream& operator<< (ostream& saida, ?& p);

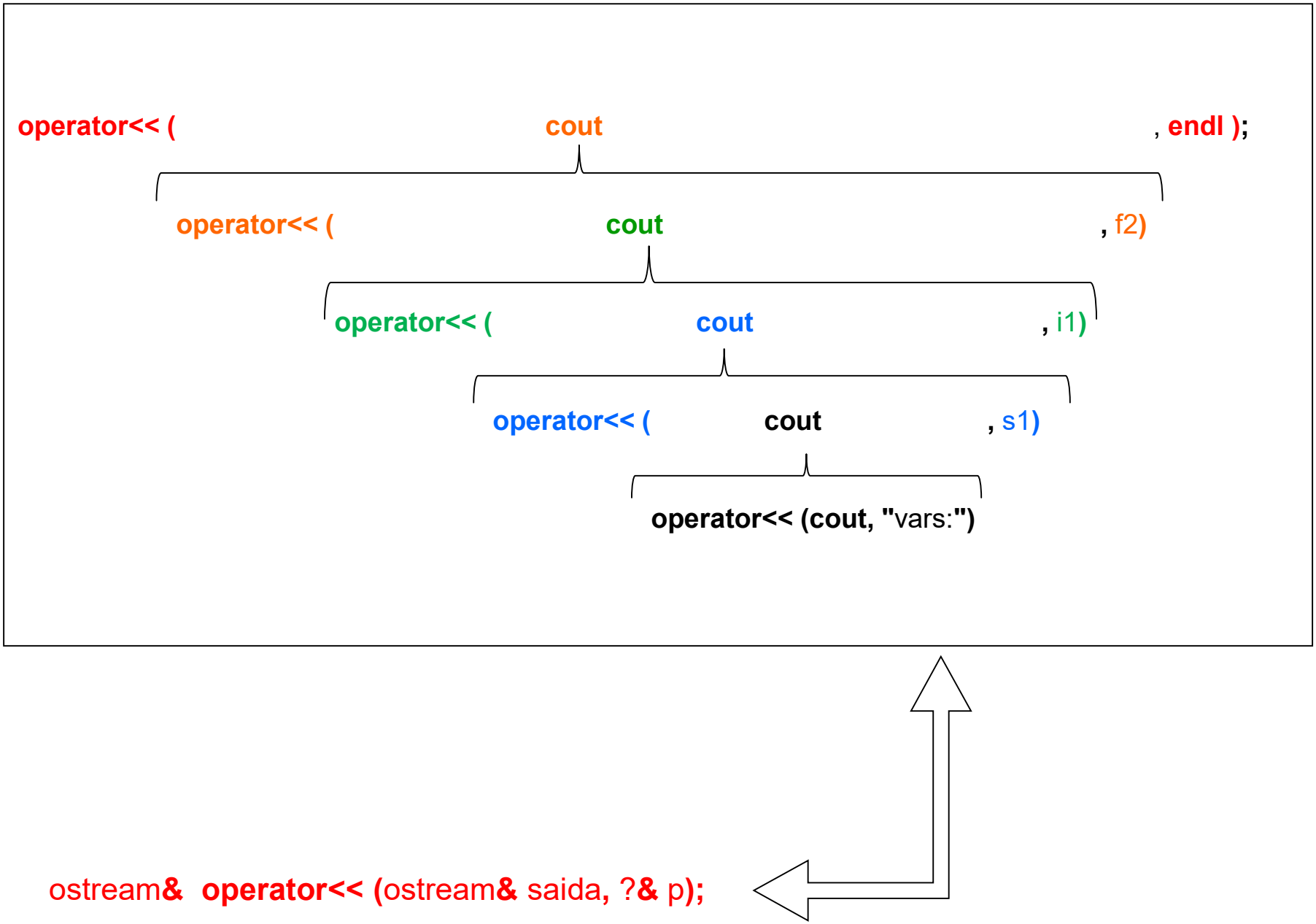
```
operator<< (operator<< (operator<< (operator<< (operator<< (cout, "vars:") , s1) , i1) , f2) , endl);
```



```
int main ( )
{
    string s1 ( “bom dia! " )

    int i1 = 1;

    float f2  = 2.1;


    cout << "vars:" << s1 << i1 << f2 << endl;


    return 0;
}
```



```
ostream&  operator<< (ostream& saida, ?& p);
```


operator<< (cout , endl);
operator<< (cout , f2)
operator<< (cout , i1)
operator<< (cout , s1)
operator<< (cout, "vars:")
ostream& operator<< (ostream& saida, ?& p);

```
#ifndef _MINHASTRING_H_
#define _MINHASTRING_H_

class MinhaString
{
public:
   // MinhaString () { }
   MinhaString (const char* s = "");
   ~MinhaString ( );
   const char* getString();

    // Sobre carga de operadores
    void operator = (const char* s);
    void operator = (MinhaString& s);
    bool operator == (MinhaString& s);

private:
    void setString ( const char* s );

private:
    int tamanho;
    char* pStr;
};

ostream&  operator<< (ostream& saida, MinhaString& s);

#endif
```

```
...

void MinhaString::operator = ( MinhaString& s )
{
      operator = ( s.getString() );
}

bool MinhaString::operator == ( MinhaString& s )
{
      if ( 0 == strcmp ( pStr, s.getString() ) )
      {
            return true;
      }
      else
      {
            return false;
      }
}

// Operador de Saída sobrecarregado.

ostream &operator<< ( ostream &saida, MinhaString& s )
{
            saida << s.getString();
            return saida;   // possibilita encadeamento
}
```

```
#include "stdafx.h"
#include "MinhaString.h"
int _tmain( int argc, _TCHAR* argv[] )
{
    MinhaString S1 ( "Minha primeira string soh minha." );
    MinhaString S2A, S2B;

     S2A = "Operador de atribuicao sobrecarregado é muito útil.";
     S2B = S2A;

     // cout << S1.getString() << endl;
     cout << S1 << endl;

     cout << S2A << S2B << endl;

      cout << endl;
      if ( S2A == S2B )
      {
              cout << "S2A eh igual a S2B!" << endl << endl;
      }
     else
     {
              cout << "S2A NAO eh igual a S2B!" << endl << endl;
     }

     cout << endl;
     if ( S1 == S2B  )
     {
            cout << "S1 eh igual a S2B!" << endl;
      }
     else
     {
           cout << "S1 NAO eh igual a S2B!" << endl;
     }
     return 0;
}
```

```
#include "stdafx.h"
#include "MinhaString.h"

int _tmain(int argc, _TCHAR* argv[])
{
   MinhaString S1 ( "Minha primeira string soh minha." );

   MinhaString S2A, S2B;

   S2A = "Operador de atribuicao sobrecarregado é muito útil.";
   S2B = S2A;

   cout << S1   << endl;
   cout << S2A << endl;
   cout << S2B << endl;
   cout << endl;

   // Operador ternário
   // Se (Expressão) Então (ComandosA) Senão (ComandosB)
   // (Expressão) ? (ComandosA) : (ComandosB)

   ( S2A == S2B ) ? ( cout << "S2A eh igual a S2B!" ) : ( cout << "S2A NAO eh igual a S2B!" );

   cout << endl << endl;

   ( S1  == S2B ) ? ( cout << "S1 eh igual a S2B!" ) : ( cout << "S1 NAO eh igual a S2B!" );

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

Exercícios

- Na classe *MinhaString*:
  - Sobrecarregue o operador != .
  - Sobrecarregue o operador + .
  - Sobrecarregue o operador - .
  - Sobrecarregue o operador [ ] .
  - Sobrecarregue o operador >> .
- No projeto que desenvolvemos antes em sala (o das universidades), utilize a classe *MinhaString* no lugar de *char**.

Itens para estudar:

- Espaço de nomes (*namespace*).
- Classes Aninhadas.
- Classe Pré-definida *String*.
- Objetos Constantes (*const*)...
- Atributos e métodos estáticos (*static*).