

# 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...

- Polimorfismo
- Mais exemplos de programação orientada a objetos...

# Templates

- Funções templates
  - Especificam um conjunto completo de funções (sobrecarregada) relacionadas
    - Cada uma é uma função template especializada
- Classes templates
  - Especificam um conjunto completo de classes relacionadas
    - Cada uma é uma classe template especializada

# Funções Templates

- Funções sobrecarregadas
  - Operações similares ou idênticas
    - Tipos diferentes de dados

## Operações similares e tipos diferentes

```
void print (int a) { cout << "Inteiro " << a ; }  
void print (double a) { cout << "Double " << a ; }
```

## Operações idênticas e tipos diferentes

```
void print (int a) { cout << a; }  
void print (double a) { cout << a; }
```

# Funções Templates

- Se as operações forem idênticas para cada tipo...
  - Funções podem ser escritas de maneira mais compacta
    - **Funções template**

```
void print (T a) { cout << "T " << a ; },  
    T pode ser int ou double!
```

# Funções Template

- Funções templates
  - Operações idênticas
    - Tipos diferentes de dados
  - Template de função única
    - Compilador gera código objeto para cada função utilizada em separado
  - Checagem de tipo
    - Diferente das macros em C

```
#define min(X, Y) ((X) < (Y) ? (X) : (Y))
```

# Funções Template

- Definição de funções template
  - Palavra-chave: `template`
  - Lista tipos formais de parâmetros em parênteses angulares (`< e >`)
    - Precedido pela palavra-chave `class` ou `typename`
      - `class` e `typename` podem ser intercalados
- Especifica tipos de:
  - Argumentos para funções, tipo de retorno de função e variáveis dentro da função

# Primeiro Exemplo de Programa Usando Template em C++

```
/*
 * Aula 14 - Exemplo 1
 * Programa Principal
 * Autor: Miguel Campista
 */
#include <iostream>
using namespace std;

// Definição da função template printArray
template <class T>
void printArray (const T *array, const int count) {
    for (int i = 0; i < count; i++)
        cout << array [i] << ' ';
    cout << endl;
}

int main () {
    const int aCount = 5;
    const int bCount = 7;
    const int cCount = 6;

    int a [aCount] = {1, 2, 3, 4, 5};
    double b [bCount] = {1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7};
    char c [cCount] = "Hello";
}
```

# Primeiro Exemplo de Programa Usando Template em C++

```
cout << "Array a contains: " << endl;
// Chama a especialização de função template de inteiro
printArray (a, aCount);

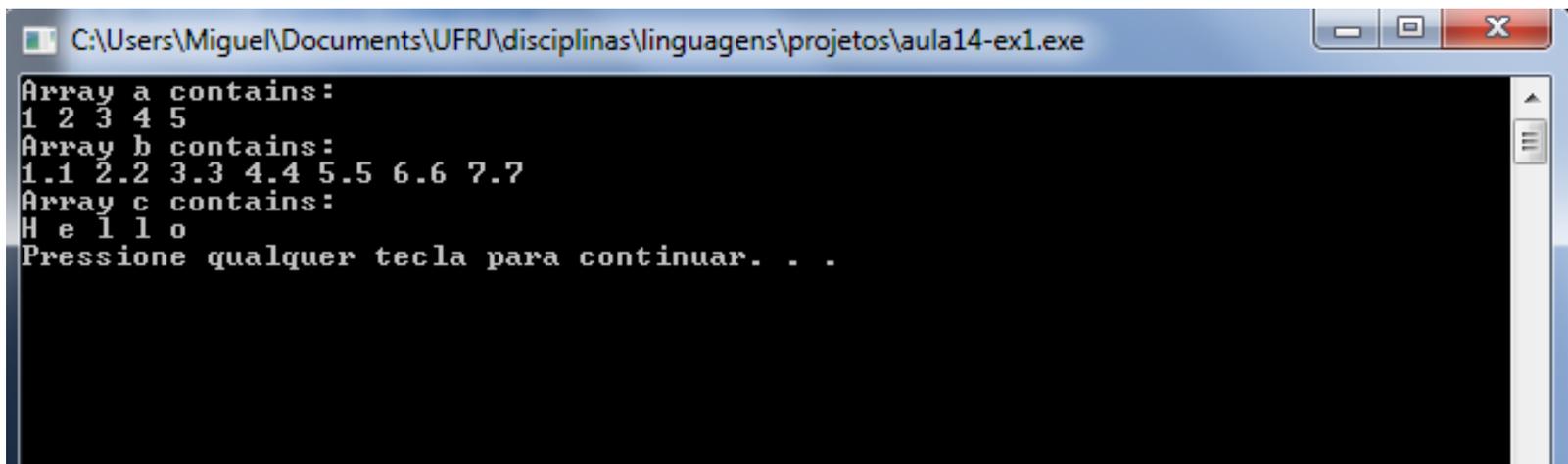
cout << "Array b contains:" << endl;
// Chama a especialização de função template de double
printArray( b, bCount );

cout << "Array c contains:" << endl;
// Chama a especialização de função template de char
printArray( c, cCount );

return 0;
}
```

# Primeiro Exemplo de Programa Usando Template em C++

```
cout << "Array a contains: " << endl;  
// Chama a especialização de função template de inteiro  
printArray (a, aCount);
```



```
C:\Users\Miguel\Documents\UFRJ\disciplinas\linguagens\projetos\aula14-ex1.exe  
Array a contains:  
1 2 3 4 5  
Array b contains:  
1.1 2.2 3.3 4.4 5.5 6.6 7.7  
Array c contains:  
H e l l o  
Pressione qualquer tecla para continuar. . .
```

# Templates de Funções Sobrecarregadas

- Especialização de funções template relacionadas
  - Compilador usa resolução sobrecarregada para identificar a função que melhor se enquadra com a chamada no código fonte
  - Compilador deduz a substituição que deve ser feita entre o tipo do parâmetro do template e o tipo do parâmetro da chamada da função
  - Compilador compila a versão especializada que atenda a chamada no código fonte
    - No exemplo anterior, três versões especializadas foram criadas para `printArray`

# Templates de Funções Sobrecarregadas

- Função template sobrecarregada
  - Outras funções templates com o mesmo nome
    - Parâmetros diferentes

```
printArray (const T *array, int count) e  
printArray (const T *array, int count, int lowSubscript)
```

- Funções que não são templates com o mesmo nome
  - Argumentos diferentes de função

```
printArray (const T *array, int count) e  
printArray (const char *array, double count)
```

# Templates de Funções Sobrecarregadas

- Função template sobrecarregada
  - Compilador realiza processo de identificação de padrão
    - Tenta achar o mesmo padrão do nome da função e dos tipos de argumentos
      - Compilador procura a função mais próxima da função chamada
        - » Ao encontrar, a utiliza
    - Se falhar,
      - Função template correspondente não é encontrada ou se houver mais de uma função que atende às características
        - » Compilador gera um erro

# Classe Template

- Pilha (*stack*)
  - Estrutura LIFO (*Last-In-First-Out*)
- Classes templates
  - Programação genérica
  - Descreve pilha genericamente
    - Instanciação de versão de tipo específico
  - Tipos parametrizados
    - Requerem um ou mais tipos de parâmetros
      - Personaliza template de "classe genérica" para formar classe template especializada

```

/*
 * Aula 14 - Exemplo 2
 * Arquivo pilhaCap14Ex2.h
 * Autor: Miguel Campista
 */
#ifndef TSTACK1_H
#define TSTACK1_H

using namespace std;

template <class T>
class Stack {
public:
    Stack (int = 10); // Construtor padrão

    // Destrutor
    ~Stack () {
        delete [] stackPtr;
    }

    bool push (const T&); // insere um elemento na pilha
    bool pop (T&);        // retira um elemento da pilha

    // Verifica se a Pilha está vazia
    bool isEmpty () const {
        return top == -1;
    }

    // Verifica se a Pilha está cheia
    bool isFull () const {
        return top == size - 1;
    }

private:
    int size;
    int top;
    T *stackPtr;
};

```

```

/*
 * Aula 14 - Exemplo 2
 * Arquivo pilhaCap14Ex2.h
 * Autor: Miguel Campista
 */
#ifndef TSTACK1_H
#define TSTACK1_H

using namespace std;

template <class T>
class Stack {
public:
    Stack (int = 10); // Construtor padrão

    // Destrutor
    ~Stack () {

```

**Classes template são precedidas pelo cabeçalho  
template <class T>**

```

    // Verifica se a Pilha está vazia
    bool isEmpty () const {
        return top == -1;
    }

    // Verifica se a Pilha está cheia
    bool isFull () const {
        return top == size - 1;
    }

private:
    int size;
    int top;
    T *stackPtr;
};

```

# Segundo Exemplo de Programa Usando Template em C++

```
template <class T>
Stack <T>::Stack (int s) { // Construtor padrão
    size = s > 0 ? s : 10;
    top = -1;
    stackPtr = new T [size];
}

// insere elemento na pilha
// se bem sucedido, retorna verdadeiro; senão, retorna falso
template <class T>
bool Stack <T>::push (const T &pushValue) {
    if ( !isFull() ) {
        stackPtr [++top] = pushValue; // coloca item na Pilha
        return true; // inserção bem sucedida
    }
    return false; // inserção mal sucedida
}

// retira elemento da pilha
// se bem sucedido, retorna verdadeiro; senão, retorna falso
template <class T>
bool Stack <T>::pop( T &popValue ) {
    if ( !isEmpty() ) {
        popValue = stackPtr[ top-- ]; // remove item da pilha
        return true; // remoção bem sucedida
    }
    return false; // remoção mal sucedida
}

#endif
```

# Segundo Exemplo de Programa Usando Template em C++

```
template <class T>
Stack <T>::Stack (int s) { // Construtor padrão
    size = s > 0 ? s : 10;
    top = -1;
    stackPtr = new T [size];
}

// insere elemento na pilha
```

Funções membro de classe template são também funções template, portanto devem ser precedidas por `template <class T>`

O operador de resolução de escopo (`Stack <T>`) é utilizado para associar as funções membro ao escopo da classe template

```
return true; // inserção mal sucedida
}

return false; // remoção mal sucedida
}

#endif
```

```
/*
 * Aula 14 - Exemplo 2
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include "pilhaCap14Ex2.h"

int main() {
    Stack <double> doubleStack (5);
    double doubleValue = 1.1;

    cout << "Pushing elements onto doubleStack\n";

    while (doubleStack.push (doubleValue)) {
        cout << doubleValue << ' ';
        doubleValue += 1.1;
    }

    cout << "\nStack is full. Cannot push " << doubleValue
        << "\n\nPopping elements from doubleStack\n";

    while (doubleStack.pop (doubleValue))
        cout << doubleValue << ' ';

    cout << "\nStack is empty. Cannot pop\n";

    Stack< int > intStack;
    int intValue = 1;
    cout << "\nPushing elements onto intStack\n";

    while (intStack.push (intValue)) {
        cout << intValue << ' ';
        ++intValue;
    }
}
```

# Segundo Exemplo de Programa Usando Template em C++

```
    cout << "\nStack is full. Cannot push " << intValue
         << "\n\nPopping elements from intStack\n";

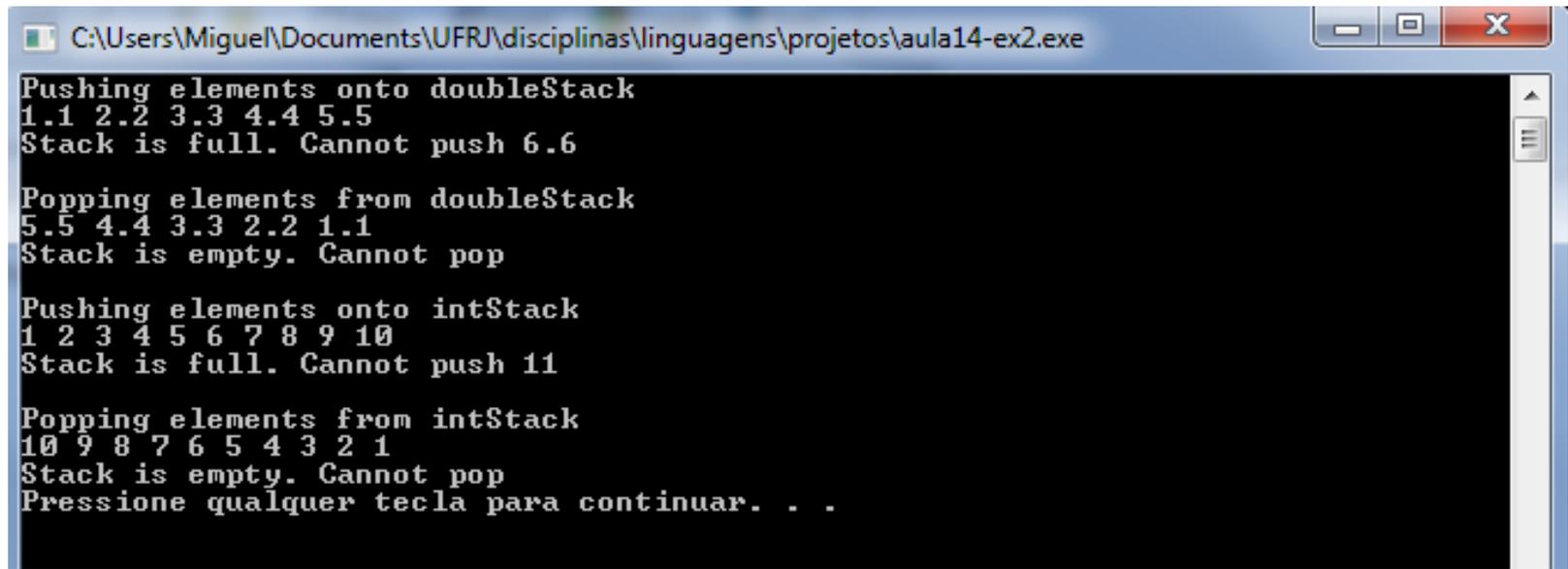
    while (intStack.pop (intValue))
        cout << intValue << ' ';

    cout << "\nStack is empty. Cannot pop\n";

    return 0;
}
```

# Segundo Exemplo de Programa Usando Template em C++

```
cout << "\nStack is full. Cannot push " << intValue  
<< "\n\nPopping elements from intStack\n";
```



```
C:\Users\Miguel\Documents\UFRJ\disciplinas\linguagens\projetos\aula14-ex2.exe  
Pushing elements onto doubleStack  
1.1 2.2 3.3 4.4 5.5  
Stack is full. Cannot push 6.6  
  
Popping elements from doubleStack  
5.5 4.4 3.3 2.2 1.1  
Stack is empty. Cannot pop  
  
Pushing elements onto intStack  
1 2 3 4 5 6 7 8 9 10  
Stack is full. Cannot push 11  
  
Popping elements from intStack  
10 9 8 7 6 5 4 3 2 1  
Stack is empty. Cannot pop  
Pressione qualquer tecla para continuar. . .
```

# Terceiro Exemplo de Programa Usando Template em C++

- Já que as operações realizadas sobre as pilhas de doubles e de inteiros foram as mesmas...
  - Encher e depois esvaziar a pilha



**Cria-se uma função template para realizar a mesma sequência de operações independente do tipo dos dados inseridos na pilha**

# Terceiro Exemplo de Programa Usando Template em C++

```
/*
 * Aula 14 - Exemplo 3
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include "pilhaCap14Ex3.h"

// função template para manipular Stack< T >
template< class T >
void testStack(
    Stack< T > &theStack,    // referência para Stack< T >
    T value,                // valor inicial para inserir
    T increment,           // incremento para valores subsequentes
    const char *stackName ) // nome do objeto Stack < T >
{
    cout << "\nPushing elements onto " << stackName << '\n';

    while ( theStack.push( value ) ) {
        cout << value << ' ';
        value += increment;
    }

    cout << "\nStack is full. Cannot push " << value
         << "\n\nPopping elements from " << stackName << '\n';

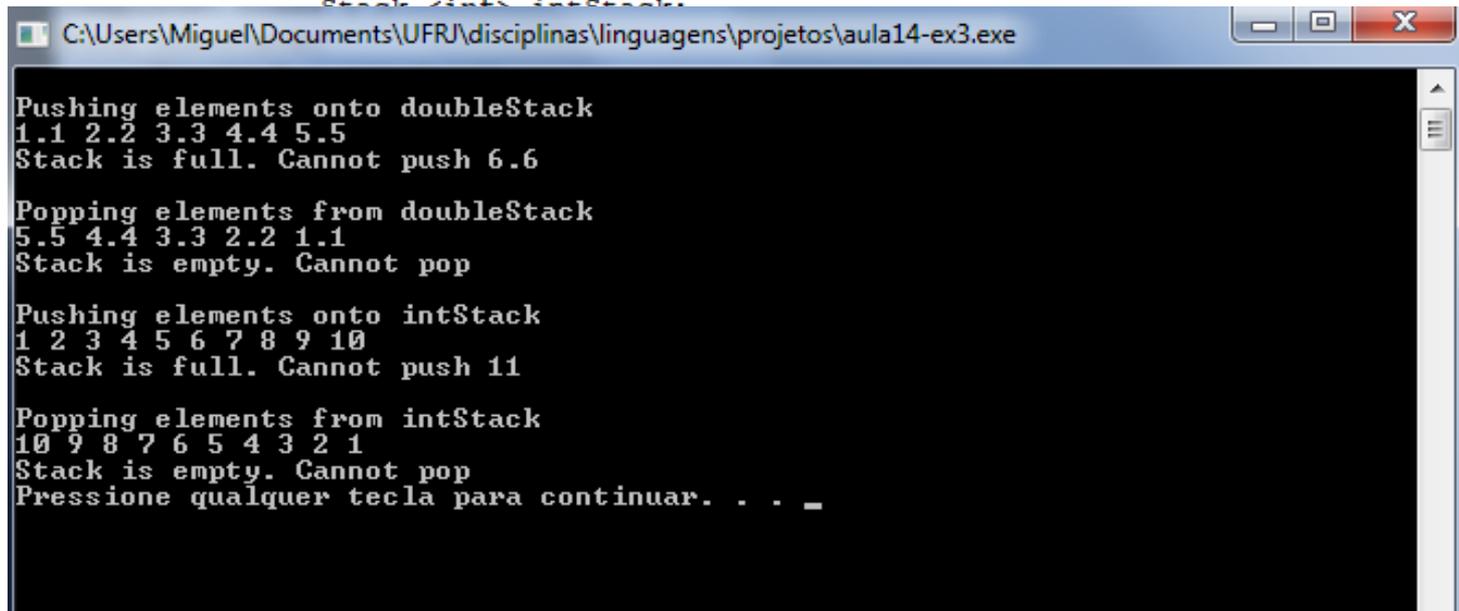
    while ( theStack.pop( value ) )
        cout << value << ' ';
    cout << "\nStack is empty. Cannot pop\n";
}
```

# Terceiro Exemplo de Programa Usando Template em C++

```
int main() {  
    Stack <double> doubleStack (5);  
    Stack <int> intStack;  
  
    testStack( doubleStack, 1.1, 1.1, "doubleStack" );  
    testStack( intStack, 1, 1, "intStack" );  
  
    return 0;  
}
```

# Terceiro Exemplo de Programa Usando Template em C++

```
int main() {  
    Stack <double> doubleStack (5);  
    Stack <int> intStack;
```



```
C:\Users\Miguel\Documents\UFRJ\disciplinas\linguagens\projetos\aula14-ex3.exe  
Pushing elements onto doubleStack  
1.1 2.2 3.3 4.4 5.5  
Stack is full. Cannot push 6.6  
  
Popping elements from doubleStack  
5.5 4.4 3.3 2.2 1.1  
Stack is empty. Cannot pop  
  
Pushing elements onto intStack  
1 2 3 4 5 6 7 8 9 10  
Stack is full. Cannot push 11  
  
Popping elements from intStack  
10 9 8 7 6 5 4 3 2 1  
Stack is empty. Cannot pop  
Pressione qualquer tecla para continuar. . . _
```

# Classe Template e Parâmetros sem Tipo

- Classes templates

- Parâmetros template sem tipo

- Argumentos padrão
    - Tratados como `const`'s

- Ex.:

```
template< class T, int elements >
```

```
Stack< double, 100 > mostRecentSalesFigures;
```

- Declara objeto do tipo `Stack< double, 100>` pilha
      - Na classe poderia ser declarado um array `a [elements]`;

- Também podem ser usados em funções

# Classe Template e Parâmetros sem Tipo

```
/*
 * Aula 14 - Exemplo 6
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>

using namespace std;

template <typename T, int elementos>
void printArray (T *a) {
    for (int i = 0; i < elementos; i++)
        cout << a [i] << endl;
}

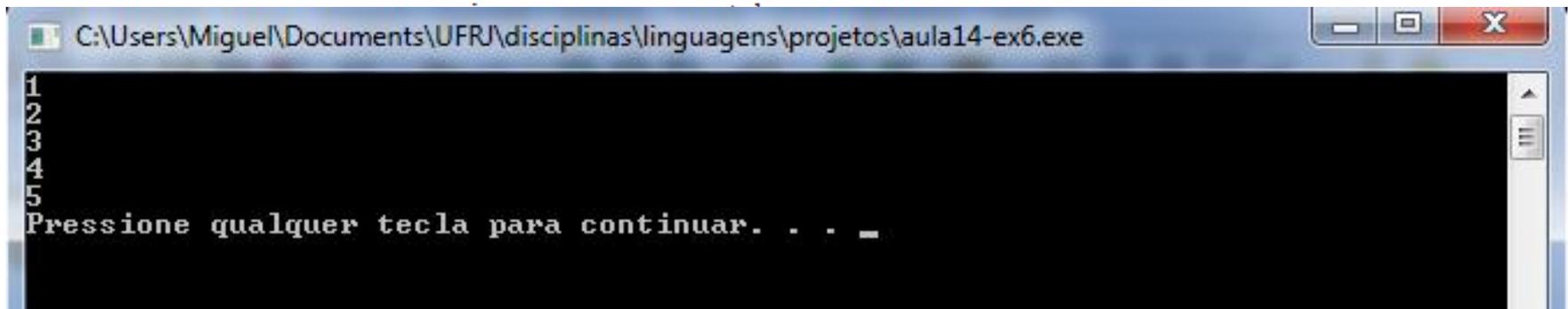
int main(int argc, char *argv[])
{
    int array [] = {1, 2, 3, 4, 5};

    printArray <int, 5> (array);

    return 0;
}
```

# Classe Template e Parâmetros sem Tipo

```
/*  
 * Aula 14 - Exemplo 6  
 * Arquivo Principal  
 * Autor: Miguel Campista  
 */  
#include <iostream>
```



```
C:\Users\Miguel\Documents\UFRJ\disciplinas\linguagens\projetos\aula14-ex6.exe  
1  
2  
3  
4  
5  
Pressione qualquer tecla para continuar. . . _
```

```
{  
    int array [] = {1, 2, 3, 4, 5};  
  
    printArray <int, 5> (array);  
  
    return 0;  
}
```

# Classe Template e Parâmetros sem Tipo

```
/*
 * Aula 14 - Exemplo 6
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>

using namespace std;

template <typename T, int elementos>
void printArray (T *a) {
    for (int i = 0; i < elementos; i++)
        cout << a [i] << endl;
}

int main(int argc, char *argv[])
{
    int array [] = {1, 2, 3, 4, 5};
    int elementos = 5;

    printArray <int, elementos> (array);

    return 0;
}
```

# Classe Template e Parâmetros sem Tipo

```
/*
 * Aula 14 - Exemplo 6
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>

using namespace std;

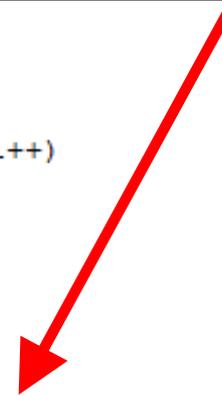
template <typename T, int elementos>
void printArray (T *a) {
    for (int i = 0; i < elementos; i++)
        cout << a [i] << endl;
}

int main(int argc, char *argv[])
{
    int array [] = {1, 2, 3, 4, 5};
    int elementos = 5;

    printArray <int, elementos> (array);

    return 0;
}
```

E agora? Posso utilizar uma variável inteira?



# Classe Template e Parâmetros sem Tipo

```
/*  
 * Aula 14 - Exemplo 6  
 * Arquivo Principal  
 * Autor: Miguel Campista  
 */  
#include <iostream>
```

	C:\Users\Miguel\Documents\UFRJ\...	In function `int main(int, char**):
21	C:\Users\Miguel\Documents\UFRJ\...	`elementos' cannot appear in a constant-expression
21	C:\Users\Miguel\Documents\UFRJ\...	template argument 2 is invalid
21	C:\Users\Miguel\Documents\UFRJ\...	no matching function for call to `printArray(int[5])'
	C:\Users\Miguel\Documents\UFRJ\...	[Build Error] exe: *** [aula14-ex6.o] Error 1

```
int main(int argc, char *argv[])  
{  
    int array [] = {1, 2, 3, 4, 5};  
    int elementos = 5;  
  
    printArray <int, elementos> (array);  
  
    return 0;  
}
```

# Classe Template e Parâmetros sem Tipo

```
/*
 * Aula 14 - Exemplo 6
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>

using namespace std;

template <typename T, int elementos>
void printArray (T *a) {
    for (int i = 0; i < elementos; i++)
        cout << a [i] << endl;
}

int main(int argc, char *argv[])
{
    int array [] = {1, 2, 3, 4, 5};
    const int elementos = 5;

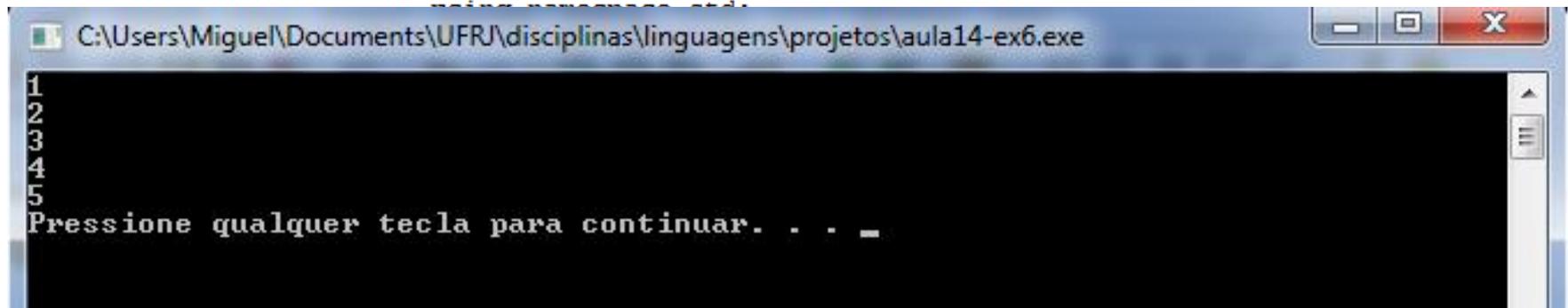
    printArray <int, elementos> (array);

    return 0;
}
```

E agora? Posso utilizar uma constante inteira?

# Classe Template e Parâmetros sem Tipo

```
/*  
 * Aula 14 - Exemplo 6  
 * Arquivo Principal  
 * Autor: Miguel Campista  
 */  
#include <iostream>
```



```
C:\Users\Miguel\Documents\UFRJ\disciplinas\linguagens\projetos\aula14-ex6.exe  
1  
2  
3  
4  
5  
Pressione qualquer tecla para continuar. . . _
```

```
int array [] = {1, 2, 3, 4, 5};  
const int elementos = 5;  
  
printArray <int, elementos> (array);  
  
return 0;  
}
```

# Classe Template e Parâmetros sem Tipo

- Classes templates
  - Parâmetro tipados
    - Tipo padrão
    - Ex.: `template< class T = string >`
      - Declara objeto do tipo `Stack<> pilha;`
    - Não podem ser usados em funções

# Classe Template e Parâmetros sem Tipo

- Sobrecarregando classes templates
  - Classe para tipo especializado definido explicitamente
    - Não usa nada da classe template original e pode até implementar suas próprias funções membro
      - Empregada quando um determinado tipo ou classe exige funções membro específicas

- Ex.:

```
template<>  
class Array< Martian > {  
    // corpo da definição de classe  
};
```

Objetos da classe *Martian* exigem um construtor padrão e funções membro próprias

# Classe Template e Parâmetros sem Tipo

```
#include <iostream>
#include <string>

using namespace std;

template <class T>
class SimpleStack {
public:
    SimpleStack () {
        cout << "Construtor na pilha simples!" << endl;
    }

    ~SimpleStack () {
        cout << "Destrutor na pilha simples!" << endl;
    }
};

template <>
class SimpleStack <string> {
public:
    SimpleStack () {
        cout << "Construtor na pilha string!" << endl;
    }

    ~SimpleStack () {
        cout << "Destrutor na pilha string!" << endl;
    }
};
```

# Classe Template e Parâmetros sem Tipo

```
#include <iostream>
#include <string>
#include "aula14-ex8-template.h"

using namespace std;

int main() {
    SimpleStack <int> s;
    SimpleStack <string> ss;

    return 0;
}
```

# Classe Template e Parâmetros sem Tipo

```
#include <iostream>
#include <string>
#include "aula14-ex8-template.h"

using namespace std;

int main() {
    SimpleStack <int> s;
    SimpleStack <string> ss;

    return 0;
}
```

```
miguel@pegasus-linux:~$ g++ -Wall aula14-ex8.cpp -o a
miguel@pegasus-linux:~$ ./a
Construtor na pilha simples!
Construtor na pilha string!
Destrutor na pilha string!
Destrutor na pilha simples!
```

# Templates e Herança

- Há muitas maneiras de relacionar templates e herança
  - Classe template derivada de classe template especializada
  - Classe template derivada de classe que não é template
  - Classe template especializada derivada de classe template especializada
  - Classe não template derivada de classe template especializada

# Template e friend

- “Amizade” entre classe template e...
  - Função global
  - Função membro de outra classe
    - Possivelmente uma classe template especializada
  - Classe inteira
    - Possivelmente uma classe template especializada

# Template e friend

- Funções friend
  - Na definição de `template< class T > class X`
    - `friend void f1 ();`
      - `f1 ()` friend de todas as classes template especializadas
    - `friend void f2 ( X< T > & );`
      - `f2 ( X< float > & )` friend de `X< float >` somente,  
`f2 ( X< double > & )` friend de `X< double >` somente,  
`f2 ( X< int > & )` friend de `X< int >` somente,  
...

# Template e friend

- Funções friend
  - Na definição de `template< class T > class X`
    - `friend void A::f4();`
      - Função membro `f4` da classe `A` friend de todas as classes template especializadas
    - `friend void C< T >::f5( X< T > & );`
      - Função membro `C<float>::f5( X< float> & )` friend da class `X<float>` somente

# Template e friend

- **Classes friend**
  - Na definição de `template< class T > class X`
    - **friend class Y;**
      - Toda função membro de `Y` é friend de toda classe `template` especializada
    - **friend class Z<T>;**
      - `class Z<float>` friend da classe `template` especializada `X<float>`, etc.

# Template e friend

```
#include <iostream>

using namespace std;

template <class T>
class F {
    public:
        void printArray (T &a) {
            for (int i = 0; i < a.size; i++) {
                cout << a.array [i] << endl;
            }
        }
};
```

# Template e friend

```
#include <iostream>
#include "friendCap14Ex7.h"

using namespace std;

template <class T>
class Aceitaf {
    friend class F < Aceitaf<T> >;

public:
    Aceitaf (int s) : size (s), array (new T [s]) {}
    ~Aceitaf () { delete [] array; }
    void insere (T elemento) {
        static int index = 0;
        array [index++] = elemento;
    }

private:
    T *array;
    int size;
};
```

# Template e friend

```
#include <iostream>
#include "aceitafCap14Ex7.h"

using namespace std;

int main() {
    Aceitaf <int> ac (5);
    F <Aceitaf <int> > f;

    ac.insere (1);
    ac.insere (2);
    ac.insere (3);
    ac.insere (4);
    ac.insere (5);

    f.printArray (ac);

    return 0;
}
```

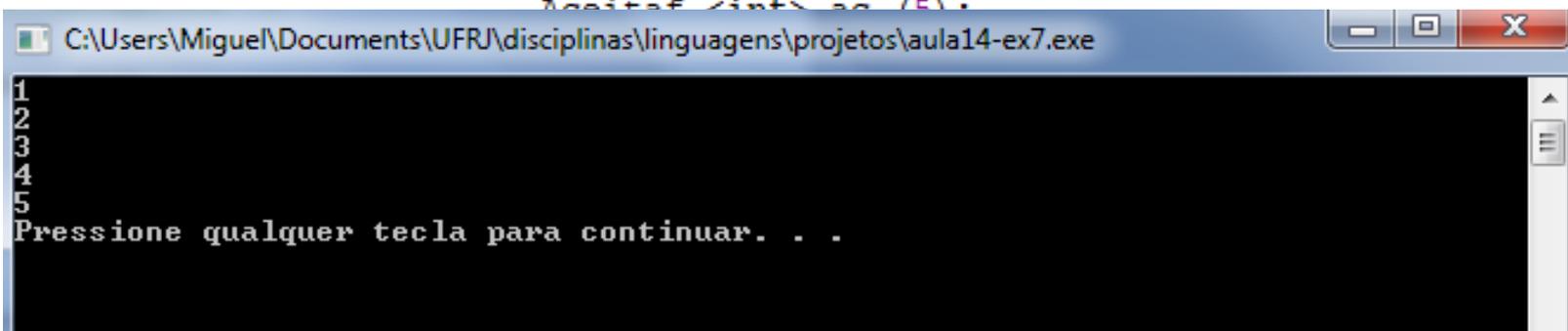
# Template e friend

```
#include <iostream>
#include "aceitafCap14Ex7.h"
```

```
using namespace std;
```

```
int main() {
```

```
    aceitaf<int> ac(5);
```



```
C:\Users\Miguel\Documents\UFRJ\disciplinas\linguagens\projetos\aula14-ex7.exe
1
2
3
4
5
Pressione qualquer tecla para continuar. . .
```

```
    f.printArray (ac);
```

```
    return 0;
```

```
}
```

# Templates e Membros static

- **Classe que não é template**
  - Membros de dados estáticos (`static`) são compartilhados entre todos os objetos
- **Classe template**
  - Cada classe especializada tem a sua própria cópia do membro de dados estático e de funções membro (`static`)
    - Todos os objetos da mesma classe especializada compartilham os dados estáticos
  - Variáveis estáticas (`static`) devem ser inicializadas em escopo de arquivo
    - Como das classes que não são templates

# Exemplo 1

- Escreva um programa que implemente a classe template List que retira elementos de uma fila na mesma ordem que foram inseridos (esquema first-in first-out FIFO)



# Exemplo 1

```
/*
 * Aula 14 - Exemplo 4
 * Arquivo listaCap14Ex4.h
 * Autor: Miguel Campista
 */
#ifndef TLIST1_H
#define TLIST1_H

using namespace std;

template <class T>
class List {
public:
    List (int = 10); // Construtor padrão

    // Destrutor
    ~List () {
        delete [] listPtr;
    }

    bool push (const T&); // insere um elemento na Lista
    bool pop (T&);       // retira um elemento da Lista

    // Verifica se a Lista está vazia
    bool isEmpty () const {
        return top == -1;
    }

    // Verifica se a Lista está cheia
    bool isFull () const {
        return top == size - 1;
    }
}
```

# Exemplo 1

```
        private:
            int size;
            int top;
            T *listPtr;
};

template <class T>
List <T>::List (int s) { // Construtor padrão
    size = s > 0 ? s : 10;
    top = -1;
    listPtr = new T [size];
}

// insere elemento na lista
// se bem sucedido, retorna verdadeiro; senão, retorna falso
template <class T>
bool List <T>::push (const T &pushValue) {
    if ( !isFull() ) {
        listPtr [++top] = pushValue; // coloca item na Pilha
        return true; // inserção bem sucedida
    }
    return false; // inserção mal sucedida
}
```

# Exemplo 1

```
// retira elemento da lista
// se bem sucedido, retorna verdadeiro; senão, retorna falso
template <class T>
bool List <T>::pop( T &popValue ) {
    if ( !isEmpty() ) {
        popValue = listPtr[0]; // remove item da lista
        for (int i = 0; i < top; i++)
            listPtr [i] = listPtr [i+1];
        --top;
        return true; // remoção bem sucedida
    }
    return false; // remoção mal sucedida
}

#endif
```

# Exemplo 1

```
/*
 * Aula 14 - Exemplo 4
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include "listaCap14Ex4.h"

int main() {
    List<double> doubleList (5);
    double doubleValue = 1.1;

    cout << "Pushing elements onto doubleList\n";

    while (doubleList.push (doubleValue)) {
        cout << doubleValue << ' ';
        doubleValue += 1.1;
    }

    cout << "\nList is full. Cannot push " << doubleValue
        << "\n\nPopping elements from doubleList\n";

    while (doubleList.pop (doubleValue))
        cout << doubleValue << ' ';

    cout << "\nList is empty. Cannot pop\n";

    List< int > intList;
    int intValue = 1;
    cout << "\nPushing elements onto intList\n";

    while (intList.push (intValue)) {
        cout << intValue << ' ';
        ++intValue;
    }
}
```

# Exemplo 1

```
cout << "\nList is full. Cannot push " << intValue
      << "\n\nPopping elements from intList\n";

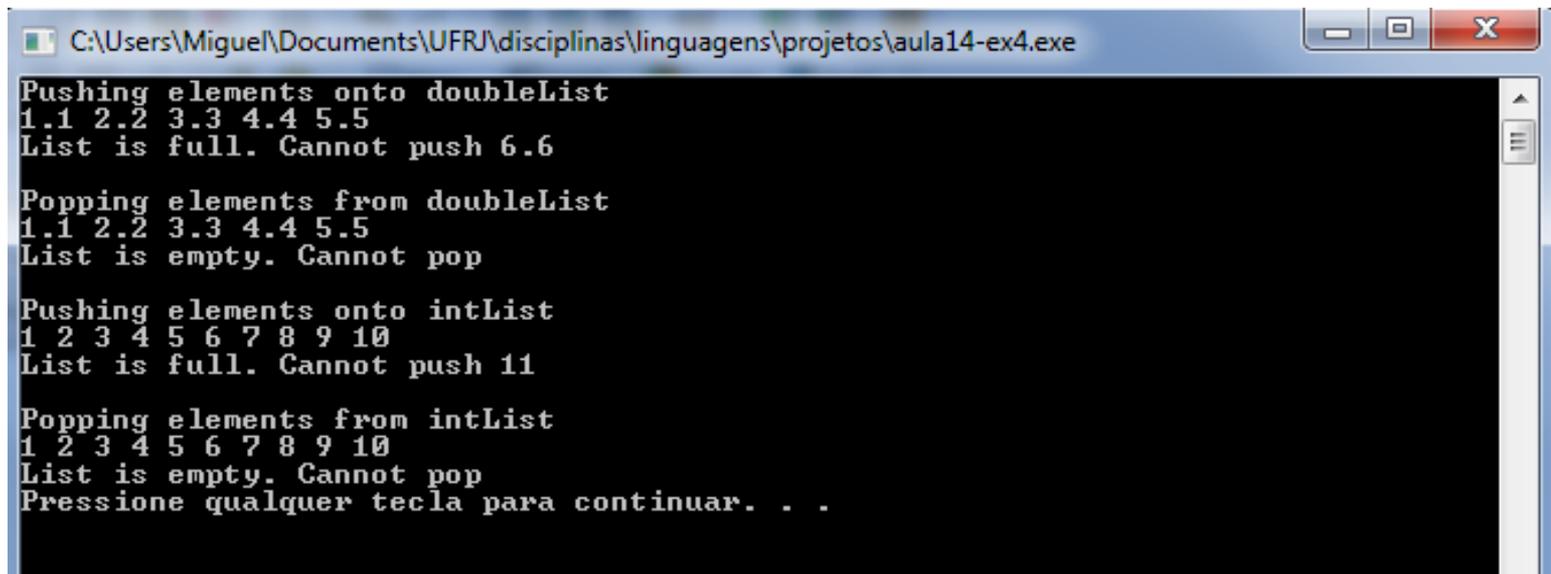
while (intList.pop (intValue))
    cout << intValue << ' ';

cout << "\nList is empty. Cannot pop\n";

return 0;
}
```

# Exemplo 1

```
cout << "\nList is full. Cannot push " << intValue  
      << "\n\nPopping elements from intList\n";
```



```
C:\Users\Miguel\Documents\UFRJ\disciplinas\linguagens\projetos\aula14-ex4.exe  
Pushing elements onto doubleList  
1.1 2.2 3.3 4.4 5.5  
List is full. Cannot push 6.6  
  
Popping elements from doubleList  
1.1 2.2 3.3 4.4 5.5  
List is empty. Cannot pop  
  
Pushing elements onto intList  
1 2 3 4 5 6 7 8 9 10  
List is full. Cannot push 11  
  
Popping elements from intList  
1 2 3 4 5 6 7 8 9 10  
List is empty. Cannot pop  
Pressione qualquer tecla para continuar. . .
```

# Exemplo 2

- Escreva um programa que implemente a classe template `SonList` que herda da classe `List` do Exemplo 1 e adiciona o método `testList`, semelhante ao método do segundo exemplo desta aula.



# Exemplo 2

```
/*
 * Aula 14 - Exemplo 5
 * Arquivo listaCap14Ex5.h
 * Autor: Miguel Campista
 */
#ifndef TLIST1_H
#define TLIST1_H

using namespace std;

template <class T>
class List {
public:
    List (int = 10); // Construtor padrão

    // Destrutor
    ~List () {
        delete [] listPtr;
    }

    bool push (const T&); // insere um elemento na Lista
    bool pop (T&);       // retira um elemento da Lista

    // Verifica se a Lista está vazia
    bool isEmpty () const {
        return top == -1;
    }

    // Verifica se a Lista está cheia
    bool isFull () const {
        return top == size - 1;
    }
};
```

# Exemplo 2

```
        private:
            int size;
            int top;
            T *listPtr;
};

template <class T>
List <T>::List (int s) { // Construtor padrão
    size = s > 0 ? s : 10;
    top = -1;
    listPtr = new T [size];
}

// insere elemento na lista
// se bem sucedido, retorna verdadeiro; senão, retorna falso
template <class T>
bool List <T>::push (const T &pushValue) {
    if ( !isFull() ) {
        listPtr [++top] = pushValue; // coloca item na Lista
        return true; // inserção bem sucedida
    }
    return false; // inserção mal sucedida
}
```

# Exemplo 2

```
// retira elemento da lista  
// se bem sucedido, retorna verdadeiro; senão, retorna falso  
template <class T>  
bool List <T>::pop( T &popValue ) {  
    if ( !isEmpty() ) {  
        popValue = listPtr[0]; // remove item da lista  
        for (int i = 0; i < top; i++)  
            listPtr [i] = listPtr [i+1];  
        --top;  
        return true; // remoção bem sucedida  
    }  
    return false; // remoção mal sucedida  
}
```

# Exemplo 2

```
/*
 * Aula 14 - Exemplo 5
 * Arquivo listaCap14Ex5.h
 * Autor: Miguel Campista
 */
#ifndef TLIST2_H
#define TLIST2_H

#include "listaCap14Ex5.h"

using namespace std;

template <class T>
class SonList : public List <T> {
public:
    SonList (int = 10);
    ~SonList () {}

    void testList (SonList<T> &, T, T, const char *);

};

// Construtor padrão
template <class T>
SonList <T>::SonList (int s) : List <T> (s){}
```

# Exemplo 2

```
// função template para manipular Stack< T >
template <class T>
void SonList <T>::testList (
    SonList <T> &theList,    // referência para List< T >
    T value,                // valor inicial para inserir
    T increment,           // incremento para valores subsequentes
    const char *listName ) // nome do objeto List< T >
{
    cout << "\nPushing elements onto " << listName << '\n';

    while ( theList.push( value ) ) {
        cout << value << ' ';
        value += increment;
    }

    cout << "\nList is full. Cannot push " << value
         << "\n\nPopping elements from " << listName << '\n';

    while ( theList.pop( value ) )
        cout << value << ' ';
    cout << "\nList is empty. Cannot pop\n";
}

#endif
```

```
/*
 * Aula 14 - Exemplo 5
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include "sonlistaCap14Ex5.h"

int main() {
    SonList <double> doubleList (5);
    double doubleValue = 1.1;

    cout << "Pushing elements onto doubleList\n";

    while (doubleList.push (doubleValue)) {
        cout << doubleValue << ' ';
        doubleValue += 1.1;
    }

    cout << "\nList is full. Cannot push " << doubleValue
        << "\n\nPopping elements from doubleList\n";

    while (doubleList.pop (doubleValue))
        cout << doubleValue << ' ';

    cout << "\nList is empty. Cannot pop\n";

    SonList< int > intList;
    int intValue = 1;
    cout << "\nPushing elements onto intList\n";

    while (intList.push (intValue)) {
        cout << intValue << ' ';
        ++intValue;
    }
}
```

# Exemplo 2

```
cout << "\nList is full. Cannot push " << intValue
      << "\n\nPopping elements from intList\n";

while (intList.pop (intValue))
    cout << intValue << ' ';

cout << "\nList is empty. Cannot pop\n";

SonList< double > newList;
newList.testList ( newList, 1.0, 1.5, "newList" );

return 0;
}
```

# Exemplo 2

```
cout << "\nList is full. Cannot push " << intValue
...

C:\Users\Miguel\Documents\UFRJ\disciplinas\linguagens\projetos\aula14-ex5.exe
Pushing elements onto doubleList
1.1 2.2 3.3 4.4 5.5
List is full. Cannot push 6.6

Popping elements from doubleList
1.1 2.2 3.3 4.4 5.5
List is empty. Cannot pop

Pushing elements onto intList
1 2 3 4 5 6 7 8 9 10
List is full. Cannot push 11

Popping elements from intList
1 2 3 4 5 6 7 8 9 10
List is empty. Cannot pop

Pushing elements onto newList
1 2.5 4 5.5 7 8.5 10 11.5 13 14.5
List is full. Cannot push 16

Popping elements from newList
1 2.5 4 5.5 7 8.5 10 11.5 13 14.5
List is empty. Cannot pop
Pressione qualquer tecla para continuar. . .
```

# Leitura Recomendada

- Capítulos 14 do livro
  - Deitel, "*C++ How to Program*", 5th edition, Editora Prentice Hall, 2005