TEMPLATES

- São usadas quando múltiplas "cópias" de código são necessárias para implementar uma mesma função com diferentes tipos de dados:
  - Pilha de inteiros, pilha de caracteres, pilha de números complexos, etc.

TEMPLATES

- Permitem especificar uma série de funções relacionadas , ou uma série de classes relacionadas, com um único segmento de código
  - Templates de funções
  - Templates de classes

Sobrecarga x Templates de funções

- Sobrecarga de funções
  - Operações similares
    - Diferentes tipos de dados
- Templates de funções
  - Operações idênticas
    - Diferentes tipos de dados
    - Compilador gera códigos-objeto separados para cada tipo de dados

Templates de funções

- Definição de templates de funções
  - Palavra chave template
    - Template <parâmetros de template>

      Ex.:

```
template< typename TipoElemento >
template< class T >
template< class BorderType, class FillType >
```

    - Especificam :
      - Tipos de argumentos para a função
      - Tipos de retorno da função
      - Variáveis dentro da função

Exemplo

```
template <typename Tgenerico>
Tgenerico Maior (Tgenerico x, Tgenerico y) {
      if(x>y)
            return x;
      else
            return y;
}
```

...

```
    int i1=2, i2=6;

    char c1='a', c2='h';

    Conta co1("Renato", 1000), co2("Luis", 5000);


    cout << "Inteiro maior: "<< Maior(i1, i2)<<endl;

    cout << "Caracter maior: "<< Maior(c1, c2)<<endl;

    cout << "Conta maior: "<< Maior(co1, co2)<<endl;
```

Templates de funções

```
1  // Fig. 11.1: fig11_01.cpp
2  // Using template functions.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  // function template printArray definition
9  template< class T >
10 void printArray( const T *array, const int count )
11 {
12   for ( int i = 0; i < count; i++ )
13     cout << array[ i ] << " ";
14
15   cout << endl;
16
17 } // end function printArray
18
19 int main()
20 {
21   const int aCount = 5;
22   const int bCount = 7;
23   const int cCount = 6;
24
```

Se T é um tipo definido pelo usuário, operador << precisa ser sobrecarregado para a classe T.

Templates de funções

```
25  int a[ aCount ] = { 1, 2, 3, 4, 5 };
26  double b[ bCount ] = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7 };
27  char c[ cCount ] = "HELLO";  // 6th position for null
28
29  cout << "Array a contains:" << endl;
30
31  // call integer function-template specialization
32  printArray( a, aCount );
33
34  cout << "Array b contains:" << endl;
35
36  // call double function-template specialization
37  printArray( b, bCount );
38
39  cout << "Array c contains:" << endl;
40
41  // call character function-template specialization
42  printArray( c, cCount );
43
44  return 0;
45
46 } // end main
```

Compilador substitui T por int:

```
void printArray( const int *array,
const int count )
{
   for ( int i = 0; i < count; i++ )
      cout << array[ i ] << " "
   cout << endl;
} // end function printArray
```

Templates de classes

- Pilha
  - LIFO (last-in-first-out)
- Class templates
  - Programação genérica
  - Descreve a noção de pilha genericamente
    - Versões instanciadas para tipo específico

```
1   // Fig. 11.2: tstack1.h
2   // Stack class template.
3   #ifndef TSTACK1_H
4   #define TSTACK1_H
5
6   template< class T >
7   class Stack {
8
9   public:
10    Stack( int = 10 );  // default constructor (stack size 10)
11
12    // destructor
13    ~Stack()
14    {
15       delete [] stackPtr;
16
17    } // end ~Stack destructor
18
19    bool push( const T& );  // push an element onto the stack
20    bool pop( T& );         // pop an element off the stack
21
```

O parâmeto de tipo `T` indica o tipo da classe `Stack` a ser criada.

Parâmetros de funções de tipo `T`.

```
22    // determine whether Stack is empty
23    bool isEmpty() const
24    {
25       return top == -1;
26
27    } // end function isEmpty
28
29    // determine whether Stack is full
30    bool isFull() const
31    {
32       return top == size - 1;
33
34    } // end function isFull
35
36 private:
37    int size;    // # of elements in the stack
38    int top;     // location of the top element
39    T *stackPtr;  // pointer to the stack
40
41 }; // end class Stack
42
```

Vetor de elementos do tipo `T`.

```
43  // constructor
44  template< class T >
45  Stack< T >::Stack( int s )
46  {
47     size = s > 0 ? s : 10;
48     top = -1;  // Stack initially empty
49     stackPtr = new T[ size ]; // allocate memory for elements
50
51  } // end Stack constructor
52
53  // push element onto stack;
54  // if successful, return true; otherwise, return false
55  template< class T >
56  bool Stack< T >::push( const T &pushValue )
57  {
58     if ( !isFull() ) {
59        stackPtr[ ++top ] = pushValue; // place item on Stack
60        return true;  // push successful
61
62     } // end if
63
64     return false;  // push unsuccessful
65
66  } // end function push
67
```

Construtor creia um vetor de elementos do tipo `T`.

Cabecalho dos métodos:

```
template< class T >
```

Operador `(::)` com o nome da class-template (`Stack< T >`)

Templates de classes

```
68  // pop element off stack;
69  // if successful, return true; otherwise, return false
70  template< class T >
71  bool Stack< T >::pop( T &popValue )
72  {
73     if ( !isEmpty() ) {
74        popValue = stackPtr[ top-- ]; // remove item from Stack
75        return true;  // pop successful
76
77     } // end if
78
79     return false;  // pop unsuccessful
80
81  } // end function pop
82
83  #endif
```

Usando um template de classe

- Pilha de inteiros (tamanho padrão:10)

```
Stack <int> intStack;
```

- Pilha de 'doubles' de tamanho 5

```
Stack <double> doubleStack(5);
```

Exercício

Utilizando o template da classe Stack, crie no programa principal pilhas de inteiros e doubles

Usando um template de classe

```
// Stack-class-template test program.
#include <iostream>

using std::cout;
using std::cin;
using std::endl;

#include "stack.h"  // Stack class template definition

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;

   } // end while

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

Inclusão do class-template

Instancia objeto da classe `Stack< double >`.

Invoca `push` da especialização `Stack< double >`.

Usando um template de classe

```
   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;

   } // end while

   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;
```

Exercício

Crie uma classe genérica que permita a criação de listas.

Podem ser criadas listas de inteiros, doubles, floats e caracteres.

As listas são criadas com tamanho fornecido para o construtor da classe, e devem estar disponíveis métodos para acessar e configurar qualquer elemento da lista.

Mostre como utilizar a classe para criar uma lista de inteiros e uma lista de "Tartarugas" no programa principal.