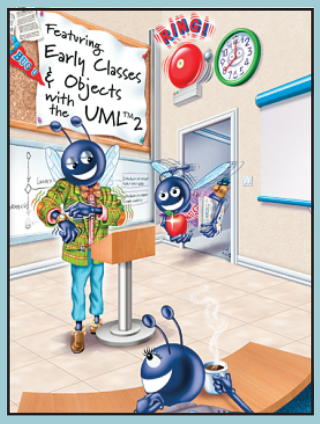
14



Templates

OBJECTIVES

In this chapter you will learn:

- To use function templates to conveniently create a group of related (overloaded) functions.
- To distinguish between function templates and function-template specializations.
- To use class templates to create a group of related types.
- To distinguish between class templates and class-template specializations.
- To overload function templates.
- To understand the relationships among templates, friends, inheritance and static members.

Assignment Checklist

Name: _____ Date: _____

Section: _____

Exercises	Assigned: Circle assignments	Date Due
Prelab Activities		
Matching	YES NO	
Fill in the Blank	10, 11, 12, 13, 14, 15	
Short Answer	16, 17	
Programming Output	18, 19	
Correct the Code	20, 21, 22	
Lab Exercises		
Exercise 1 — Overloading `printArray`	YES NO	
Exercise 2 — Equality Function Template	YES NO	
Follow-Up Questions and Activities	1, 2	
Debugging	YES NO	
Labs Provided by Instructor		
1.		
2.		
3.		
Postlab Activities		
Coding Exercises	1, 2	
Programming Challenges	1, 2	

Prelab Activities

Matching

Name: _____ **Date:** _____

Section: _____

After reading Chapter 14 of *C++ How to Program: Fifth Edition*, answer the given questions. These questions are intended to test and reinforce your understanding of key concepts and may be done either before the lab or during the lab.

For each term in the column on the left, write the corresponding letter for the description that best matches it from the column on the right.

Term	Description
___ 1. Nontype parameter	a) Class that is a type-specific version of a generic class.
___ 2. Class template	b) Class templates are called this because they require one or more type parameters.
___ 3. `template`	c) Parameter to a class template that can have a default argument and is treated as a constant.
___ 4. Type parameter	d) Parameter to a function template that can be any valid identifier and is replaced when the function is invoked.
___ 5. `class/typename`	e) Placed before every formal type parameter of a function template.
___ 6. Function template	f) Performs similar operations on different types of data, assuming the operations are identical for each type.
___ 7. `static` member function	g) Technique provided by class templates.
___ 8. parameterized type	h) All function-template definitions begin with this keyword.
___ 9. generic programming	i) Each class-template specialization gets its own copy.

Prelab Activities

Name:

Fill in the Blank

Name: _____ **Date:** _____

Section: _____

Fill in the blank for each of the following statements:

10. _____ provide the means for describing a class generically and for instantiating classes that are type-specific versions of this generic class.

11. Each class-template specialization gets a copy of the class template's _____ member functions.

12. Templates allow the programmer to specify a range of related _____ and _____.

13. Formal _____ names among function templates need not be unique.

14. Each class-template specialization has its own copy of each _____ data member of the class template; all objects of that class-template specialization share that one data member.

15. Class templates are called _____ as they require type parameters to specify how to customize a generic class template to form a class-template specialization.

Prelab Activities

Name:

Short Answer

Name: _____ **Date:** _____

Section: _____

In the space provided, answer each of the given questions. Your answers should be as concise as possible; aim for two or three sentences.

16. What are the advantages of using function templates instead of macros?

17. How does the compiler determine which function to call when a function is invoked? Your answer should include a discussion of how the function invocation is matched with the template function.

Prelab Activities

Name:

Programming Output

Name: _____ **Date:** _____

Section: _____

For each of the given program segments, read the code and write the output in the space provided below each program. [*Note:* Do not execute these programs on a computer.]

18. What is output by the following program?

```
1  #include <iostream>
2
3  using std::cout;
4  using std::endl;
5
6  // function template mystery definition
7  template< typename T >
8  void mystery( const T *a, const int c )
9  {
10    for ( int i = 0; i < c; i++ )
11       cout << a[ i ] << " ** ";
12
13    cout << endl;
14 } // end function mystery
15
16 int main()
17 {
18    const int size = 5;
19
20    int i[ size ] = { 22, 33, 44, 55, 66 };
21    char c[ size ] = { 'c', 'd', 'g', 'p', 'q' };
22
23    mystery( i, size );
24    cout << endl;
25    mystery( c, size - 2 );
26    cout << endl;
27    return 0;
28 } // end main
```

Your answer:

Prelab Activities

Name:

Programming Output

19. What is output by the following program?

```
1  #include <iostream>
2
3  using std::cout;
4  using std::endl;
5  using std::ios;
6
7  #include <new>
8
9  #include <iomanip>
10 using std::fix;
11 using std::setprecision;
12
13 // class template for class Array
14 template< typename T >
15 class Array
16 {
17 public:
18    Array( int = 5 );
19    ~Array() { delete [] arrayPtr; }
20    T arrayRef( int ) const;
21    int getSize() const;
22 private:
23    int size;
24    T *arrayPtr;
25 }; // end class Array
26
27 // constructor for class Array
28 template< typename T >
29 Array< T >::Array( int x )
30 {
31    size = x;
32    arrayPtr = new T[ size ];
33
34    for ( int i = 0; i < size; i++ )
35       arrayPtr[ i ] = 1.0 * i;
36 } // end class Array constructor
37
38 // function arrayRef definition
39 template< typename T >
40 T Array< T >::arrayRef( int num ) const
41 {
42    return arrayPtr[ num ];
43 } // end function arrayRef
44
45 // return size
46 template< typename T >
47 int Array< T >::getSize() const
48 {
49    return size;
50 } // end function getSize
51
52 // non-member function template to print an object of type Array
53 template< typename T >
54 void printArray( const Array< T > &a )
55 {
```

Prelab Activities

Name:

Programming Output

```
56       for ( int i = 0; i < a.getSize(); i++ )
57          cout << a.arrayRef( i ) << " ";
58
59       cout << endl << endl;
60    } // end function printArray
61
62    int main()
63    {
64       Array< int > intArray( 4 );
65       Array< double > doubleArray;
66
67       cout << setprecision( 2 ) << fixed;
68       printArray( intArray );
69       printArray( doubleArray );
70
71       return 0;
72    } // end main
```

Your answer:

Prelab Activities

Name:

Correct the Code

Name: _____ **Date:** _____

Section: _____

For each of the given program segments, determine if there is an error in the code. If there is an error, specify whether it is a logic, syntax or compilation error, circle the error in the program, and write the corrected code in the space provided after each problem. If the code does not contain an error, write "no error." [*Note:* It is possible that a program segment may contain multiple errors.]

20. The following code invokes the function `print`:

```
1  #include <iostream>
2
3  using std::cout;
4  using std::endl;
5
6  // template function print definition
7  template < class T >
8  void print( T left, T right )
9  {
10    cout << "Printing arguments: " << left
11         << "  **  " << right;
12 } // end function print
13
14 int main()
15 {
16    cout << endl;
17    print( 3, 5.8 );
18    cout << endl;
19
20    return 0;
21 } // end main
```

Your answer:

Prelab Activities

Name:

Correct the Code

21. The following is a class definition for a class template `Stack`:

```
1  #ifndef TSTACK_H
2  #define TSTACK_H
3
4  // template class Stack definition
5  template< class T >
6  class Stack
7  {
8  public:
9     Stack( int = 10 );
10
11    // destructor
12    ~Stack()
13    {
14       delete [] stackPtr;
15
16    } // end class Stack destructor
17
18    bool push( const T& );
19    bool pop( T& );
20 private:
21    int size;
22    int top;
23    T *stackPtr;
24
25    // function isEmpty definition
26    bool isEmpty() const
27    {
28       return top == -1;
29
30    } // end function isEmpty
31
32    // function isFull definition
33    bool isFull() const
34    {
35       return top == size - 1;
36
37    } // end function isFull
38 }; // end class Stack
39
40 // constructor
41 Stack< T >::Stack( int s )
42 {
43    size = s > 0 ? s : 10;
44    top = -1;
45    stackPtr = new T[ size ];
46 } // end class Stack constructor
47
48 // function push definition
49 bool Stack< T >::push( const T &pushValue )
50 {
51    if ( !isFull() )
52    {
53       stackPtr[ ++top ] = pushValue;
54       return true;
55    } // end if
```

Prelab Activities

Name:

Correct the Code

```
56
57       return false;
58    } // end function push
59
60    // function pop definition
61    bool Stack< T >::pop( T &popValue )
62    {
63       if ( !isEmpty() )
64       {
65          popValue = stackPtr[ top-- ];
66          return true;
67       } // end if
68
69       return false;
70    } // end function pop
71
72    #endif // TSTACK_H
```

Your answer:

Prelab Activities

Name:

Correct the Code

22. The following code invokes function `print`:

```
1  #include <iostream>
2
3  using std::cout;
4  using std::cin;
5  using std::endl;
6
7  // class MyClass definition
8  class MyClass
9  {
10 public:
11    MyClass();
12
13    void set( int );
14    int get() const;
15 private:
16    int data;
17 }; // end class MyClass
18
19 // constructor
20 MyClass::MyClass()
21 {
22    set( 0 );
23 } // end class MyClass constructor
24
25 // function setData definition
26 void MyClass::setData( int num )
27 {
28    data = num >= 0 ? num : -1;
29 } // end function setData
30
31 // return data
32 int MyClass::getData() const
33 {
34    return getData;
35 } // end function getData
36
37 // template function print
38 template < class T >
39 void print( T left, T right )
40 {
41    cout << "Printing arguments: " << left
42         << "  **  " << right;
43 } // end function print
44
45 int main()
46 {
47    MyClass m1;
48    MyClass m2;
49
50    m1.setData( 2 );
51    m2.setData( 7 );
52
53    cout << endl;
54    print( m1, m2 );
55    cout << endl;
56
```

Prelab Activities

Name:

Correct the Code

```
57      return 0;
58   } // end main
```

Your answer:

Lab Exercises

Lab Exercise 1 — Overloading `printArray`

Name: _____ **Date:** _____

Section: _____

This problem is intended to be solved in a closed-lab session with a teaching assistant or instructor present. The problem is divided into five parts:

1. Lab Objectives
2. Description of the Problem
3. Sample Output
4. Program Template (Fig. L 14.1)
5. Problem-Solving Tips

The program template represents a complete working C++ program, with one or more key lines of code replaced with comments. Read the problem description and examine the sample output; then study the template code. Using the problem-solving tips as a guide, replace the `/* */` comments with C++ code. Compile and execute the program. Compare your output with the sample output provided. The source code for the template is available at `www.deitel.com` and `www.prenhall.com./deitel`.

Lab Objectives

This lab was designed to reinforce programming concepts from Chapter 14 of *C++ How To Program: Fifth Edition*. In this lab, you will practice

- Overloading a function template for printing an array.
- Using function templates to create function-template specializations.

Problem Description

Overload function template `printArray` of Fig. 14.1 so that it takes two additional integer arguments, namely `int lowSubscript` and `int highSubscript`. A call to this function will print only the designated portion of the array. Validate `lowSubscript` and `highSubscript`; if either is out of range or if `highSubscript` is less than or equal to `lowSubscript`, the overloaded `printArray` function should return `0`; otherwise, `printArray` should return the number of elements printed. Then modify `main` to demonstrate both versions of `printArray` on arrays `a`, `b` and `c` (lines 23–25 of Fig. 14.1). Be sure to test all capabilities of both versions of `printArray`.

Lab Exercises

Name:

Lab Exercise 1 — Overloading `printArray`

Sample Output

```
Using original printArray function
1 2 3 4 5
Array a contains:
1 2 3 4 5
5 elements were output
Array a from positions 1 to 3 is:
2 3 4
3 elements were output
Array a output with invalid subscripts:
0 elements were output


Using original printArray function
1.1 2.2 3.3 4.4 5.5 6.6 7.7
Array b contains:
1.1 2.2 3.3 4.4 5.5 6.6 7.7
7 elements were output
Array b from positions 1 to 3 is:
2.2 3.3 4.4
3 elements were output
Array b output with invalid subscripts:
0 elements were output


Using original printArray function
H E L L O
Array c contains:
H E L L O
5 elements were output
Array c from positions 1 to 3 is:
E L L
3 elements were output
Array c output with invalid subscripts:
0 elements were output
```

Template

```
1  // Lab 1: TemplateOverload.cpp
2  // Using template functions
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  // function template printArray definition
8  // original function from Fig. 14.1
9  template< typename T >
10 void printArray( const T *array, int count )
11 {
12    // display array
13    for ( int i = 0; i < count; i++ )
14       cout << array[ i ] << " ";
15
16    cout << endl;
17 } // end function printArray
```

Fig. L 14.1 | `TemplateOverload.h`. (Part 1 of 3.)

Lab Exercises

Name:

Lab Exercise 1 — Overloading `printArray`

```
18
19  // overloaded function template printArray
20  // takes upper and lower subscripts to print
21  /* Write a header for an overloaded printArray function
22     that takes two additional int arguments, lowSubscrip
23     and highSubscript; remember to include the template
24     header */
25  {
26     // check if subscript is negative or out of range
27     if ( /* Write conditions to test if the size if negative,
28             or if the range is invalid */ )
29        return 0;
30
31     int count = 0;
32
33     // display array
34     for ( /* Write code to iterate from lowSubscript up to
35              and including highSubscript */ )
36     {
37        count++;
38        cout << array[ i ] << ' ';
39     } // end for
40
41     cout << '\n';
42     return count; // number or elements output
43  } // end overloaded function printArray
44
45  int main()
46  {
47     const int ACOUNT = 5; // size of array a
48     const int BCOUNT = 7; // size of array b
49     const int CCOUNT = 6; // size of array c
50
51     // declare and initialize arrays
52     int a[ ACOUNT ] = { 1, 2, 3, 4, 5 };
53     double b[ BCOUNT ] = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7 };
54     char c[ CCOUNT ] = "HELLO"; // 6th position for null
55     int elements;
56
57     // display array a using original printArray function
58     cout << "\nUsing original printArray function\n";
59     printArray( a, ACOUNT );
60
61     // display array a using new printArray function
62     cout << "Array a contains:\n";
63     elements = /* Write a call to printArray that specifies
64                   0 to ACOUNT - 1 as the range */
65     cout << elements << " elements were output\n";
66
67     // display elements 1-3 of array a
68     cout << "Array a from positions 1 to 3 is:\n";
69     elements = /* Write a call to printArray that specifies
70                   1 to 3 as the range */
71     cout << elements << " elements were output\n";
72
```

Fig. L 14.1 | `TemplateOverload.h`. (Part 2 of 3.)

Lab Exercises

Name:

Lab Exercise 1 — Overloading `printArray`

```
73      // try to print an invalid element
74      cout << "Array a output with invalid subscripts:\n";
75      elements = /* Write a call to printArray that specifies
76                    -1 to 10 as the range */
77      cout << elements << " elements were output\n\n";
78
79      // display array b using original printArray function
80      cout << "\nUsing original printArray function\n";
81      printArray( b, BCOUNT );
82
83      // display array b using new printArray function
84      cout << "Array b contains:\n";
85      elements = /* Write a call to printArray that specifies
86                    0 to BCOUNT - 1 as the range */
87      cout << elements << " elements were output\n";
88
89      // display elements 1-3 of array b
90      cout << "Array b from positions 1 to 3 is:\n";
91      elements = /* Write a call to printArray that specifies
92                    1 to 3 as the range */
93      cout << elements << " elements were output\n";
94
95      // try to print an invalid element
96      cout << "Array b output with invalid subscripts:\n";
97      elements = /* Write a call to printArray that specifies
98                    -1 to 10 as the range */
99      cout << elements << " elements were output\n\n";
100
101     // display array c using original printArray function
102     cout << "\nUsing original printArray function\n";
103     printArray( c, CCOUNT );
104
105     // display array c using new printArray function
106     cout << "Array c contains:\n";
107     elements = /* Write a call to printArray that specifies
108                   0 to CCOUNT - 2 as the range */
109     cout << elements << " elements were output\n";
110
111     // display elements 1-3 of array c
112     cout << "Array c from positions 1 to 3 is:\n";
113     elements = /* Write a call to printArray that specifies
114                   1 to 3 as the range */
115     cout << elements << " elements were output\n";
116
117     // try to display an invalid element
118     cout << "Array c output with invalid subscripts:\n";
119     elements = /* Write a call to printArray that specifies
120                   -1 to 10 as the range */
121     cout << elements << " elements were output" << endl;
122     return 0;
123 } // end main
```

Fig. L 14.1 | `TemplateOverload.h`. (Part 3 of 3.)

Lab Exercises

Name:

Lab Exercise 1 — Overloading `printArray`

Problem-Solving Tips

1. To overload the `printArray` function template, declare another function template, also named `printArray`, that takes two additional `int` parameters, `lowSubscript` and `highSubscript`.
2. When iterating over the range from `lowSubscript` to `highSubscript`, make sure to include both values within the range, to avoid an off-by-one error.

Lab Exercises

Name:

Lab Exercise 2 — Equality Function Template

Name: _____ **Date:** _____

Section: _____

This problem is intended to be solved in a closed-lab session with a teaching assistant or instructor present. The problem is divided into six parts:

1. Lab Objectives
2. Description of the Problem
3. Sample Output
4. Program Template (Fig. L 14.2)
5. Problem-Solving Tip
6. Follow-Up Questions and Activities

The program template represents a complete working C++ program, with one or more key lines of code replaced with comments. Read the problem description and examine the sample output; then study the template code. Using the problem-solving tip as a guide, replace the `/* */` comments with C++ code. Compile and execute the program. Compare your output with the sample output provided. Then answer the follow-up questions. The source code for the template is available at `www.deitel.com` and `www.prenhall.com./deitel`.

Lab Objectives

This lab was designed to reinforce programming concepts from Chapter 14 of *C++ How To Program: Fifth Edition*. In this lab, you will practice

- Overloading a function template for testing equality.
- Using function templates to create function-template specializations.

The follow-up questions and activities also will give you practice:

- Overloading operators to enable function templates to function with user-defined types.

Problem Description

Write a simple function template for predicate function `isEqualTo` that compares its two arguments of the same type with the equality operator (==) and returns `true` if they are equal and `false` if they are not equal. Use this function template in a program that calls `isEqualTo` only with a variety of built-in types.

Sample Output

```
Enter two integer values: 2 5
2 and 5 are not equal

Enter two character values: a a
a and a are equal

Enter two double values: 2.5 7.5
2.5 and 7.5 are not equal
```

Lab Exercises

Name:

Lab Exercise 2 — Equality Function Template

Template

```
1  // Lab 2: isEqualTo.cpp
2  // A function template for equality
3  #include <iostream>
4  using std::cin;
5  using std::cout;
6  using std::ostream;
7
8  // function template isEqualTo
9  /* Write a template header with one formal type parameter */
10 /* Write a header for function template isEqualTo */
11 {
12    return arg1 == arg2;
13 } // end function isEqualTo
14
15 int main()
16 {
17    int a; // integers used for
18    int b; // testing equality
19
20    // test if two ints input by user are equal
21    cout << "Enter two integer values: ";
22    cin >> a >> b;
23    cout << a << " and " << b << " are "
24       << ( /* Write a call to isEqualTo */ ? "equal" : "not equal" ) << '\n';
25
26    char c; // chars used for
27    char d; // testing equality
28
29    // test if two chars input by user are equal
30    cout << "\nEnter two character values: ";
31    cin >> c >> d;
32    cout << c << " and " << d << " are "
33       << ( /* Write a call to isEqualTo */ ? "equal" : "not equal" ) << '\n';
34
35    double e; // double values used for
36    double f; // testing equality
37
38    // test if two doubles input by user are equal
39    cout << "\nEnter two double values: ";
40    cin >> e >> f;
41    cout << e << " and " << f << " are "
42       << ( /* Write a call to isEqualTo */ ? "equal" : "not equal" ) << '\n';
43    return 0;
44 } // end main
```

Fig. L 14.2 | `isEqualTo.cpp`.

Problem-Solving Tip

1. Function template `isEqualTo` compares two values for equality, so both parameters should be of the same type and use the same formal type parameter.

Follow-Up Questions and Activities

1. Now write a version of the program that calls `isEqualTo` with a user-defined class type, but does not overload the equality operator for that class type. What happens when you attempt to compile and run this program?

Lab Exercises

Name:

Lab Exercise 2 — Equality Function Template

2. Now overload the equality operator for the user-defined type. Now what happens when you attempt to compile and run this program?

Lab Exercises

Name:

Debugging

Name: _____ **Date:** _____

Section: _____

The program (Fig. L 14.3–Fig. L 14.4) in this section does not run properly. Fix all the compilation errors so that the program will compile successfully. Once the program compiles, compare the output with the sample output, and eliminate any logic errors that may exist. The sample output demonstrates what the program's output should be once the program's code has been corrected.

Sample Output

```
Arithmetic performed on object a:
The result of the operation is: 8
The result of the operation is: 2
The result of the operation is: 15
The result of the operation is: 1

Arithmetic performed on object b:
The result of the operation is: 12.5
The result of the operation is: 2.1
The result of the operation is: 37.96
The result of the operation is: 1.40385
```

Broken Code

```
1  // Debugging: Arithmetic.h
2
3  #ifndef ARITHMETIC_H
4  #define ARITHMETIC_H
5
6  // template class Arithmetic
7  template< T >
8  class Arithmetic
9  {
10 public:
11    Arithmetic( T, T );
12    T addition() const;
13    T subtraction() const;
14    T multiplication() const;
15    T division() const;
16 private:
17    int value1;
18    int value2;
19 }; // end class Arithmetic
20
21 // constructor
22 Arithmetic::Arithmetic( T v1, T v2 )
23 {
24    value1 = v1;
```

Fig. L 14.3 | `Arithmetic.h`. (Part 1 of 2.)

Lab Exercises

Name:

Debugging

```
25       value2 = v2;
26    } // end class Arithmetic constructor
27
28    // template function addition
29    template< typename T >
30    T Arithmetic::addition() const
31    {
32       return value1 + value2;
33    } // end function addition
34
35    // template function subtraction
36    template< typename T >
37    T Arithmetic< T >::subtraction() const
38    {
39       return value1 - value2;
40    } // end function subtraction
41
42    // template function multiplication
43    template< typename T >
44    T Arithmetic< T >::multiplication() const
45    {
46       return value1 * value2;
47    } // end function multiplication
48
49    // template function division
50    template< typename X >
51    X Arithmetic< X >::division() const
52    {
53       return val1 / val2;
54    } // end function division
55
56    #endif //ARITHMETIC_H
```

Fig. L 14.3 | `Arithmetic.h`. (Part 2 of 2.)

```
1   // Debugging: debugging.cpp
2
3   #include <iostream>
4
5   using std::cout;
6   using std::endl;
7
8   #include "Arithmetic.h"
9
10  // template function printResult definition
11  < typename T >
12  void printResult( T number )
13  {
14     cout << "The result of the operation is: " << number << endl;
15  } // end function printResult
16
17  int main()
18  {
19     Arithmetic a( 5, 3 );
20     Arithmetic< int > b( 7.3, 5.2 );
```

Fig. L 14.4 | `debugging.cpp`. (Part 1 of 2.)

Lab Exercises

Name:

Debugging

```
21
22     cout << "Arithmetic performed on object a:\n";
23     printResult( a< int >.addition() );
24     printResult( a< int >.subtraction() );
25     printResult( a< int >.multiplication() );
26     printResult( a< int >.division() );
27
28     cout << "\nArithmetic performed on object b:\n";
29     printResult( b.addition() );
30     printResult( b.subtraction() );
31     printResult( b.multiplication() );
32     printResult( b.division() );
33
34     return 0;
35
36  } // end main
```

Fig. L 14.4 | `debugging.cpp`. (Part 2 of 2.)

Postlab Activities

Coding Exercises

Name: _____ **Date:** _____

Section: _____

These coding exercises reinforce the lessons learned in the lab and provide additional programming experience outside the classroom and laboratory environment. They serve as a review after you have completed the *Prelab Activities* and *Lab Exercises* successfully.

For each of the following problems, write a program or a program segment that performs the specified action:

1. Write a function template that determines the largest of its four arguments. Assume that all four arguments are of the same type.

2. Convert the `linearSearch` program of Fig 7.19 in *C++ How to Program: Fifth Edition* into a template function.

Postlab Activities

Name:

Programming Challenges

Name: _____ **Date:** _____

Section: _____

The *Programming Challenges* are more involved than the *Coding Exercises* and may require a significant amount of time to complete. Write a C++ program for each of the problems in this section. The answers to these problems are available at `www.deitel.com` and `www.prenhall.com/deitel`. Pseudocode, hints and/or sample outputs are provided to aid you in your programming.

1. Write a function template `selectionSort` based on the sort program of Fig. 8.15. Write a driver program that inputs, sorts and outputs an `int` array and a `float` array.

Hints:

- Examine the selection sort code closely to determine which `int` declarations are for index values (these must remain as `int`s) and which refer to array elements (these should be changed to formal type parameter `T`).
- The `swap` function will also have to be templatized to complete function template `selectionSort`.
- Sample output:

```
int data items in original order
    10     9     8     7     6     5     4     3     2     1
int data items in ascending order
     1     2     3     4     5     6     7     8     9    10

double point data items in original order
  10.1   9.9   8.8   7.7   6.6   5.5   4.4   3.3   2.2   1.1
double point data items in ascending order
   1.1   2.2   3.3   4.4   5.5   6.6   7.7   8.8   9.9  10.1
```

2. Overload function template `printArray` of Fig. 14.1 with a nontemplate version that specifically prints an array of `string` objects in neat tabular, column format.

Hints:

- The nontemplate version of `printArray` for `string` arrays will be its own function, independent of the `printArray` function template. Because the nontemplate version explicitly specifies a `string` array as its parameter, it will take precedence over the template version when `printArray` is called with a `string` array argument.

Postlab Activities

Name:

Programming Challenges

- Sample output:

```
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

Array strings contains:
       one       two     three      four
      five       six     seven     eight
```