13

Templates

REVIEW QUESTIONS

1. A function has only one level of generalization—one function with different sets of data.

3. A function template can only change the arguments and return types. It cannot change the function logic. Therefore, we cannot write one template to sort or search an array.

5. Yes, if we use a static cast of one to another.

7. No because templates can not be used in all situations. Two common situations are when there is a pointer argument or when the relational operators are not defined for the object.

9. Yes. Macros are useful for small problems only. Templates can solve large and small problems.

11. Yes. Just like with standard types, there is no limit to how many generic types can be used in a template. The search template is a good example. We need a generic type for the array and one for the key.

13. No. The size of an array can only be an integer; it cannot be generic. For example, if it were generic, then the user could create a version in which the array size was a float.

15. If the class itself is generic (template class), we need to use a template function to implement a generic object type. Even when he class is not generic, template functions are more efficient.

EXERCISES

17. The parameter list in template prefix needs to be enclosed in pointed brackets.

19. Each generic type requires the keyword *class*.

21. Assuming that Z is not a declared type somewhere earlier in the program, it must be a generic type. To be a generic type, it needs to be introduced in the template prefix.

23. There are two errors. First, the keyword *template* must be all lowercase. In the function declaration, it is coded with an uppercase *T*. Second, the generic type must be specified in the function header as shown below.

```
template<class T>
Sample<T> :: Sample () {}
```

PROBLEMS

25.

```
/* Swap two numbers
     Pre  Given two numbers
     Post Exchange their values
*/
template<class TYPE>
void swapTwo (TYPE& x, TYPE& y)
{
   TYPE hold;
   hold = x;
   x    = y;
   y    = hold;
   return;
}; // swapTwo
```

27.

```
/* Return sum of an array
      Pre  ary is array
           size is number of elements in array
     Post Return sum of array
*/
template<class TYPE>
TYPE sumAry (TYPE ary, int size)
{
   TYPE sum = 0;
   for (int i = 0; i < size; i++)
        sum += ary[i];
   return sum;
}; // sumAry
```

29.

```
/* Return largest element of an array
      Pre  ary is array
           size is number of elements in array
     Post Return largest element
*/
template<class TYPE>
TYPE largest (TYPE ary[], int size)
{
   TYPE large = ary[0];
   for (int i = 1; i < size; i++)
        (ary[i] > large) ? large = ary[i] : 0;
   return large;
}; // largest
```

31.

```
/* Test driver to test smallest generic function with
   type double
      Written by:
      Date:
*/
#include <iostream>
```

```
using namespace std;

int main ()
{
   cout << "=== Start Test Driver ===\n\n";

   double a[4] = {15.15, -111.1, 83.38, 3.14159};
   cout << "Smallest: " << smallest (a, 4) << endl;

   cout << "\n=== End Test Driver ===\n";
   return 0;
}  // main
```