Project 6: Sort Templates

Making InsertionSort generic

Revision dated 01/02/18

Educational Objectives: After completing this assignment the student should have the following knowledge, ability, and skills:

- Distinguish between a function template and a concrete function: definition, implementation, and file structure conventions
- Demonstrate correct notation and syntax for defining and implementing a function template
- State the rules compilers use to instantiate a function template
- Test a function template for correct syntax
- Test a function template for correct behavior
- Test a function template for genericity

Operational Objectives: Implement and test the function templates InsertionSort and Display. Build applications and test programs from previous projects.

Deliverables: Files:

```
tsort.h           # contains function templates
cstringdiff.h     # from project 1 - will be re-tested
cstringdiff.cpp   # from project 1 - will be re-tested
product.h         # from project 2, with input operator and less-than operator added
product.cpp       # implements product.h
log.txt           # as usual: work log and testing diary
```

(We supply a makefile for this project.)

Assessment Rubric

```
builds:                                    [0..5]:   x
test1: [intsort]                           [0..5]:   x
test2: [stringsort]                        [0..5]:   x
test3: [charsort]                          [0..5]:   x
test4: [productsort]                      [0..10]:  xx
test5: [fcstringdiff]                      [0..5]:   x
test6: [fcstringdiff]                      [0..5]:   x
log.txt [answered questions]              [0..10]:  xx
log.txt [including test diary]           [-20..0]: ( x)
project specs                            [-20..0]: ( x)
code quality                             [-20..0]: ( x)
dated submission deduction            [2 pts per]: ( x)
                                                    ---
total                                     [0..50]:  xx

Notes: 1. input files may vary over time
       2. answers to questions should be near top of log

Code quality includes:
  - conformance to assignment requirements and specifications
  - conformance to coding standards [see course organizer]
  - engineering and design, including appropriateness of name choices
  - readability
```

Background: See lecture notes Chapter 12. Templates.

Procedural Requirements

1. Copy all files from `LIB/proj6/`. You should see at least these:

   ```
   charsort.cpp      # sorts files of characters
   productsort.cpp   # sorts files of Product tokens
   intsort.cpp       # sorts files of integers
   stringsort.cpp    # sorts files of strings
   makefile          # builds all required components of project
   deliverables.sh   # submission configuration file
   ```

2. Begin your log file named `log.txt`. (See Assignments for details.)
3. Read and understand the supplied `makefile`. Pay careful attention to the targets and dependency relationships.
4. Read and understand the supplied client sort programs `charsort.cpp`, `intsort.cpp`, `stringsort.cpp`, and `productsort.cpp`. Pay careful attention to the similarities among these files.
5. Create the file `tsort.h` containing the function templates `Display` and `InsertionSort`, as detailed in the requirements section.
6. Test your templates by building and running `charsort.x`, `intsort.x`, and `stringsort.x` using the supplied makefile. (Command `make intsort.x` builds `intsort.x`.) Note: "make" also attempts to build productsort.x, which won't work until you have completed steps 7 and 8. So you need to make the 3 specific targets at this point.
7. Copy your files `proj1/cstringdiff.h` and `proj1/cstringdiff.cpp` to `proj6/`. If these need revision based on feedback from proj1, do that now.
8. Copy your files `proj2/product.h` and `proj2/product.cpp` into your `proj6` directory. Modify these files by adding `operator>>` and `operator<`, as detailed below (and, of course, updating the file header doc).
9. Test and take notes ...
10. Answer the questions at the end of this document. Place the questions, each followed by your answer, near the top of your log.txt (before the chronology and test diary).
11. Turn in all deliverables using the submit script system. (Read here for reminders how the submit system works.)

    ***Warning:** Submit scripts do not work on the `program` and `linprog` servers. Use `shell.cs.fsu.edu` or `quake.cs.fsu.edu` to submit projects. If you do not receive two confirmations, the second with the contents of your project, there has been a malfunction.*

Code Requirements and Specifications

1. In file `tsort.h` define and implement function templates with these prototypes:

   ```
   template < typename T >
   void Display (const T* beg, const T* end, char ofc = '\0');

   template < typename T >
   void InsertionSort (T* beg, T* end);
   ```

2. Both function templates use pointers to an unknown type `T` to define a range of values that is assumed, as usual, to be "half open", that is, include the beginning element and exclude the ending element. We use the notation `[beg,end)` to denote the range.

3. `Display` writes the data in the range `[beg,end)` to standard output. If the output formatting character `ofc` is `'\0'` then the output data is not separated by anything. If `ofc` is any other character, then `ofc` should be output preceeding each item in the range.

4. `InsertionSort` transforms the data in the range `[beg,end)` into sorted order using the InsertionSort algorithm. This is the same algorithm discussed in the Project 1 document and implemented in your file `proj1/cstringsort.cpp`.

5. The software will only work on types for which there is an input operator overload (to fill up an array using `std::cin >> ...`) and a less-than operator overload (so that the sort algorithm can compare two elements).

   Overload these operators for class Product using the following prototypes:

   ```
   std::istream& operator>> ( std::istream& is , Product& p );
   bool          operator<  ( const Product& p1 , const Product& p2 );
   ```

   The behavior of `>>` should be:

   a. Read a string into a locally declared buffer of size 121. Be sure to protect the string read against buffer overflow with `std::setw`. Note that this restricts product names to 120 characters.
   b. Read a hexadecimal number into a locally declared `uint32_t` "code" variable.
   c. Read a decimal fraction into a locally declared "prive" variable.
   d. Call the `p.Set...(...)` methods to put the read data into the object `p`.
   e. Return `is` by reference as usual.
   f. Hint: the data read can be done with a chain of six calls to operator>>: three data reads each preceded by a manipulator call (std::setw, std::hex, std::dec).

   The behavior of `<` should be based on the value `d = DictionaryDiff( p1.GetName() , p2.GetName() )` according to these cases:

   a. If d < 0 return true.
   b. If d > 0 return false.
   c. If d == 0 return (p1.GetAge() < p2.GetAge()).

   In words: first consider the name (independent of case), then use age to break ties.

   Note that neither of these operators is a member operator of class Product, nor should either be a friend of the class.

6. Be sure that you have tested your code for syntax errors. All warnings should be eliminated.

7. Be sure that you have tested your code for logic errors with the supplied clients `charsort.cpp`, `intsort.cpp` and `stringsort.cpp`.

8. You should take some time to think about the results. For example, if you have a file of unsigned integers, that same file can be processed by `charsort.x`, `intsort.x` and `stringsort.x`, and the outputs differ. (If there are an even number of numbers in the file, it can be processed by `productsort.x`!) Make sure you can explain why and how this occurs.

9. Be sure that you have tested your code for genericity by compiling and running `productsort.cpp`.

10. Be sure your code conforms to the C++ Code Standards (available also through the Course Organizer).

11. Be sure you understand why the parameters for all operators and functions discussed above are declared as they are.

Discussion Questions

Answer the following questions. Submit the questions, each followed by your answer, near the beginning of your log, before the chronology and test diary.

1. State what changes you made to transform the InsertionSort code from the specific code in project 1 to a function template. And for each change, explain why it was made.
2. A file of integers can be sorted as type int or type std::string. Explain why the results sometimes differ.
3. Explain why the sort template works for type std::string. (Hint: what operators are available for std::string?)
4. Explain why the sort template does not work (without some code enhancements) for C-strings.
5. Considering the way we were able to apply generic InsertionSort to type Product, describe what would be required to make the sort template actually work for C-strings. (You don't need details, just a few sentences that draw on the experience with Product.)

Hints

- As we experienced in Project 1, the various sorts operate on files using redirect:

  ```
  intsort.x < datafile.in                 # sorted data should appear on screen
  intsort.x < datafile.in > datafile.out  # sorted data should be in file datafile.out
  ```

  It's up to you to creatively create your test data files.

- Suppose you have a file `data.9` with this content

  ```
  250 16 28 562 10 15 28 400 122
  ```

  Then the screen should appear as follows for the four sort executables:

  ```
  intsort.x < data.9
   A as entered:  250 16 28 562 10 15 28 400 122
   A after sort:  10 15 16 28 28 122 250 400 562

  stringsort.x < data.9
   A as entered:  250 16 28 562 10 15 28 400 122
   A after sort:  10 122 15 16 250 28 28 400 562

  charsort.x < data.9
  ```

```
 A as entered: 250162856210152840012 2
 A after sort: 0000111122222245556688

productsort.x < data.9
 A as entered:
250     00000016        28.00
562     00000010        15.00
28      00000400        122.00
 A after sort:
250     00000016        28.00
28      00000400        122.00
562     00000010        15.00
```

Be sure you understand why these results occur and why they are correct.

- Suppose you have a file `data.product` with this content

```
widget_B  ABCDEF01  131.00
gadget_1  ABCDEF02  109.99
widget_A  ABCDEF03  129.95
gadget_2  12345604  109.00
widget_A  00123405  109.95
widget_b  12344506  109.95
gadget_1  23456707   99.99
```

This is clearly intended to be data for productsort. But the file can be processed as a file of strings (and even as a file of char). The results are as follows:

```
productsort.x < data.product
 A as entered:
widget_B        ABCDEF01        131.00
gadget_1        ABCDEF02        109.99
widget_A        ABCDEF03        129.95
gadget_2        12345604        109.00
widget_A        00123405        109.95
widget_b        12344506        109.95
gadget_1        23456707        99.99
 A after sort:
gadget_1        23456707        99.99
gadget_1        ABCDEF02        109.99
gadget_2        12345604        109.00
widget_A        00123405        109.95
widget_A        ABCDEF03        129.95
widget_b        12344506        109.95
widget_B        ABCDEF01        131.00

stringsort.x < data.product
 A as entered:  widget_B ABCDEF01 131.00 gadget_1 ABCDEF02 109.99 widget_A ABCDEF03 129.95 gadget_2 12345604 109.00 widget_A 001
 A after sort:  00123405 109.00 109.95 109.95 109.99 12344506 12345604 129.95 131.00 23456707 99.99 ABCDEF01 ABCDEF02 ABCDEF03 g

charsort.x < data.product
 A as entered:
 widget_BABCDEF01131.00gadget_1ABCDEF02109.99widget_AABCDEF03129.95gadget_212345604109.00widget_A00123405109.95widget_b123445061
 A after sort:
 .......00000000000000000111111111111122222223333334444445555555666779999999999999999AAAAABBBBCCCDDDEEEFFF_______aaabddddddddeeeeee
```

Be sure you understand why these results occur and why they are technically correct. In partcular, note the role that variable type plays.