Project 2: The Product Class

Classes with Dynamically Allocated Members

Revision dated 01/02/18

Educational Objectives. After successfully completing this assignment, the student should be able to accomplish the following:

- Design a class based on non-language-specific specifications
- Implement a class of your own design
- Implement constructors, copy constructor, destructor, and assignment operator for a class that has resource allocation requirements
- Implement Set and Get methods for class data
- Correctly separate class definition and implementation using files
- Create, edit, build and run multi-file projects using the Linux/Emacs/Make environment announced in the course organizer.
- Test a class using specs and an existing test platform

Operational Objectives: Define and implement the class `Product` and deliver the code in two files `product.h` and `product.cpp` along with a makefile for the supplied test harness.

Deliverables: `product.h`, `product.cpp`, `makefile`, `log.txt`

Assessment Rubric

```
student build:                      [0..4]:   x
assess build:
    product.o                       [0..2]:   x
    producttest1.o                  [0..2]:   x
    producttest2.o                  [0..2]:   x
    producttest1.x                  [0..2]:   x
    producttest2.x                  [0..2]:   x
test:
    producttest1.x                 [0..10]:  xx
    producttest2.x                 [0..10]:  xx
code:
    constructor 0                   [0..1]:   x
    init list                       [0..1]:   x
    constructor 2                   [0..1]:   x
    init list                       [0..1]:   x
    copy constructor                [0..1]:   x
    init list                       [0..1]:   x
    destructor                      [0..1]:   x
    assignment operator             [0..1]:   x
engineering etc:
   requirements                    [-20..4]:   x   # note negative points awarded during assessment
   coding standard                 [-20..4]:   x   # note negative points awarded during assessment
dated submission deduction    [2 pts per]: ( x)  # note negative points awarded during assessment
                                           --
total                                [0..50]:  xx
```

Background

See lecture notes Chapter 4. Classes Part 1, Chapter 5. Pointers, Chapter 6. Classes Part 2, and Chapter 7: C-Strings.

See also the COP3014 Notes on C-Strings.

Procedural Requirements:

1. Begin as usual by creating your assignment directory and copying the distribution files for the assignment:

   ```
   cp ~cop3330p/LIB/proj2/* ~/cop3330/proj2/
   cp ~cop3330p/LIB/area51/product*.x ~/cop3330/proj2/
   ```

   Then a long listing of your assignment directory should look like this:

   ```
   -rw------- 1 xxxxxxxx CS-Class   505 Sep 14 11:45 deliverables.sh
   -rw------- 1 xxxxxxxx CS-Class 16469 Sep 14 11:46 producttest1_i.x
   -rw------- 1 xxxxxxxx CS-Class 13355 Sep 14 11:46 producttest2_i.x
   -rw------- 1 xxxxxxxx CS-Class 12216 Sep 14 11:46 producttest2ShallowCopy_i.x
   -rw------- 1 xxxxxxxx CS-Class  2030 Sep 14 11:45 producttest1.cpp
   -rw------- 1 xxxxxxxx CS-Class  1306 Sep 14 11:45 producttest2.cpp
   ```

   The area51 executables are for demonstration purposes. You can erase these and get them back by copying again. After invoking "`clean .`" to declutter the directory a long listing should be:

```
-rw------- 1 xxxxxxxx CS-Class   505 Sep 14 11:45 deliverables.sh
-rw------- 1 xxxxxxxx CS-Class  2030 Sep 14 11:45 producttest1.cpp
-rw------- 1 xxxxxxxx CS-Class  1306 Sep 14 11:45 producttest2.cpp
```

Now continue to the next step:

2. Begin your log file named `log.txt`. (See Assignments for details.)
3. Create a makefile that builds executables `producttest1.x` and `producttest2.x`. Look at the #include statements in producttest1.cpp and producttest2.cpp to deduce what the intermediate targets and dependencies should be.
4. Design the class Product, placing the definition in file `product.h`
5. Implement the class Product, placing the class implementation in file `product.cpp`. You can test the code for syntax errors with the command "`make product.o`" or the command "`co3330 product`".
6. Thoroughly test class Product, starting out with the supplied test harnesses in file `proj2/test?.cpp` using your makefile to build the executables. (Note - you could also use the command line compile scripts "co3330" to create object files and then and create an executable at the command line.)
7. Turn in `product.h`, `product.cpp`, and `makefile` using the `submit.sh` submit script.

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

Technical Requirements and Specifications

1. The class should implement the following diagram:

Class Name:	Product

Services :	`void SetName ( const char* ) // sets the name field`
	`void SetBarCode ( uint32_t ) // sets the bar code field`
	`void SetCost ( float ) // sets the cost field`
	`const char* GetName () const // returns a const pointer to the name field`
	`uint32_t GetBarCode () const // returns the bar code by value`
	`float GetCost () const // returns cost by value`

Properties :	`Constructable: objects can be declared as ordinary variables`
	`Assignable: objects can be assigned one to another`
	`Passable: objects can be passed by value to and returned as values from functions`

Private variables:	`char * name_ // the product name`
	`uint32_t code_ // the product bar code`
	`float cost_ // the product cost`

2. The class should be a proper type, to include default constructor, 3-argument constructor (that initializes the three data fields), copy constructor, assignment operator, and destructor. Note that the default constructor should set the name to "#", the bar code to 0x00000000, and the cost to 0.0. The `uint32_t` type is defined in the `cstdint` library.
3. Be sure to use initialization lists for all of the constructors, including the copy constructor.
4. The output operator `operator<<` should be overloaded for the type `Product`. Display the three fields with TAB character between them. Do not make this operator a class friend. (Don't output any newlines.) (See Hint below.)
5. Class Product should pass testing with the supplied `proj2/test?.cpp` with no compile or runtime errors and no compiler warnings when the warning flags `-Wall` and `-Wextra` are set. The test compiler is `clang++ -std=c++11` on linprog. This compiler and library is as close to compliance with c++11 as we have available.
6. Building and running the supplied `proj2/test?.cpp` should result in output identical to the supplied executable `area51/producttest?.x` [`?` = `1` or `2`] .

Hints

- The "resource allocation" aspects of this assignment are where most mistakes are made - not just by students doing this assignment, but also throughout the professional world: failure to correctly and safely manage C strings has been at the root of many security leaks and system intrusions over the years.
- The distributed "productTest2ShallowCopy.x" shows one of the things that can go wacky if the programming isn't up to muster. What you see on the screen is:

```
lacher@linprog2.cs.fsu.edu:~/3330/proj2>producttest2d.x
 Products after declaration:
  p1 = product_1        FEDCBA98        100.00
  p2 = product_2        89ABCDEF        200.00
 Products after assignment p2 = p1
  p1 = product_1        FEDCBA98        100.00
  p2 = product_1        FEDCBA98        100.00
```

```
*** glibc detected *** producttest2d.x: double free or corruption (fasttop): 0x0000000004b5e010 ***
======= Backtrace: =========
...
```

Note that the product name of p1 has changed (as you would expect after assignment p2 = p1), but the program "crashes" with a message "double free or corruption". What has happened is that the assignment operator made a "shallow copy" of the product name string. As the program terminates, the Product destructor is called for both p2 and p1, which results in a call to delete[] name_. Because of the shallow copy, delete[] is called twice on the same allocation, which generates the error.

- The output operator overload can use the following implementation:

```
std::ostream& operator << ( std::ostream& os , const Product& p)
{
  std::ios_base::fmtflags flags = os.flags();
  os.setf(std::ios::fixed | std::ios::showpoint); // prep for $ output
  os.precision(2);
  os << p.GetName() << '\t'
     << std::hex << std::uppercase << std::setfill('0') << std::setw(8) // prep for hex output
     << p.GetBarCode()
     << std::dec << std::setfill(' ') << '\t'          // return dec output
     << p.GetCost();
  os.setf(flags); // return flags to previous states
  return os;
}
```

It would be a good idea to understand this code as much as possible now, and then return to it after we cover C++ I/O.