$Id: cmps109-2016q1-exam1.mm,v 1.48 2016-01-28 14:50:26-08 - - $

page 1	page 2	page 3	Total / 32

Please print clearly :

Name :
Login : @ucsc.edu

Code only in C++11. No books ; No calculator ; No computer ; No email ; No internet ; No notes ; No phone. Neatness counts ! Points will be deducted for messy or unreadable answers. Do your scratch work elsewhere and enter only your final answer into the spaces provided.

1. Write the prototypes for class `foo` which will be implicitly generated unless otherwise specified. Point allocation is given in the table at left. **[2✔]**

6 correct :	**2**	✔
5 correct :	**1.5**	✔
4 correct :	**1**	✔
3 correct :	**0.5**	✔
else :	**0**	✔

2. Write a function that will check to see if one range is lexicographically less than another range. Each range is delimited by a pair of iterators. Use only `operator<` to compare elements in the ranges. Do not assume the iterators are direct access. For example, the range `{1,4,6,9}` is less than `{1,5,8}` and `{1,2,3}` is less than `{1,2,3,4}`. In other words, the first element less than the other indicates the range is less. If all elements of the shorter range are equal to the prefix of the longer range, then the shorter range is less than the longer range. **[3✔]**

```
template <typename itor>
bool lexic_less (itor begin1, itor end1, itor begin2, itor end2) {
```

3. Define the template function `find_if` whose first two arguments are iterators bounding a range, and whose third argument is a predicate. An iterator is returned which points at the first element found that satisfies the predicate. Example : the following call will return an iterator to the first `6` in the range : **[2✔]**

```
auto i = find_if (v.begin(), v.end(), [](int i){ return i == 6; });
```

4. Code the template function `merge`, which merges two ranges into a single output container. The first two arguments indicate one range and the next two indicate the other range. The last argument is a container with a `push_back` function. The input ranges are sorted into ascending order and `operator<` is available to compare elements of these ranges. **[3✔]**

```
template <typename itor, typename container>
void merge (itor begin1, itor end1, itor begin2, itor end2,
            container out) {
```

5. Assume class `complex` is defined as shown here. Write a single constructor inline in the class that: acts as a default constructor, setting both fields to 0.0 ; is useable as an implicit conversion operator that converts a `double` into a `complex`, setting the imaginary field to 0.0 ; and accepts two arguments, both `double`s. **[1✔]**

```
class complex {
   double real;
   double imag;
   public:
```

6. Consider the following abstract base class used as a base for expression tree evaluation. Code all functions inline inside of the class declarations. Then define derived classes `number` and `adder` which override the abstract functions.

```
class expr {
   public:
      virtual double eval() const = 0;
      virtual void print (ostream&) const = 0;
};
```

   (a) Define the class `number` which has a private field holding a `double`. Override the base class functions, and add a constructor whose argument is a `double` which has a default value of 0. **[2✔]**

   (b) Define the class `adder` whose private fields are `shared_ptr`s to `expr`s called `left` and `right`. `Eval` returns the sum of the values of its children. `Print` prints out the tree itself by printing an open parenthesis, followed by printing the left subtree, then a comma, then the right subtree, then a closing parenthesis. The constructor takes two arguments which are used to initialize the left and right pointers. **[4✔]**

   (c) Define a non-member `operator<<` which dispatches the print function based on the right argument. **[1✔]**

7. Define the template `operator<<` which takes a constant vector of any type by reference. It prints out the contents of the vector with one space separating each element from the next. No space is printed before the first or after the last element of the vector. **[2✔]**

Multiple choice. To the ***left*** of each question, write the letter that indicates your answer. Write ***Z*** if you don't want to risk a wrong answer. Wrong answers are worth negative points. **[12✓]**

number of correct answers		× 1 =	= a
number of wrong answers		× ½ =	= b
number of missing answers		× 0 =	0
column total $c = \max(a - b, 0)$	12		= c

1. What classes can see the protected parts of class `foo`?
   (A) classes derived from class `foo`.
   (B) classes which declare class `foo` as a friend.
   (C) other classes in the same package as class `foo`.
   (D) the base classes of class `foo`.

2. Which declares the move constructor for class foo?
   (A) `foo(const foo&&)`
   (B) `foo(const foo**)`
   (C) `foo(foo&&)`
   (D) `foo(foo**)`

3. Which class uses reference counting as a method of memory management?
   (A) `auto_ptr`
   (B) `reference_ptr`
   (C) `shared_ptr`
   (D) `unique_ptr`

4. What is the space overhead (extra pointers) for classes `vector<T>` and `list<T>` when there are $O(n)$ items in the data structure?
   (A) `vector<T>` is $O(1)$ and `list<T>` is $O(1)$
   (B) `vector<T>` is $O(n)$ and `list<T>` is $O(1)$
   (C) `vector<T>` is $O(1)$ and `list<T>` is $O(n)$
   (D) `vector<T>` is $O(n)$ and `list<T>` is $O(n)$

5. Which container supports `push_back` and `push_front` and allows direct access to any arbitrary element?
   (A) `array`
   (B) `deque`
   (C) `list`
   (D) `vector`

6. Which container can be used to find a value associated with a specific key using only $O(1)$ time?
   (A) `map`
   (B) `set`
   (C) `unordered_map`
   (D) `unordered_set`

7. Given `String s;` and `String t;` which is impossible?
   (A) `s != t and &s != &t`
   (B) `s != t and &s == &t`
   (C) `s == t and &s != &t`
   (D) `s == t and &s == &t`

8. Given the following declarations:
   ```
   class a { int x; };
   struct b { int y; };
   ```
   (A) `x` is private and `y` is private
   (B) `x` is private and `y` is public
   (C) `x` is public and `y` is private
   (D) `x` is public and `y` is public

9. How can implicit generation of the copy constructor for class `foo` be prevented?
   (A) `foo (const foo&) = default;`
   (B) `foo (const foo&) = delete;`
   (C) `foo (foo&&) = default;`
   (D) `foo (foo&&) = delete;`

10. Assuming `m` is a `map`, after the following statement is executed, how can one find the value associated with the given key?
    ```
    auto& i = m.find (key);
    ```
    (A) `i->first`
    (B) `i->second`
    (C) `i.first`
    (D) `i.second`

11. What does the following statement do?
    ```
    cout << 3.0/0.0 << endl;
    ```
    (A) Throws a division by zero error.
    (B) Prints `0`.
    (C) Prints `inf`.
    (D) Prints `nan`.

12. Is half of two plus two equal to two or three?
    (A) two
    (B) three
    (C) yes
    (D) no