`$Id: cmps109-2015q1-exam2.mm,v 1.47 2015-02-20 17:15:19-08 - - $`

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 ! Do your scratch work elsewhere and enter only your final answer into the spaces provided.

1. Rewrite the following statement by using the two-semicolon version of the `for`-loop, i.e., by using iterators explicitly : `for (auto i&: c) f(i);` **[1✔]**

2. Define the template `operator<<` whose second argument is a constant reference to a `pair`. It has two template arguments which give the type of the `first` and `second` fields. It prints out a left brace (`{`), then the first of the pair, then a comma (`,`), then the second of the pair, then a right brace (`}`). Example : a `pair<int,int>` would look like `{3,4}`. **[2✔]**

```
template <typename First, typename Second>
```

3. A hash map full of math functions and evaluation.
   (a) Write two `using` statements : Define `mathfn` as a pointer to a function taking a `double` argument and returning a `double` result. Define `fnmap` as an `unordered_map` whose keys are `string`s and whose values are `mathfn`s. **[1✔]**

   (b) Define the function `evalfn`, which returns a `double` result. Its arguments are a `fnmap`, a `string` representing a function name, and a `double` value. The first two arguments are passed by constant reference. It looks up the function by name in the map. If found, it applies the function to the double argument and returns the result. If not found, it returns `numeric_limits<double>::quiet_NaN()`. **[2✔]**

4. Define the function `lenless` so that the following statement :

```
    sort (args.begin(), args.end(), lenless);
```

   will sort `vector<string> args` in such a way that shorter strings come before longer strings (use `size()`). Strings of the same length are sorted lexicographically. **[2✔]**

5. Define the function `find` which takes a pair of iterators and a key, and performs a linear search for the first item in the iterator range that is equal to the key. **[2✔]**

```
template <typename Itor, typename Key>
Itor find (Itor begin, Itor end, Key key) {
```

6. Given a pair of forward iterators specifying one range and another pair specifying a second range, `push_back` each pair onto a container. For example, if one input has `{1,2}` and the other has `{'a','b'}`, then the pairs to be pushed would be `{{1,'a'},{2,'b'}}`. Stop pushing pairs when one or the other ranges is exhausted. After pushing pairs, if both ranges are not exhausted, throw a `length_error`. The template function `make_pair` can create a pair object. **[3✔]**

```
template <typename Itor1, typename Itor2, typename Container>
void zip (Itor1 b1, Itor1 e1, Itor2 b2, Itor2 e2, Container& out) {
```

7. Define the function `monotonic` which returns true if its elements are monotonically increasing according to the binary functional parameter. Its arguments are a pair of iterators and a binary function. For example,
   `monotonic (v.begin(), v.end(), less<int>())`
   will return true if the range is increasing. It would return true for a monotonically decreasing function if passed `greater<int>` instead. Monotonically increasing means that each element is larger than the one before it. An empty sequence or a sequence of one element is always monotonically increasing. **[3✔]**

```
template <typename Itor, typename Function>
bool monotonic (Itor begin, Itor end, Function fn) {
```

8. Define the class `ivec::iterator` as outlined here :
   (a) Code all members of `iterator` inline, and code only those functions needed by the following statement:
   `for (auto i = vec.begin(); i != vec.end(); ++i) cout << *i << endl;`
   and also the constructor neede by `begin` and `end`. **[3✔]**
   (b) Also show the code for the functions `ivec::begin()` and `ivec::end()`. **[1✔]**

```
struct ivec {
   size_t size;
   int *data;
   struct iterator{
      int *curr;
      // Code ivec::iterator ctor and members here.







   };
   // Code ivec::begin and ivec::end here.
```

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 is the signature of the implicitly generated move assignment operator as it would appear inside class `foo` ?
   (A) `foo operator= (const foo&&);`
   (B) `foo operator= (foo&&);`
   (C) `foo& operator= (const foo&&);`
   (D) `foo& operator= (foo&&);`

2. What kind of a cast would be used to make a `size_t` from a pointer ?
   (A) `const_cast`
   (B) `dynamic_cast`
   (C) `reinterpret_cast`
   (D) `static_cast`

3. Which class allows $O(1)$ insertions at both ends of the sequence of elements and also allows $O(1)$ access to any arbitrary element given its index ?
   (A) `std::deque`
   (B) `std::list`
   (C) `std::stack`
   (D) `std::vector`

4. If `vs` is a `vector<string>`, what expression can be used to print out its first element ?
   (A) `cout<<**vs.begin()`
   (B) `cout<<*vs.begin()`
   (C) `cout<<vs.*begin()`
   (D) `cout<<vs.begin()`

5. Given the following declarations, which assignments are neither errors nor will cause slicing ?
```
class foo {};
class bar: public foo {};
foo *f; bar *b;
```
   (A) `*b = *f;`
   (B) `*f = *b;`
   (C) `b = f;`
   (D) `f = b;`

6. If `i` and `j` are iterators and `n` is an `int`, which of the following is possible for a direct access iterator but not for a forward iterator ?
   (A) `*i`
   (B) `++i`
   (C) `i!=j`
   (D) `i[n]`

7. Which of the following are C++ operators that can be declared with any number of arguments ?
   (A) `operator()`
   (B) `operator<>`
   (C) `operator[]`
   (D) `operator{}`

8. Which of the following kinds of data members will make the implicitly generated `operator=` inappropriate ?
   (A) `pointer`
   (B) `primitive`
   (C) `reference`
   (D) `string`

9. The constructors for an abstract class should usually be classified as :
   (A) `private`
   (B) `protected`
   (C) `public`
   (D) `friend`

10. Unless otherwise specified, members of a class are [x] and members of a struct are [y].
   (A) [x] = private, [y] = private.
   (B) [x] = private, [y] = public.
   (C) [x] = public, [y] = private.
   (D) [x] = public, [y] = public.

11. In a `Makefile`, what variable should fill in the blank ?
```
%.o : %.cpp
        ${COMPILECPP} -c __
```
   (A) `$<`
   (B) `$?`
   (C) `$@`
   (D) `$_`

12. What is the correct syntax to define an abstract virtual function ?
   (A) `virtual void show() = 0;`
   (B) `virtual void show() = delete;`
   (C) `virtual void show() = abstract;`
   (D) `virtual void show() = override;`