`$Id: cmps109-2019q2-midterm.mm,v 1.72 2019-05-31 11:10:27-07 - - $`

page 1	page 2	page 3	page 4	Total / 42

PLEASE PRINT CLEARLY :

Name :
CruzID : `@ucsc.edu`

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. Assuming `bigint& bigint::operator+= (const bigint&);`
   Show the complete code for the ***non-member*** `operator+`, which adds two `bigint`s and returns a `bigint`. **[2✔]**

2. Assuming that `bigint::bigint(long)` is non-expicit, and assuming `operator+=` above, write the code for the two versions of `operator++`, as their would appear outside of the class definition as non-members. (Show the code, not just the prototypes.)

   (a) Prefix `operator++` (i.e., `++x`). **[2✔]**

   (b) Postfix `operator++` (i.e., `x++`). **[2✔]**

3. Given the outline of `bigint` shown here, implement `bigint::operator<` as it appears outside of the class definition. Use `ubigint::operator<` for the necessary comparison, but no other `ubigint` operator or function. **[2✔]**

```
class bigint {
   private:
      bool is_negative;
      ubigint uvalue;
   public:
      bool operator< (const bigint& that) const;
};
```

4. Finish `contains`, which takes a pair of iterators indicating a range, and an item, and returns true if and only if the item is in the range. The template parameters are the iterator type, type item type, and a function object that checks for equality between items. **[2✔]**

```
template <typename itor_t, typename item_t, class equal=equal_to<item_t>>
bool contains (itor_t begin, itor_t end, const item_t& item) {
```

5. Define the function **find_if** whose template arguments are an iterator type and a predicate type and whose function arguments are a pair if iterators indicating a range and a boolean function or function object. Return an iterator pointing at the first element in the range for which the predicate returns true. **[2✔]**

```
template <typename iterator, class predicate>
iterator find_if (iterator begin, iterator end, predicate pred) {
```

6. Define classes **shape**, and classes **circle** and **square**, which inherit from **shape.** At the bottom of the page is a sample program using these classes, followed by its output.†

   (a) Class **shape** has abstract virtual constant functions **area** and **border** which do not take arguments and which return **double**. It has a virtual constant function **where** which has no arguments and returns its **location**. It has a private field of type **location** and a ctor which accepts a location as an argument and which also works as a default ctor by using a default location argument of **{0,0}**. **[4✔]**

   ```
   using location = pair<double,double>;
   class shape {
   ```

   (b) Class **circle** inherits from **shape** and has a ctor which also behaves as a default ctor by providing default arguments. Its first ctor argument is a constant reference to a **location** and its second argument is the value of its private **radius** field. (For a circle, border = circumference.) Use **M_PI** from the library for the value of π. Override **area** (πr^2) and **border** ($2\pi r$). **[2✔]**

   (c) Class **square** inherits from **shape** and is similar to **circle**, except that it has a private field called **edge** which specifies the length of one edge of the square. Its constructor has the same specification as for **circle**. It overrides **area** of the square and **border** (the sum of the lengths of the edges). **[2✔]**

† Program :

```
using shapemap = map<string,shared_ptr<shape>>;
int main() {
   shapemap m { {"one", make_shared<circle> (location(1,4), 6)},
                {"two", make_shared<square> (location(8,3), 5)},
                {"zz1", make_shared<circle> ()} };
   for (auto& i: m) {
      cout << i.first << ": (" << i.second->where().first << ","
           << i.second->where().second << "): a= " << i.second->area()
           << ", b= " << i.second->border() << endl;
   }
}
```

Output :

```
one: (1,4): a= 113.097, b= 37.6991
two: (8,3): a= 25, b= 20
zz1: (0,0): a= 0, b= 0
```

7. Assume some class has defined the member function **operator<**. Define the inline non-member template function **operator>** with the expected semantics. **[1✔]**

8. Define **operator<<** so that it prints a **pair** by printing a left parenthesis, then the first element, then a comma, then the second element, then a right parentheses. **[2✔]**

```
template <typename T1, typename T2>
ostream& operator<< (ostream& out, const pair<T1,T2>& pair_) {
```

9. Write a complete program (everything that needs to be put in **echo.cpp**) that will duplicate the Unix command **echo**(1). Note that there are ***no spaces*** after the last word is printed. **[2✔]**

```
-bash$ g++ echo.cpp -o echo
-bash$ ./echo

-bash$ ./echo foo bar baz
foo bar baz
-bash$ ./echo Hello, World!
Hello, World!
```

10. Finish the **Makefile** so that it will compile a program consisting of the two modules listed and produce an executable binary. Do not delete the object files. Fill in the blanks. Use separate steps to compile the sources into object files and to link the object files into the executable binary. Omit **g++** options that do not directly affect compilation and linking. For the variable **OBJECTS**, use proper substitution syntax — do not just repeat the sources by changing the names. Fill in the blanks. **[2✔]**

```
SOURCES = foo.cpp bar.cpp
OBJECTS = ________________________________
EXECBIN = foo
all : ${EXECBIN}
${EXECBIN} : __________________
        ____________________________________
_____ : _________
        ________________
```

11. Write a template function **lessrange** which returns true of one range is less than another range. It has one template parameter: an iterator. It has four iterator arguments: begin and end for the first range; and begin and end for the second range. It scans both ranges in parallel and returns true as soon a it finds a pair which is **operator<** than the other, and false otherwise. If one range is a subrange of the other, the shorter one is less than the other. Use only **operator<** to perform comparisons and assume that it exists for the elements in the range.

```
vector<int> v1 {1, 3, 5, 9, 11, 13, 15, 22};
vector<int> v2 {3, 4, 7, 10, 16};
vector<int> v3 {3, 4, 7, 10, 16, 19, 20};
```

For example: **v1<v2** is true, **v1<v1** is false, **v2<v3** is true. **[3✔]**

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 should be placed in the blank to initialize the vector to the command line arguments, excluding the name of the program ?
   `vector<string> args ________;`
   (A) `(&argv[0], &argv[argc - 1])`
   (B) `(&argv[0], &argv[argc])`
   (C) `(&argv[1], &argv[argc - 1])`
   (D) `(&argv[1], &argv[argc])`

2. What is the type of the constant `"abc"` ?
   (A) `char*`
   (B) `const char*`
   (C) `const char[3]`
   (D) `const char[4]`

3. Which is a copy constructor ?
   (A) `foo (const foo&&);`
   (B) `foo (const foo&);`
   (C) `foo (foo&&);`
   (D) `foo (foo&);`

4. What uses reference counting to manage memory for `foo` objects ?
   (A) `foo*`
   (B) `forward_list<foo>`
   (C) `shared_ptr<foo>`
   (D) `unique_ptr<foo>`

5. Which declares a post-increment operator ?
   (A) `T T::operator++();`
   (B) `T T::operator++(int);`
   (C) `T& T::operator++();`
   (D) `T& T::operator++(int);`

6. What might be one of the lines produced by the command `g++ -MM` ?
   (A) `foo.cpp: foo.o foo.h`
   (B) `foo.h foo.cpp: foo.o`
   (C) `foo.h: foo.cpp foo.o`
   (D) `foo.o: foo.cpp foo.h`

7. Which library container has the best locality of reference ?
   (A) `deque`
   (B) `list`
   (C) `map`
   (D) `vector`

8. What element of a `vector` does `v.end()` point at ?
   (A) `v[1-v.size()]`
   (B) `v[v.size()+1]`
   (C) `v[v.size()-1]`
   (D) `v[v.size()]`

9. Given some arbitrary iterator `i`, what is almost certainly the most efficient way of incrementing it so that it points at the next element in a container ?
   (A) `++i`
   (B) `i++`
   (C) `i+=1`
   (D) `i=i+1`

10. Declare a destructor virtual if any ____ is declared virtual.
    (A) constructor
    (B) member function
    (C) non-member function
    (D) other destructor

11. Given
    ```
    map<string,int> msi;
    for (const auto& i: msi) f(i);
    ```
    and assuming the map has n elements and f runs in $O(1)$ time per element, how long does the `for`-loop take ?
    (A) $O(1)$
    (B) $O(\log_2 n)$
    (C) $O(n)$
    (D) $O(n \log_2 n)$

12. The keyword `explicit` :
    (A) Prevents default members, such as the default copy constructor, from being automatically provided.
    (B) Prevents implicit access to the standard input, output, and error streams.
    (C) Prevents local variables from being destroyed when they go out of scope.
    (D) Prevents constructors from behaving as automatic type conversion functions.

Q : I'm having problems with my Windows software. Will you help me ?
A : Yes. Go to a DOS prompt and type "`format c:`". Any problems you are experiencing will cease within a few minutes.
— Eric S. Raymond
`http://www.catb.org/~esr/faqs/hacker-howto.html`