$Id: cse111-2020q1-midterm.mm,v 1.39 2020-02-21 17:03:53-08 - - $

page 1 page 2 page 3 page 4 Total / 42

Last Name :	
First 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 unreadable answers. Do your scratch work elsewhere and enter your answer into the spaces provided.

1. Write the prototypes (but not the function bodies) as they would appear in the header file for the definition of class **foo** for all of the class members which would otherwise be implicitly generated. **[2✔]**

Scoring :
6 correct : 2 ✔
5 correct : 1½ ✔
4 correct : 1 ✔
3 correct : ½ ✔
else : 0 ✘

```
class foo {
   public:
```

2. Write a function **distance** which counts the number of elements in a range. Do not assume the iterator supports **operator-**. It is at most a forward iterator. The function will use a loop and run in $O(n)$ time. **[2✔]**

```
template <typename iterator>
size_t distance (iterator begin, iterator end) {
```

3. Write a function **average** which returns the average of a sequence of numbers. Assume the iterators point at things which are **double**s or which can be implicitly converted to **double**s. Assume only a forward iterator. **[2✔]**

```
template <typename iterator>
double average (iterator begin, iterator end) {
```

4. Write a function **has**. Its first argument is a **vector<int>** and its second argument is an **int**. It returns true only if the **int** occurs somewhere in the vector, and false otherwise. **[2✔]**

5. The function **find** returns the position of the first occurrence of **n** in **vec**. Make the standard assumption for what to do in the case of not found. **[2✔]**

```
vector<int>::const_iterator find (const vector<int>& vec, int n) {
```

6. Show the prototypes (not function bodies) of `operator++` in its various formats : class member and non-member, unary and binary. It operates on class `foo`. Score : 1/2 point for each exactly correct answer. **[2✔]**

prefix class member	prefix non-member
postfix class member	postfix non-member

7. Define the ***unary*** member function `bigint::operator-` as it would appear in `bigint.cpp`. Reminder : some declarations are listed here : **[1✔]**

```
private:
   ubigint uvalue;
   bool is_negative {false};
public:
   bigint (const ubigint&,
           bool is_negative = false);
```

8. Assume that `foo& foo::operator+= (const foo&)` has been defined as a class member. Write the body of the ***non-member*** `operator+` that adds two values of class `foo` and returns a `foo`. **[2✔]**

9. Assume that the ***prefix*** `foo::operator++` has been defined. Write the body of the ***non-member postfix*** `operator++` that will increment a `foo`. **[2✔]**

10. Write a function `is_sorted` which returns true if the range is sorted into ascending order according to the comparison function. The expression `decltype(*iterator())` is the type of what the iterator points at. A call to `less(a,b)` will return true if `a` should be considered less than `b`. The sequences `{3,9,11,21}`, `{4}`, and `{}` would all be considered sorted for `less<int>`, but the sequences `{5,1,2,8}` and `{5,9,9,11,11}` would not. But `{5,9,9,11,11}` is sorted if `less_equal<int>` is used. Use `less` for comparison, not `operator<`. **[3✔]**

```
template <typename iterator,
          typename less_t = less_equal<decltype(*iterator())>>
bool is_sorted (iterator begin, iterator end, less_t less = less_t()) {
```

11. Write a single constructor for struct **complex** so that the following declarations have the effect described. In addition, the constructor should provide an implicit conversion from a **double** to a **complex**. Use field initializers. The body of the constructor should be an empty **{}**. **[1✔]**
complex a; sets real = imag = 0.0.
complex b (10); sets real = 10.0 and imag = 0.0.
complex c (10, 20); sets real = 10.0 and imag = 20.0.

```
struct complex {
   double real;
   double imag;
```

12. Define the necessaary functions for class **container** and class **container::iterator** so that the following **for**-loop can be compiled. Show prototypes only, not complete bodies.
container cont; for (const auto& i: cont) f(i);

template <typename T> **class container { // [1✔]**	**template <typename T>** **class container::iterator { // [1✔]**

13. Write the complete template function **copy**. It has two template parameters : (1) an input iterator, (2) an output iterator. It has three function parameters : (1) a begin input iterator, (2) an end input iterator, and (3) a begin output iterator. All elements from the input range are copied into the output range. It is assumed that the output iterator is large enough to hold the input range, and may be a back inserter. **[2✔]**

14. Define the function **copy_if** which copies an input range to an output range, but only copies those elements for which the predicate is true. Again, assume the output range is large enough. An example call might be : **[2✔]**
copy_if (vi.begin(), vi.end(), back_inserter(out), [](int i){ return i > 0; });

```
template <typename in_itor, typename out_itor, typename predicate>
void copy_if (in_itor begin, in_itor end, out_itor out, predicate ok) {
```

15. Define the template class **queue**. It has the public functions **push_back**, **pop_front**, **front** (in both the constant and non-constant versions), **size**, and **empty**. It has a private field of the template class **std::deque**. Code each function inline with a direct call to the corresponding **std::deque** function. Note : **deque** has all of these functions with the same names. **[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. If `void f (size_t n, int* a);` is a C function, what declaration allows `f (x.size(), &x[0]);` ?
   (A) `deque<int> x;`
   (B) `list<int> x;`
   (C) `map<int,int> x;`
   (D) `vector<int> x;`

2. What should a search function do to indicate failure to find ?
   (A) Print an error message and exit the program.
   (B) Return false.
   (C) Return the end iterator.
   (D) Throw a not found exception.

3. Given the function `f` what is automatically called on `x` when `f` returns ?
   `void f() { foo x; }`
   (A) the constructor
   (B) the destructor
   (C) the function free
   (D) the garbage collector

4. Given an iterator `i`, What is the preferred (probably most efficient) way of incrementing it in the third part of a `for`-loop, and which will work for all iterators, as in
   `for (i = v.begin(); i != v.end; _____)`
   (A) `i=i+1`
   (B) `i+=1`
   (C) `i++`
   (D) `++i`

5. What keyword in a base class is used to specify that when a function is called, the actual function to be called must be looked up on a table at the time of the function call ?
   (A) `abstract`
   (B) `override`
   (C) `protected`
   (D) `virtual`

6. If two classes need to access all of the parts of each other, but neither is part of the same class hierarchy, what declaration is needed ?
   (A) `friend`
   (B) `private`
   (C) `protected`
   (D) `public`

7. Inside a virtual member function of class `foo`, what is the type of `this` ?
   (A) `foo`
   (B) `foo&`
   (C) `foo&&`
   (D) `foo*`

8. Which of the following specifies that the parameter to `f` is an rvalue reference ?
   (A) `void f (string&&);`
   (B) `void f (string&);`
   (C) `void f (string);`
   (D) `void f (string*);`

9. An `unordered_map` is implemented as a :
   (A) balanced binary search tree
   (B) hash table
   (C) linked list
   (D) radix trie structure

10. At which element of a vector does `v.rbegin()` point ?
    (A) `v[-1]`
    (B) `v[0]`
    (C) `v[v.size()-1]`
    (D) `v[v.size()]`

11. What cast might be used to convert a `uintptr_t` to a `char*` ?
    (A) `const_cast`
    (B) `dynamic_cast`
    (C) `reinterpret_cast`
    (D) `static_cast`

12. What is the amount of time taken to insert, find, or remove an element from a `std::map` ?
    (A) $O(1)$
    (B) $O(\log n)$
    (C) $O(n)$
    (D) $O(n \log n)$

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`