

CSE-111 0 Advanced Programming 0 Fall 2020 0 Midterm Exam 1 of 1

\$Id: cse111-2020q4-midterm.tt,v 1.4 2020-11-17 18:26:23-08 - - \$

.PS

examboxes(4)

.PE

.EQ

delim \$\$

.EN

Assume, do not show, #include directives. Neatness counts. Use proper indentation.

1. Given the following contents of stack.h, show the implementation of push in stack.cpp [4pt]

```
class stack {
private:
    struct node {
        int val;
        node* link;
    };
    node* top;
public:
    void push (int n);
};
```

2. Write a function called sum. It has a single template argument of iterator type. It has two function arguments of that iterator type indicating the begin and the end of a range. Sum returns a double which is the sum of all elements in the range. Assume the iterators point at elements of type double, or whatever can be implicitly converted to type double. Code your solution as a loop using the two-semicolon version of the for loop. [4pt]
3. Write a template function is\_monotonic. The template type argument is an iterator type. It has two function arguments, begin and end, of that iterator type. It returns a bool value which is true if and only if the sequence is sorted in monotonically increasing order. Assume the elements pointed at by the iterators have operator< defined for them as the comparison operator. E.g., {1,2,6,99} and {3,6,8,42} are monotonic, but neither {3,5,5,7} nor {5,2,8,7} are. Monotonic means every number is larger than the preceding one. [4pt]
4. Write a function reverse whose argument is a vector<int>. E.g., the reverse of {1,3,5,7,9} is {9,7,5,3,1}. It must work for an empty vector, and also for an even or odd number of elements in the vector. [4pt]

5. Write a function `largest` which takes a `list<int>` as its argument. It returns a `list<int>::iterator` to indicate the position of the largest int in the list. Return the end iterator for an empty list. Your function must work on any list of zero, one, or more ints. [4pt]
7. Assume you have files `foo.cpp`, `bar.cpp`, and `main.cpp`. Write a Makefile which will build the executable binary named "runfile". Use wildcards as appropriate. The first target should be `all`. Don't bother thinking about any header files. Just use `g++` to compile. [4pt]
8. Indicate, using big-O notation, the amount of time it takes to find an arbitrary element in each of: `vector`, `unordered_map`, `map`, `list`. For each of `map` and `unordered_map`, what is the underlying data structure? [4pt]
9. Code a template operator<< which takes any kind of a vector and prints out the contents of the vector in the order `v[0]`, `v[1]`, `v[2]`, ..., to the end. Assume operator<< is defined for the elements of the vector. Print exactly one space between vector elements, but no space before the first or after the last element. [4pt]
10. Given that `ubigint` has a field called `vector<unsigned char>` `ubigvalue`, code the non-const function `ubigint::multiply_by_2`, which modifies `ubigvalue`. [4pt]
11. Define the function `find_if` whose template arguments are an iterator type and a predicate type and whose function arguments are a pair of 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. [4pt]

SCORE-TOTAL=40

\$Id: soln-cse111-2020q4-midterm.txt,v 1.16 2020-11-17 18:59:13-08 - - \$  
Solution to CSE-111 2020q1-midterm, page 1

Any logically correct variation on these answers is ok too.  
Assign fractional points in increments of 0.5 per question.

Don't cut/paste code from student into a file to verify that it does compile, but if you detect syntax or semantic errors and know that it won't compile in a suitable environment, then deduct points.

Do not deduct points for obvious typos.

---

Question 1. [4]

```
void stack::push (int n) {
    node* t = new node();
    t->val = n;
    t->link = top;
    top = t;
}
```

---

Question 2. [4]

```
template <typename itor>
double sum (itor begin, itor end) {
    double s = 0;
    for (; begin != end; ++begin) s += *begin;
    return s;
}
```

---

Question 3. [4]

```
template <typename itor>
bool is_monotonic (itor begin, itor end) {
    if (begin == end) return true;
    itor prev = begin++;
    for (; begin != end; prev = begin++) {
        if (not (*prev < *begin)) return false;
    }
    return true;
}
```

Must work on empty container, for which result is true.

---

Question 4. [4]

```

void reverse (vector<int>& v) { ///// soln 1
    for (ssize_t b = 0, e = v.size() - 1; b < e; ++b, --e) {
        swap (v[b], v[e]);
    }
}

vector<int> reverse (const vector<int>& v) { ///// soln 2
    size_t s = v.size();
    vector<int> r (s);
    for (size_t i = 0; i < v.size(); ++i) {
        r[s - 1 - i] = v[i];
    }
    return r;
}

vector<int> reverse (const vector<int>& v) { ///// soln 3
    vector<int> r;
    for (auto i = v.rbegin(); i != v.rend(); ++i) {
        r.push_back (*i); // note i is a reverse_iterator.
    }
    return r;
}

```

Or three or four other alternatives.

Deduct 1 point from soln 2 of vector r is not properly dimensioned.

Deduct 1 point if argument is not non-const reference, but if their solution is to return a vector<int>, be sure the argument is a const.

Deduct 1 point if used size\_t in soln 1 because that would overflow if size() == 0.

OK to use size\_t if a special zero check is made before the loop.

OK to use ssize\_t or int or long for control variable.

---

Question 5. [4]

```

list<int>::iterator largest (list<int>& lst) {
    auto begin = lst.begin();
    if (begin == lst.end()) return lst.end();
    auto largest = begin++;
    for (; begin != lst.end(); ++begin) {

```

```
    if (*largest < *begin) largest = begin;
}
return largest;
}
```

---

Question 7. [4]

```
SOURCES = foo.cpp bar.cpp main.cpp
OBJECTS = ${SOURCES:.cpp=.o}
```

```
all: runfile
```

```
runfile: ${OBJECTS}
        g++ ${OBJECTS} -o runfile
```

```
%.o: %.cpp
        g++ -c $<
```

In the runfile target, in the indented command line, may use:

\$@ instead of runfile

\$^ instead of \${OBJECTS}

---

Question 8. [4]

```
vector          O(n)
unordered_map    O(1)
map              O(log n)
list             O(n)
```

```
map              balanced binary search tree, red-black tree, etc.
unordered_map    hash table
```

---

Question 9. [4]

```
template <typename T>
ostream& operator<< (ostream& out, const vector<T>& vec) {
    bool space = false;
    for (const auto& item: vec) {
        if (space) out << " ";
        space = true;
        out << item;
    }
    return out;
}
```

OK to use variations on how spaces are handled, as long

as no spaces before or after.

Deduct 1/2 point if vec is not const reference.

---

Question 10. [4]

```
void ubigint::multiply_by_2() {
    int carry = 0;
    for (i = 0; i < ubigvalue.size(); ++i) {
        int n = ubigvalue[i] * 2 + carry;
        ubigvalue[i] = n % 10;
        carry = n / 10;
    }
    if (carry != 0) ubigvalue.push_back (carry);
}
```

OK to use unsigned char instead of int for carry, etc.

---

Question 11. [4]

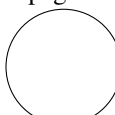
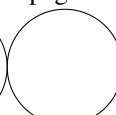
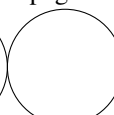
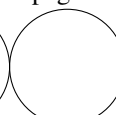

```
template <typename iterator, class predicate>
iterator find_if (iterator begin, iterator end, predicate pred) {
    for (; begin != end; ++begin) {
        if (pred (*begin)) break;
    }
    return begin;
}
```

OK if iterators are passed as const, provided that a local copy is made of the begin iterator.

OK if special check for begin == end.

Deduct 1 point if the result for not found is other than end.

\$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.**

- 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:
```

- 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) {
```

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

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

- 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✓]**

- 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 necessary 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		$\times 1 =$	$= a$
number of wrong answers		$\times \frac{1}{2} =$	$= b$
number of missing answers		$\times 0 =$	0
column total $c = \max(a - b, 0)$	12		$= c$

- 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;`
- 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.
- 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
- 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`
- 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`
- 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`
- Inside a virtual member function of class `foo`, what is the type of `this` ?  
 (A) `foo`  
 (B) `foo&`  
 (C) `foo&&`  
 (D) `foo*`
- 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*);`
- An `unordered_map` is implemented as a :  
 (A) balanced binary search tree  
 (B) hash table  
 (C) linked list  
 (D) radix trie structure
- 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()]`
- 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`
- 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>

\$Id: soln-cse111-2020q1-midterm.txt,v 1.5 2020-02-07 14:12:01-08 - - \$  
Solution to CSE-111 2020q1-midterm, page 1

Any logically correct variation on these answers is ok too.  
Assign fractional points in increments of 1/2 per question.

---

Question 1. [2]

```
foo();  
foo (const foo&);  
foo (foo&&);  
foo& operator= (const foo&);  
foo& operator= (foo&&);  
~foo();
```

---

Question 2. [2]

```
template <typename iterator>  
size_t distance (iterator begin, iterator end) {  
    size_t dist = 0;  
    while (begin != end) { ++begin; ++dist; }  
    return dist;  
}
```

---

Question 3. [2]

```
template <typename iterator>  
double average (iterator begin, iterator end) {  
    size_t count = 0; double sum = 0.0;  
    while (begin != end) { sum += *begin++; ++count; }  
    return sum / count;  
}  
//// empty range: 0.0/0 == 0.0/double(0) == nan  
//// ok if throw domain_error or something else for empty range
```

---

Question 4. [2]

```
bool has (const vector<int>& vec, int n) {  
    for (auto i: vec) if (i == n) return true;  
    return false;  
}
```

---

Question 5. [2]

```
vector<int>::const_iterator find (const vector<int>& vec, int n) {
    auto itor = vec.cbegin();
    for (auto i = vec.cbegin(); i != vec.cend(); ++i) {
        if (*i == n) return i;
    }
    return vec.cend();
}
```

Solution to CSE-111 2020q1-midterm, page 2

---

Question 6. [2]

prefix class member	prefix non-member
foo& operator++();	foo& operator++ (foo&);
postfix class member	postfix non-member
foo operator++ (int);	foo operator++ (foo&, int);

---

Question 7. [1]

```
bigint bigint::operator- () const {
    return {uvalue, not is_negative};
}
```

---

Question 8. [2]

```
foo operator+ (const foo& left, const foo& right) {
    foo result = left;
    return result += right;
}
```

---

Question 9. [2]

```
foo opeator++ (foo& x, int) {
    foo res = x;
    ++x;
    return res;
}
```

---

Question 10. [3]

```
template <typename iterator,
          typename less_t = less_equal<decltype(*iterator())>>
```

```
bool is_sorted (iterator begin, iterator end, less_t less = less_t()) {
    if (begin == end) return true;
    for (;;) {
        auto curr = begin++;
        if (begin == end) break;
        if (not less (*curr, *begin)) return false;
    }
    return true;
}
```

Solution to CSE-111 2020q1-midterm, page 3

---

Question 11. [1]

```
complex (double re = 0.0, double im = 0.0): real(re), imag(im) {}
```

---

Question 12. [2]

template <typename T>	template <typename T>
class container {	class container<T>::iterator {
class iterator;	bool operator!= (iterator);
iterator begin();	iterator& operator++();
iterator end();	T& operator*();
};	};

---

Question 13. [2]

```
template <typename in_itor, typename out_itor>
void copy (in_itor begin, in_itor end, out_itor obegin) {
    while (begin != end) *obegin++ = *begin++;
}
```

---

Question 14. [2]

```
template <typename in_itor, typename out_itor, typename predicate>
void copy_if (in_itor begin, in_itor end, out_itor out, predicate ok) {
    for (; begin != end; ++begin) {
        if (ok (*begin)) *out++ = *begin;
    }
}
```

---

Question 15. [3]

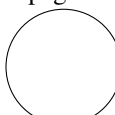
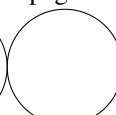
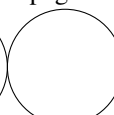
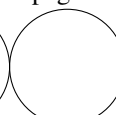

```
template <typename T>
```

```
class queue {
    private:
        deque<T> data;
    public:
        void push_back (const T& x) { data.push_back (x); }
        void pop_front() { data.pop_front(); }
        T& front() { return data.front(); }
        const T& front() const { return data.front(); }
        size_t size() { return data.size(); }
        bool empty() { return empty(); }
};
```

Solution to CSE-111 2020q1-midterm, page 4

1. (D) vector<int> x;
2. (C) Return the end iterator.
3. (B) the destructor
4. (D) ++i
5. (D) virtual
6. (A) friend
7. (D) foo\*
8. (A) void f (string&&);
9. (B) hash table
10. (C) v[v.size()-1]
11. (C) reinterpret\_cast
12. (B)  $\Theta(\log n)$

\$Id: cse111-2019q4-midterm.mm,v 1.132 2019-11-01 15:39:56-07 - - \$

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 messy or unreadable answers. Do your scratch work elsewhere and enter ONLY your final answer into the spaces provided, and ONLY in the spaces provided.*

- Write a complete C++ program : everything that needs to be in the file **echo.cpp**, which implements the **echo(1)** command. It prints out each of its command line arguments with exactly one space in between arguments, but no space before the first argument or after the last argument. **[4✓]**

```
-bash-1$ echo hello world
hello world
-bash-2$ echo

-bash-3$ echo foo bar baz qux
foo bar baz qux
```

- Complete the function **trim** which will ensure a **ubigint** is in canonical form. Assume a field declared as **vector<udigit\_t> ubigvalue;** **[1✓]**  
**void ubigint::trim() {**
- Code a template function **print** which prints to **cout** all of the elements of a container, one element per line. Assume **operator<<** is defined for the elements. The function should work on any container, such as **list**, **vector**, etc. It has a single argument, which is a container.
  - Code **print** using the colon version of a **for**-loop. Make only implicit, not explicit, use of an iterator. **[2✓]**
  - Code **print** using the three-part version of a **for**-loop using the container's **cbegin** and **cend** functions. Assume the container has only forward iterators (not bidirectional, not random access). **[3✓]**

4. Code the function `equal_range`, whose template arguments are two types of forward iterators and a comparison function type, and whose function arguments are two begin and end ranges and an equal comparison function object. It returns true if all elements in the ranges are equal and both ranges have the same number of elements. [3✓]

```
template <typename itor1, typename itor2, typename equal_t>
bool equal_range (itor1 begin1, itor1 end1, itor2 begin2, itor2 end2,
                  equal_t equal) {
```

5. 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;
};
```

6. Write the function `ubigint::divide_by_2` which divides by 2 the value of a `ubigint` as per the project specifications. Update the value in place and do not call any other members of `ubigint`. [2✓]

<pre>class ubigint { private:     vector&lt;unsigned char&gt; value; public:     void divide_by_2() { };</pre>	<pre>void ubigint::divide_by_2() {</pre>
--	--

7. Write a function `dup_adjacent` which searches a range and returns true if any two adjacent (next to each other) elements are equivalent, based on the comparison parameter. For example, the sequence {1,2,3,3,4,5} returns true, while {8,3,7,5,9,3} returns false, because the two 3s are not adjacent. The function must run in  $O(n)$  time. [3✓]

```
template <typename itor_t, typename equal_t>
bool dup_adjacent (itor_t begin, itor_t end, equal_t equal) {
```



8. Write a function `minimum_itor` whose argument is a constant reference to a `vector<double>` and which returns a `vector<double>::iterator` pointing at the minimum element in the vector. If there are more than one element equal to the minimum, point the iterator at the element closest to the beginning of the range. Make an appropriate assumption if the vector is empty. [2✓]

9. There are six automatically generated class members. The default constructor and the destructor are given here for class `foo`. List the prototypes (signatures) of the others as they would appear in a class header file.

<pre>class foo { public:</pre>	(a) The copiers. [1✓]
<pre>    foo();     ~foo();</pre>	(b) The movers. [1✓]

10. Given the definition of `tstack` as shown at the left, show the implementation of the class as it would appear in outside of the class definition itself. (Remember your data structures course?)

```
template <typename item_t>
class tstack {
private:
    struct node;
    using node_ptr =
        shared_ptr<node>;
    struct node {
        item_t value;
        node_ptr link;
    };
    node_ptr top_ptr;
public:
    bool empty() const {
        return top_ptr == nullptr;
    }
    const item_t& top() const;
    void pop();
    void push (const item_t&);
};
```

```
template <typename item_t>
const item_t& tstack<item_t>::top() const { // (a) [1✓]
}

template <typename item_t>
void tstack<item_t>::pop() { // (b) [1✓]
}

template <typename item_t>
void tstack<item_t>::push (const item_t& n) { // (c) [2✓]
}

}
```

11. Write a program that will copy the standard input to the standard output. Use `getline` for input. [2✓]

```
#include <iostream>
#include <string>
using namespace std;
int main() {
```

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		$\times 1 =$	$= a$
number of wrong answers		$\times \frac{1}{2} =$	$= b$
number of missing answers		$\times 0 =$	0
column total $c = \max(a - b, 0)$	12		$= c$

- What term is used to describe a virtual function in a derived class that has the same signature as a function of the same name in its base class?
  - delete
  - inherit
  - overload
  - override
- What is the best parameter description for a function that sums a vector, but does not change any of its elements? (Fill in the blank.)  
`double sum (____);`
  - `const vector<double>`
  - `const vector<double>&`
  - `vector<double>`
  - `vector<double>&`
- Which data structure will require special handling if managed by `shared_ptrs`, in order to avoid memory leak?
  - arbitrary  $n$ -way tree
  - balanced binary search tree
  - cyclic graph
  - linear linked list
- Which data structure should be used to maintain a table whose operations are insert, remove, and find by key, without any need to iterate in any particular order, in order to minimize  $O(f(n))$  time for these operations?
  - `list<pair<string, string>>`
  - `map<string, string>`
  - `unordered_map<string, string>`
  - `vector<pair<string, string>>`
- What usually follows `#include <iostream>` so that it is possible to use the name `cout` instead of always using `std::cout`?
  - `#include <std::packages>`
  - `import namespace std;`
  - `typedef std::namespace;`
  - `using namespace std;`
- When it is necessary to print an object of a particular class via `operator<<`, what keyword is generally used in the declaration of `operator<<`?
  - `friend`
  - `private`
  - `protected`
  - `public`
- Given classes `circle` and `square`, both of which inherit from abstract class `shape`, how should `area` be defined in the class definition of `shape`, knowing that it requires a concrete object in order to compute the area?
  - `virtual double area() = 0;`
  - `virtual double area() = default;`
  - `virtual double area() = delete;`
  - `virtual double area() {return nullptr;}`
- What is the link step in a `Makefile`?
  - `$(EXEC) : $(OBJS)`  
`$(GPP) -o $(EXEC) $(OBJS)`
  - `%.o : %.cpp`  
`$(GPP) -c $<`
  - `$(OBJS) : $(CPPSRC)`  
`$(GPP) -o $(OBJS) $(CPPSRC)`
  - `$(EXEC) : $(CPPSRC)`  
`$(GPP) -c $@`
- Given that `i` is an object of some iterator class, what is assumed to be the most efficient way to increment it in the third part of a `for`-loop?
  - `++i`
  - `i++`
  - `i+=1`
  - `i=i+1`
- Given the declaration `foo x;`, and assuming that `foo` has a non-static field called `n`, in the expression `x.f()`, how would the function `f` refer to the field `n`?
  - `foo::n`
  - `this->n`
  - `this.n`
  - `this::n`
- Which of the following must be declared in every C++ program?
  - `int main (char**, int);`
  - `int main (int, char**);`
  - `int main (string&);`
  - `int main (vector<string>&);`
- What should be the first non-comment line in the file `foo.h`?
  - `#define __FOO_H__`
  - `#ifdef __FOO_H__`
  - `#ifndef __FOO_H__`
  - `#include __FOO_H__`

\$Id: soln-cse111-2019q4-midterm.txt,v 1.1 2019-11-15 15:20:34-08 - - \$  
Solution to CMPS-109 2019q3-midterm, page 1

Any logically correct variation on these answers is ok too.  
Assign fractional points in increments of 1/2 per question.

---

Question 1. [4]

```
#include <iostream>
using namespace std;
int main (int argc, char** argv) {
    for (int i = 1; i < argc; ++i) {
        if (i > 1) cout << " ";
        cout << argv[i];
    }
    cout << endl;
    return 0;
}
```

---

Question 2. [1]

```
void ubigint::trim() {
    while (ubigvalue.size() > 0 and ubigvalue.back() == 0)
        ubigvalue.pop_back();
}
///// or: ubigvalue[ubigvalue.size() - 1] != 0
```

---

Question 3(a). [2]

```
template <typename container>
void print (const container& c) {
    for (const auto& i: c) cout << i << endl;
}
```

---

Question 3(b). [3]

```
template <typename container>
void print (const container& c) {
    for (auto p = c.cbegin(); p != c.cend(); ++p) {
        cout << *p << endl;
    }
}
```

Solution to CMPS-109 2019q3-midterm, page 2

---

## Question 4. [3]

```
template <typename itor1, typename itor2, typename equal_t>
bool equal_range (itor1 begin1, itor1 end1,
                  itor2 begin2, itor2 end2,
                  equal_t equal) {
    while (begin1 != end1 and begin2 != end2) {
        if (not equal (*begin1, *begin2)) return false;
        ++begin1; ++begin2;
    }
    return begin1 == end1 and begin2 == end2;
}
```

---

Question 5. [2]

```
bool bigint::operator< (const bigint& that) const {
    if (is_negative) {
        if (not that.is_negative) return true;
        else return that.uvalue < uvalue;
    } else { // is positive
        if (that.is_negative) return false;
        else return uvalue < that.uvalue;
    }
}
```

---

Question 6. [2]

```
void ubigint::divide_by_2() {
    if (value.size() == 0) return;
    for (size_t i = 0; i < value.size() - 1; ++i) {
        value[i] /= 2;
        if (value[i + 1] % 2 == 1) value[i] += 5;
    }
    value.back() /= 2;
    if (value.back() == 0) value.pop_back();
}
///// or: odd check: (value[i + 1] & 1)....
///// or: value[value.size() - 1] /= 2;
```

---

Question 7. [3]

```
template <typename itor_t, typename equal_t>
bool dup_adjacent (itor_t begin, itor_t end, equal_t equal) {
    if (begin == end) return false;
```

```
    for (auto prev = begin++; begin != end; prev = begin++) {  
        if (equal (*begin, *prev)) return true;  
    }  
    return false;  
}
```

Solution to CMPS-109 2019q3-midterm, page 3

---

Question 8. [2]

```
vector<double>::iterator minimum_itor (const vector<double>& v) {  
    auto min = v.begin();  
    for (auto i = v.begin(); i != v.end(); ++i) {  
        if (*min < *i) min = i;  
    }  
    return min;  
}
```

---

Question 9. // assign 1/2 point for each correct prototype  
// if in the correct box.

```
copiers:  foo (const foo&);  
          foo& operator= (const foo&);  
movers:  foo (foo&&);  
          foo& operator= (foo&&);
```

---

Question 10(a). [1]

```
template <typename item_t>  
const item_t& tstack<item_t>::top() const {  
    return top_ptr->value;  
}
```

---

Question 10(b). [1]

```
template <typename item_t>  
void tstack<item_t>::pop() {  
    top_ptr = top_ptr->link;  
}
```

---

Question 10(c). [2]

```
template <typename item_t>  
void tstack<item_t>::push (const item_t& n) {  
    node_ptr t = make_shared<node>();  
    t->value = n;  
    t->link = top_ptr;  
    top_ptr = t;  
}
```

---

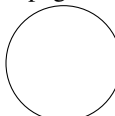
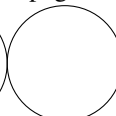
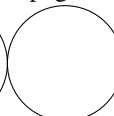
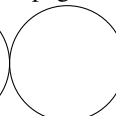

Question 11. [2]

```
int main() {
    string line;
    for(;;) {
        getline (cin, line);
        if (cin.eof()) break;
        cout << line << endl;
    }
    return 0;
}
```

Solution to CMPS-109 2019q3-midterm, page 4

1. (D) override
2. (B) `const vector<double>&`
3. (C) cyclic graph
4. (C) `unordered_map<string,string>`
5. (D) `using namespace std;`
6. (A) friend
7. (A) `virtual double area() = 0;`
8. (A) `${EXEC} : ${OBJS}`  
 `${GPP} -o ${EXEC} ${OBJS}`
9. (A) `++i`
10. (B) `this->n`
11. (B) `int main (int, char**);`
12. (C) `#ifndef __FOO_H__`

\$Id: cmps109-2019q3-midterm.mm,v 1.106 2019-07-22 16:25:18-07 - - \$

page 1	page 2	page 3	page 4	Total / 42	PLEASE PRINT CLEARLY :				
					<table border="1"> <tr> <td colspan="2">Name :</td> </tr> <tr> <td>CruzID :</td> <td>@ucsc.edu</td> </tr> </table>	Name :		CruzID :	@ucsc.edu
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. Finish the implementation of the function `minimum`, which returns an iterator pointing at the minimum element in the range. Assume that the items pointed at by the iterators have `operator<` available. [2✓]

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

2. Finish the implementation of the function `find`, which returns true if some element in the range is equal to the third argument. Assume that the items pointed at by the iterators have `operator==` available. [2✓]

```
template <typename iterator, typename item_t>
bool find (iterator begin, iterator end, item_t item) {
```

3. Write a non-template function called `sum`. The argument is a `vector<double>` and the result is a `double`. [2✓]

4. Write a template function called `tsum`, whose template argument is an arbitrary type of iterator, and whose function arguments are a pair of iterators indicating a range. Its result is a `double`. Assume the iterators point at doubles. [2✓]

5. Implement `ubigint::trim`, a member function which ensures that a `ubigint` is in canonical form. Canonical form means that there are no leading zeros and that zero is represented by an empty vector. The name of the vector field is `ubig_value`. [2✓]

6. Assume that `bigint::operator<` has been implemented. Code a non-template non-member `operator>` whose arguments are two `bigints`. [1✓]

7. Assume class `foo` has a prefix member `operator++` defined for it. Define the postfix `operator++` as a non-member which makes use of `foo::operator++()`. Assume `foo` has a copy ctor. [2✓]

8. Circle.

- (a) Write the function `circle`, which returns, as a `pair`, the area (as the first field) and circumference (as the second field) of an circle, given its radius. Use `M_PI` from `<cmath>`. All numbers are `double`. For the mathematically unsophisticated,  $A = \pi r^2$  and  $C = 2\pi r$ . [1✓]

- (b) Given the declaration `auto x = circle (e);` using `x`, write one line of code to print out the area and circumference. The line of output (assuming appropriate data) should look like the following: [1✓]  
area = 23.2134, circumference = 17.0795

9. Write a `copy` function which takes forward iterators pointing into an input container, and a forward iterator pointing at an output container. Copy the items from the input container to the output container. Assume the output iterator is such that the output container does not overflow or cause memory errors. [2✓]

```
template <typename in_itor, typename out_itor>
void copy (in_itor in_begin, in_itor in_end, out_itor out_begin) {
```

10. Define `equal_range` which takes two pairs of forward iterators and returns true if all of the elements of the ranges are equal and the ranges are of the same length. There is no `size()` function, and forward iterators can not be subtracted. Assume the elements of the range have `operator==`. [3✓]

```
template <typename itor1, typename itor2>
bool equal_range (itor1 begin1, itor1 end1, itor2 begin2, itor2 end2) {
```



11. Assuming `string s;` and `string t;` write an expression (not a complete statement) equivalent to `s == t`, but use only `operator<` and no other comparison operators. Use some combination of the operators `and (&&)`, `or (||)`, and `not (!)`. Use parentheses as appropriate. [1✓]

12. Code a fragment of a **Makefile** which will build an object (`.o`) file from a C++ (`.cpp`) source file. Show the wildcard target, prerequisite, and command. Assume the following macro has already been defined. [1✓]

```
COMPILECPP = g++ -std=gnu++17 -g -O0 -Wall -Wextra -Wold-style-cast
```

13. Code a non-member `multiply_by_2` function which is useable by `ubigint` which will double the value represented by a vector argument. Write code as a loop manipulating each digit individually. Do not call any functions from `multiply_by_2`. The value is to be updated in place. [3✓]

```
void multiply_by_2 (vector<unsigned char>& value) {
```

14. Define the function `print`. It prints out all of the elements in a range assuming that `operator<<` is defined for the elements in the range. A single space is printed out between successive elements of the range, but no space is printed before the first or after the last element. [2✓]

```
template <typename iterator>
void print (iterator begin, iterator end) {
```

15. The Collatz conjecture states that for any arbitrary positive integer  $n$ , if  $n$  is even, replace  $n$  by  $n/2$ , but if  $n$  is odd, replace  $n$  by  $3n + 1$ . Repeat until  $n = 1$ , and the procedure will terminate. Write the non-template function `collatz`. Its argument is of type `int` and it returns a `vector<int>` containing a trace of a sequence of collatz numbers. The resulting vector will contain all of the numbers in sequence, starting from the original argument, and ending with the number 1. Here is a sample program that calls `collatz`, followed by its output. [3✓]

```
int main() {
    vector<int> a {30, 5, 11};
    for (int i: a) {
        vector<int> v = collatz (i);
        for (int j: v) cout << " " << j;
        cout << endl;
    }
}
-bash-$ ./collatz
30 15 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
5 16 8 4 2 1
11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
```

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		$\times 1 =$	$= a$
number of wrong answers		$\times \frac{1}{2} =$	$= b$
number of missing answers		$\times 0 =$	0
column total $c = \max(a - b, 0)$	12		$= c$

- Multiplication of an  $m$ -digit number by an  $n$ -digit number takes how many single-digit multiplications?
  - $O(m * n)$
  - $O(m + n)$
  - $O(m - n)$
  - $O(m / n)$
- How should the function **f** be declared so that it accepts a **string** by constant reference?
  - `void f (const string&);`
  - `void f (const string);`
  - `void f (const string*);`
  - `void f (const string->);`
- Memory mangement using **shared\_ptr** needs special handline for what kind of data structure?
  - graph possibly with cycles
  - graph with no cycles
  - linear non-circular linked list
  - vector of pointers to objects
- It is expected that **v.size()** should return a value equivalent to:
  - `v.begin() - v.end()`
  - `v.begin() - v.end() + 1`
  - `v.end() - v.begin()`
  - `v.end() - v.begin() + 1`
- What keyword will allow another arbitrary class or function access to all members of the class?
  - friend**
  - private**
  - protected**
  - public**
- What keyword will allow access only to members of the class, but also to any class derived from it?
  - friend**
  - private**
  - protected**
  - public**
- What will produce an object (**.o**) file but suppress linking to an executable binary?
  - `g++ -MM`
  - `g++ -c`
  - `g++ -g`
  - `g++ -o`
- How long does it take to insert, delete, or find elements of a **map**?
  - $O(1)$
  - $O(\log n)$
  - $O(n)$
  - $O(n \log n)$
- How long does it take to insert, delete, or find elements of an **unordered\_map**?
  - $O(1)$
  - $O(\log n)$
  - $O(n)$
  - $O(n \log n)$
- What is the correct (and preferred) way to link down a list?
  - `while (p != 0)`
  - `while (p != NULL)`
  - `while (p != null)`
  - `while (p != nullptr)`
- Given the expression **p->f(what)**, inside the code for the function **f**, what expression is used to refer to the value of **p**?
  - self**
  - that**
  - this**
  - what**
- When an object belonging to a class goes out of scope, what is automatically called?
  - the constructor
  - the destructor
  - the garbage collector
  - the **free** function

## GRADE INFLATION



\$Id: soln-cmps109-2019q3-midterm.txt,v 1.10 2019-07-16 13:53:17-07 - - \$  
Solution to CMPS-109 2019q3-midterm, page 1

Any logically correct variation on these answers is ok too.  
Assign fractional points in increments of 1/2 per question.

---

Question 1. [2]

```
template <typename iterator>
iterator minimum (iterator begin, iterator end) {
    if (begin == end) return end;
    auto smallptr = *begin++;
    for (; begin != end; ++begin) {
        if (*begin < *smallptr) smallptr = begin;
    }
    return smallptr;
}
```

---

Question 2. [2]

```
template <typename iterator, typename item_t>
bool find (iterator begin, iterator end, item_t item) {
    for (; begin != end; ++begin) if (*begin == item) return true;
    return false;
}
```

---

Question 3. [2]

```
double sum (const vector<double>& vec) {
    double s = 0;
    for (double d: vec) s += d; //// OK to use 3-part for loop
    return s;
}
```

---

Question 4. [2]

```
template <typename iterator>
double tsum (iterator begin, iterator end) {
    double s = 0;
    for(; begin != end; ++begin) s += *begin;
    return s;
}
```

---

Question 5. [2]

```
void ubigint::trim() {
    while (ubig_value.size() > 0 and ubig_value.back() == 0)
        ubig_value.pop_back();
}
```

}

## Solution to CMPS-109 2019q3-midterm, page 2

---

Question 6. [1]

```
bool operator> (const bigint& a, const bigint& b) {
    return b < a;
}
```

---

Question 7. [2]

```
foo operator (foo& that, int) {
    foo result = that;
    ++that;
    return result;
}
```

---

Question 8(a). [1]

```
pair<double,double> circle (double radius) {
    return {M_PI * radius * radius, 2 * M_PI * radius};
}
///// ok: return pair<double,double> (... , ...)
///// ok: return pair (... , ...)
///// ok: use temp variable.
```

---

Question 8(b). [1]

```
cout << "area = " << x.first << ", circumference = "
    << x.second << endl;
```

---

Question 9. [2]

```
template <typename in_itor, typename out_itor>
void copy (in_itor in_begin, in_itor in_end, out_itor out_begin) {
    for (; in_begin != in_end; ++begin) *out_begin++ = *in_begin;
}
///// ok: while (in_begin != in_end) *out_begin++ = *in_begin++;
```

---

Question 10. [3]

```
template <typename itor1, typename itor2, typename equal_t>
bool equal_range (itor1 begin1, itor1 end1, itor2 begin2, itor2 end2) {
    while (begin1 != end1 and begin2 != end2) {
        if (*begin1 != *begin2) return false;
        ++begin1;
        ++begin2;
    }
}
```

```
    return begin1 == end1 and begin2 == end2;
}
```

Solution to CMPS-109 2019q3-midterm, page 3

---

Question 11. [1]

```
not (s < t) and not (t < s)
///// ok: not (s < t or t < s)
```

---

Question 12. [1]

```
%.o : %.cpp
    ${COMPILECPP} -c $<
```

---

Question 13. [3]

```
void multiply_by_2 (vector<unsigned char>& value) {
    unsigned char carry = 0;
    for (size_t i = 0; i < value.size(); ++i) {
        unsigned char d = value[i] * 2 + carry;
        value[i] = d % 10;
        carry = d / 10;
    }
    if (carry != 0) value.push_back (carry);
}
```

---

Question 14. [2]

```
template <typename iterator>
void print (iterator begin, iterator end) {
    string sp = "";
    for(; begin != end; ++begin) {
        cout << sp << *begin;
        sp = " ";
    }
}
```

---

Question 15. [3]

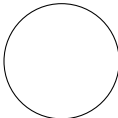
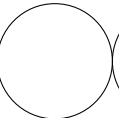
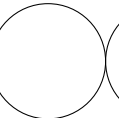
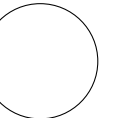
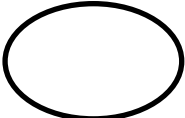
```
vector<int> collatz (int n) {
    vector<int> v;
    v.push_back (n);
    while (n != 1) {
        if (n % 2 == 1) n = 3 * n + 1;
        else n = n / 2;
        v.push_back (n);
    }
    return v;
}
```

```
} ///// better odd test:  if (n & 1)...
```

Solution to CMPS-109 2019q3-midterm, page 4

1. (A)  $\$ O ( m * n ) \$$
2. (A) `void f (const string&);`
3. (A) graph possibly with cycles
4. (C) `v.end() - v.begin()`
5. (A) friend
6. (C) protected
7. (B) `g++ -c`
8. (B)  $\$ O ( \log n ) \$$
9. (A)  $\$ O ( 1 ) \$$
10. (D) `while (p != nullptr)`
11. (C) `this`
12. (B) the destructor

\$Id: cmips109-2019q2-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 :				
					<table border="1"> <tr> <td>Name :</td> <td></td> </tr> <tr> <td>CruzID :</td> <td>@ucsc.edu</td> </tr> </table>	Name :		CruzID :	@ucsc.edu
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.*

- Assuming `bigint& bigint::operator+= (const bigint&);`  
Show the complete code for the *non-member* `operator+`, which adds two `bigints` and returns a `bigint`. [2✓]
- Assuming that `bigint::bigint(long)` is non-explicit, 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.)
  - Prefix `operator++` (i.e., `++x`). [2✓]
  - Postfix `operator++` (i.e., `x++`). [2✓]
- 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;
};
```
- 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 of 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.<sup>†</sup>

- (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  $\pi$ . Override `area` ( $\pi r^2$ ) 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✓]

<sup>†</sup>

Program:	<pre>using shapemap = map&lt;string,shared_ptr&lt;shape&gt;&gt;; int main() {     shapemap m { {"one", make_shared&lt;circle&gt; (location(1,4), 6)},                  {"two", make_shared&lt;square&gt; (location(8,3), 5)},                  {"zz1", make_shared&lt;circle&gt; ()} };     for (auto&amp; i: m) {         cout &lt;&lt; i.first &lt;&lt; ": (" &lt;&lt; i.second-&gt;where().first &lt;&lt; ", "               &lt;&lt; i.second-&gt;where().second &lt;&lt; ") : a= " &lt;&lt; i.second-&gt;area()               &lt;&lt; ", b= " &lt;&lt; i.second-&gt;border() &lt;&lt; endl;     } }</pre>
Output:	<pre>one: (1,4): a= 113.097, b= 37.6991 two: (8,3): a= 25, b= 20 zz1: (0,0): a= 0, b= 0</pre>



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 if 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 as 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		$\times 1 =$	$= a$
number of wrong answers		$\times \frac{1}{2} =$	$= b$
number of missing answers		$\times 0 =$	0
column total $c = \max(a - b, 0)$	12		$= c$

- 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])`
- What is the type of the constant "abc" ?  
 (A) `char*`  
 (B) `const char*`  
 (C) `const char[3]`  
 (D) `const char[4]`
- Which is a copy constructor ?  
 (A) `foo (const foo&&);`  
 (B) `foo (const foo&);`  
 (C) `foo (foo&&);`  
 (D) `foo (foo&);`
- What uses reference counting to manage memory for `foo` objects ?  
 (A) `foo*`  
 (B) `forward_list<foo>`  
 (C) `shared_ptr<foo>`  
 (D) `unique_ptr<foo>`
- Which declares a post-increment operator ?  
 (A) `T T::operator++();`  
 (B) `T T::operator++(int);`  
 (C) `T& T::operator++();`  
 (D) `T& T::operator++(int);`
- 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`

- Which library container has the best locality of reference ?  
 (A) `deque`  
 (B) `list`  
 (C) `map`  
 (D) `vector`
- 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()]`
- 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`
- Declare a destructor virtual if any \_\_\_\_ is declared virtual.  
 (A) constructor  
 (B) member function  
 (C) non-member function  
 (D) other destructor
- 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)  $O(n \log_2 n)$
- 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>

\$Id: soln-cmps109-2019q2-midterm.txt,v 1.7 2019-05-13 16:25:54-07 - - \$  
Solution to CMPS-109 2019q2-midterm, page 1

---

Question 1. [2]

```
bigint operator+ (const bigint& one, const bigint& two) {  
    bigint result = one;  
    result += two;  
    return result;  
}
```

---

Question 2(a). [2]

```
bigint& operator++ (bigint& one) { // prefix  
    return one += 1;  
}
```

---

Question 2(b). [2]

```
bigint operator++ (bigint& one, int) { // postfix  
    bigint result = one;  
    one += 1; ////////// also could be ++one, but not one++  
    return result;  
}
```

---

Question 3. [2]

```
bool bigint::operator< (const bigint& that) const {  
    if (is_negative) {  
        if (not that.is_negative) return true;  
        else return that.uvalue < uvalue;  
    }else {  
        if (that.is_negative) return false;  
        else return uvalue < that.uvalue;  
    }  
}
```

---

Question 4. [2]

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

```
    for (; begin != end; ++begin) {
        if (is_equal (*begin, item)) return true;
    }
    return false;
}
```

Solution to CMPS-109 2019q2-midterm, page 2

---

Question 5. [2]

```
template <typename iterator, class predicate>
iterator find_if (iterator begin, iterator end, predicate pred) {
    for (; begin != end; ++begin) {
        if (pred (*begin)) break;
    }
    return begin;
}
```

---

Question 6(a). [4]

```
using location = pair<double,double>;
class shape {
    private:
        location loc;
    public:
        shape (const location& xy = {0,0}): loc(xy) {}
        virtual double area() const = 0;
        virtual double border() const = 0;
        virtual const location& where() { return loc; }
        ~shape() {}
};
```

---

Question 6(b). [2]

```
class circle: public shape {
    double radius {};
    public:
        circle (const location& xy = {}, double r = 0):
            shape(xy), radius(r) {}
        double area() const override { return M_PI * radius * radius; }
        double border() const override { return 2 * M_PI * radius; }
};
```

---

Question 6(c). [2]

```

class square: public shape {
    double edge {};
public:
    square (const location& xy = {}, double e = 0):
        shape(xy), edge(e) {}
    double area() const override { return edge * edge; }
    double border() const override { return edge * 4; }
};

```

Solution to CMPS-109 2019q2-midterm, page 3

---

Question 7. [1]

```

template <typename T>
inline bool operator< (const T& one, const T& two) {
    return two < one;
}

```

---

Question 8. [2]

```

template <typename T1, typename T2>
ostream& operator<< (ostream& out, const pair<T1,T2>& pair_) {
    out << "(" << pair_.first << "," << pair_.second << ")";
    return out;
}

```

---

Question 9. [2]

```

#include <iostream>
using namespace std;
int main (int argc, char** argv) {
    for (int i = 1; i < argc; ++i) {
        if (i > 1) cout << " ";
        cout << argv[i];
    }
    cout << endl;
    return 0;
}

```

---

Question 10. [2]

```

SOURCES = foo.cpp bar.cpp
OBJECTS = ${SOURCES:.cpp=.o}
EXECBIN = foo
all : ${EXECBIN}
${EXECBIN} : ${OBJECTS}
    g++ -o $@ ${OBJECTS}

```

```
%.o : %.cpp
    g++ -c $<
```

---

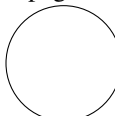
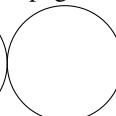
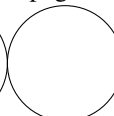
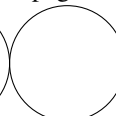

Question 11. [3]

```
template <typename itor>
bool lessrange (itor b1, itor e1, itor b2, itor e2) {
    for (; b1 != e1 and b2 != e2; ++b1, ++b2) {
        if (*b1 < *b2) return true;
        if (*b2 < *b1) return false;
    }
    return b1 == e1 and b2 != e2;
}
```

Solution to CMPS-109 2019q2-midterm, page 4

1. (D) (&argv[1], &argv[argc])
2. (D) const char[4]
3. (B) foo (const foo&);
4. (C) shared\_ptr<foo>
5. (B) T T::operator++(int);
6. (D) foo.o: foo.cpp foo.h
7. (D) vector
8. (D) v[v.size()]
9. (A) ++i
10. (B) member function
11. (C) \$ O ( n ) \$
12. (D) Prevents constructors from behaving as automatic type conversion functions.

\$Id: cmpls109-2019q1-midterm.mm,v 1.164 2019-02-08 15:12:30-08 - - \$

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.*

- Write a complete program, everything that needs to be typed into the source file `hello.cpp` in order to print the following message to the standard output. [1✓]

```
-bash-36$ ./hello
Hello, world!
```

- Assuming `iterator i, j`, write prototypes for the member operators prefix `++`, binary `!=`, and unary `*`. as they are used in the following examples : `++i`, `i!=j`, and `*i`. [3✓]

```
class iterator {
public:
```

- Assuming a header file containing the following (shown at right), show the code as it appears in the implementation file for the following two functions. Both functions update `uvalue` in place and neither calls any other function.

(a) `divide_by_2` [3✓]

```
class ubigint {
private:
    vector<uint8_t> uvalue;
    void divide_by_2();
    void multiply_by_2();
    ...
```

(b) `multiply_by_2` [3✓]

4. Rewrite the following statement using the two-semicolon version of the **for**-loop, explicitly using the iterators provided by the container. Your statement should have semantics identical to this statement. [1✓]

```
for (auto elem&: cont) foo (elem);
```

5. Write a main function that reads in words from the standard input and keeps a count of the number of times each word appears. At end of file, print out each word, in lexicographic order, one per line, followed by the number of times it appears. Use the declaration **string word;**. The loop **while (cin >> word)** will read in one word and stop at end of file. Use an appropriate container. Example output is shown. Do not show any **#include** statements, just the main function. [2✓]

```
bar 1298
foo 32
hello 9
widget 3
```

6. Write a **copy** function which takes forward iterators pointing into an input container, and a forward iterator pointing at an output container. Copy the items from the input container to the output container. It is not possible to verify whether or not the output container has enough space, so just assume that it has. [2✓]

```
template <typename in_iterator, typename out_iterator>
void copy (in_iterator in_begin, in_iterator in_end, out_iterator out_begin) {
```

7. Define a template class **stack**. All methods are to be declared inline. Do not show any of the implicitly declared methods, since, for this class, they are all acceptable for the purposes of this class.
- (a) Use a private field of class **vector** to hold the stack. [1✓]
  - (b) Function **pop** removes the top element of the stack but does not return it. [1✓]
  - (c) Function **top** returns the top element of the stack by reference, but does not alter the stack. [1✓]
  - (d) Function **push** enters a new element onto the stack. [1✓]
  - (e) Function **empty** indicates whether or not the stack is empty. [1✓]



8. Write the prototypes for class `foo` which will be implicitly generated unless otherwise specified. Show them as they would appear in a header file inside the class definition. Show prototypes only. Point allocation is given in the table at left. [2✓]

6 correct:	2	✓
5 correct:	1½	✓
4 correct:	1	✓
3 correct:	½	✓
else:	0	✓

```
class foo {
public:
```

9. Complete the following operators, assuming `operator==` and `operator<` are defined. [2✓]

```
template <class T>
inline bool operator!= (const T& x, const T& y) {
```

```
template <class T>
inline bool operator> (const T& x, const T& y) {
```

```
template <class T>
inline bool operator<= (const T& x, const T& y) {
```

```
template <class T>
inline bool operator>= (const T& x, const T& y) {
```

10. Code a linear search `find` which takes two iterators and a comparison function which returns true if its two arguments are equal. Make the usual assumptions. [2✓]

```
template <typename iterator, class comparator>
iterator find (iterator begin, iterator end, comparator equal) {
```

11. Code `find_if` which takes two iterators and a predicate and returns an iterator pointing at the first element for which the predicate is true. (A predicate is a function which returns a `bool` result.) [2✓]

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

12. Code `find_min` which returns an iterator pointing at the minimum element in a range. Its function argument `less` returns true if the first argument is less than the second argument. [2✓]

```
template <typename iterator, class less_fn>
iterator find_min (iterator begin, iterator end, less_fn less) {
```

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		$\times 1 =$	$= a$
number of wrong answers		$\times \frac{1}{2} =$	$= b$
number of missing answers		$\times 0 =$	$0$
column total $c = \max(a - b, 0)$	12		$= c$

- For a container **c** which provides random access iterators, what expression returns the same result as **c.size()** ?  
 (A) **c.begin() - c.end()**  
 (B) **c.end() - c.begin()**  
 (C) **c.end() - c.begin() + 1**  
 (D) **c.end() - c.begin() - 1**
- Which container allocates all the contents of the data structure in a single block of storage on the heap ?  
 (A) **deque**  
 (B) **list**  
 (C) **map**  
 (D) **vector**
- What will redirect both stdout and stderr to the same file ?  
 (A) **prog 1>foo.out 1>&2**  
 (B) **prog 1>foo.out 2>&1**  
 (C) **prog 2>>1 1>foo.out**  
 (D) **prog | foo.out 2>&1**
- Given an iterator pointing at an arbitrary position within a container, which container will allow an insertion at that position with a worst case time of  $O(1)$  ?  
 (A) **deque**  
 (B) **list**  
 (C) **string**  
 (D) **vector**
- What does this declaration do ?  
**class A { friend class B; };**  
 (A) Allows **B** to inherit virtual functions from **A**, but only if they are **protected**.  
 (B) Grants **A** access to the private parts of **B**, and also grants **B** access to the private parts of **A**.  
 (C) Grants **A** access to the private parts of **B**, but not vice-versa.  
 (D) Grants **B** access to the private parts of **A**, but not vice-versa.
- Which is a copy constructor of class **foo** ?  
 (A) **foo (const foo &&);**  
 (B) **foo (const foo &);**  
 (C) **foo (foo &&);**  
 (D) **foo (foo &);**
- What statement usually follows library **#includes** at the start of an implementation file ?  
 (A) **#include <std::>**  
 (B) **import std package;**  
 (C) **using namespace std;**  
 (D) **using package std;**
- Which **#include** is needed to compile the statement **cout<<3; ?**  
 (A) **<cstdio>**  
 (B) **<iomanip>**  
 (C) **<iostream>**  
 (D) **<stdio.h>**
- A destructor must be virtual if any \_\_\_\_ is virtual.  
 (A) constructor  
 (B) member data field  
 (C) member function  
 (D) nonmember function
- Which statement should appear in the following member function ?  
**foo& foo::operator= (const foo& that) {**  
 (A) **if (&this == \*that)**  
 (B) **if (\*this == that)**  
 (C) **if (this == &that)**  
 (D) **if (this == that)**
- Given an arbitrary iterator **i**, what is the most efficient way of incrementing it ?  
 (A) **++i**  
 (B) **i++**  
 (C) **i+=1**  
 (D) **i=i+1**
- The default copy constructor is almost certainly wrong if one of the members of the class is :  
 (A) a function  
 (B) a pointer  
 (C) a primitive  
 (D) an object of some class



\$Id: soln-cmps109-2019q1-midterm.txt,v 1.8 2019-02-14 17:35:09-08 - - \$  
Solution to CMPS-109 2019q1-midterm, page 1

---

Question 1. [1]

```
#include <iostream>
using namespace std;
int main (int argc, char**) {
    cout << "Hello, World!" << endl;
    return 0; ///// or return EXIT_SUCCESS if #included <cstdlib>
}
```

---

Question 2. [3]

```
class iterator {
public:
    iterator& operator++();
    bool operator!= (const iterator&) const;
    const foo& operator* () const; ///// can return foo or anything
    foo& operator* (); ///// question doesn't say iterator of what
}
```

---

Question 3(a). [3]

```
void ubigint::divide_by_2() {
    for (size_t i = 0; i < uvalue.size() - 1; ++i) {
        uvalue[i] /= 2;
        if (uvalue[i + 1] & 1) uvalue[i] += 5;
    }
    if (uvalue.size() > 0) uvalue[uvalue.size() - 1] /= 2;
}
```

---

Question 3(b). [3]

```
void ubigint::multiply_by_2() {
    size_t carry = 0;
    for (size_t i = 0; i < uvalue.size() - 1; ++i) {
        size_t sum = uvalue[i] * 2 + carry;
        uvalue[i] = sum % 10;
        carry = sum / 10;
    }
    if (carry != 0) uvalue.push_back (carry);
}
```

## Solution to CMPS-109 2019q1-midterm, page 2

---

Question 4. [1]

```
for (auto p = cont.begin(); p != cont.end(); ++p) foo (*p);
```

---

Question 5. [2]

```
int main() {
    map<string,size_t> m;
    string word;
    while (cin >> word) ++m[word];
    for (const auto& p: m) {
        cout << p.first() << " " << p.second() << endl;
    }
    return 0;
}
```

---

Question 6. [2]

```
template <typename in_itor, typename out_itor>
void copy (in_itor in_begin, in_itor in_end, out_itor out_begin) {
    while (in_begin != in_end) *out_begin++ = *in_begin++;
}
```

---

Question 7. [5]

```
template <typename item_t>
class stack {
private:
    vector<item_t> stk;
public:
    void pop() { stk.pop_back(); }
    item_t& top() { return stk.back(); }
    void push (const item_t& item) { stk.push_back (item); }
    bool empty() { return stk.empty(); }
}
```

## Solution to CMPS-109 2019q1-midterm, page 3

---

Question 8. [2]

```
foo();
foo (const foo&);
```

```
foo (foo&&);
foo& operator= (const foo&);
foo& operator= (foo&&);
~foo();
```

---

Question 9. [2]

```
template <class T>
inline bool operator!= (const T& x, const T& y) { return not (x == y);
template <class T>
inline bool operator> (const T& x, const T& y) { return y < x;
template <class T>
inline bool operator<= (const T& x, const T& y) { return not (y < x);
template <class T>
inline bool operator>= (const T& x, const T& y) { return not (x < y);
```

---

Question 10. [2]

```
// The question as stated contains an error, namely that no value
// to search for is given as an argument. An extra template
// argument and function argument must be given. Accept any
// reasonable modification to the questions, such as is presented
// here, or otherwise makes some sense.
template <typename iterator, typename wanted, class comparator>
iterator find (iterator begin, iterator end, const wanted& want,
               comparator equal) {
    while (begin != end and not equal (*begin, want)) ++begin;
    return begin;
}
```

---

Question 11. [2]

```
template <typename iterator, class predicate>
iterator find_if (iterator begin, iterator end, predicate pred) {
    while (begin != end and not pred (*begin)) ++begin;
    return begin;
}
```

---

Question 12. [2]

```
template <typename iterator, class less_fn>
iterator find_min (iterator begin, iterator end, less_fn less) {
    if (begin == end) return end;
    iterator min = begin++;
    for (; begin != end; ++begin) {
        if (less (*begin, *min)) min = begin;
    }
    return min;
}
```

## Solution to CMPS-109 2019q1-midterm, page 4

1. (A) `c.begin() - c.end()`
2. (D) `vector`
3. (B) `prog 1>foo.out 2>&1`
4. (B) `list`
5. (D) Grants B access to the private parts of A,  
but not vice-versa.
6. (B) `foo (const foo &);`
7. (C) `using namespace std;`
8. (C) `<iostream>`
9. (C) member function
10. (C) `if (this == &that)`
11. (A) `++i`
12. (B) a pointer