

# Programming Systems Design

CSCI 455  
Fall 2009

Dr. K. Narayanaswamy  
Final Examination

# SOLUTION KEY

You have 2 hours (120 minutes) to complete this examination. You can use any and all reference materials. No Internet connectivity or use of laptops/PDAs during the exam. Each student must have his/her own reference materials. No sharing of any reference materials will be permitted during the exam.

The exam contains 15 pages. Please verify that all pages are there in your test. There are 2 Parts to the test. YOU MUST ANSWER **all** questions in both parts. Part I contains mostly objective questions worth 1 point each, unless specified otherwise. Part I carries a total of 28 points. Part II contains 7 questions, together worth 72 points.

**Please be neat and organized.** Use scratch paper or the back of the answer sheets for rough work. Only final answers must appear in the space provided. Illegible answers will be marked down if they are confusing.

NAME: \_\_\_\_\_

STUDENT ID: \_\_\_\_\_

QUESTION	MAXIMUM	SCORE
<b>PART I</b>	<b>28</b>	
<b>PART II</b>		
<b>Question 1</b>	<b>12</b>	
<b>Question 2</b>	<b>9</b>	
<b>Question 3</b>	<b>14</b>	
<b>Question 4</b>	<b>10</b>	
<b>Question 5</b>	<b>9</b>	
<b>Question 6</b>	<b>9</b>	
<b>Question 7</b>	<b>9</b>	

TOTAL	(100)	
-------	-------	--

**In Section 1: each error is -1 point – no partial credit, unless indicated.**

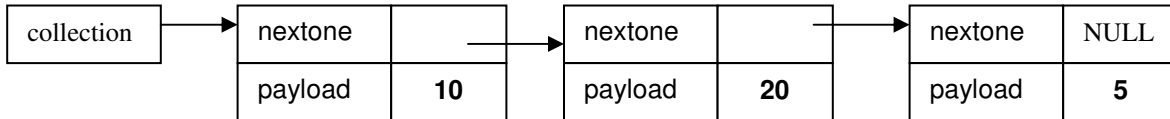
**PART I: Clearly MARK your answer. Ambiguous notations will not be given credit. Each question carries 1 point.**

- 1) The "class" construct in C++ is a more powerful form of the following:  
1) enum      2) **struct**      3) typedef      4) int
- 2) When implementing the interface functions of a component, in C++ they become:  
1) **public function members**      2) public data members  
3) private function members      4) protected member functions
- 3) Access to public functions of a class is available to:  
1) only public members of class      2) **any place the class is visible**  
3) private member functions      4) only member functions
- 4) The constructor for a class is called when:  
1) When a function is called      2) An object needs a destructor  
3) Whenever possible      4) **An object is created**
- 5) The destructor for a class is called automatically whenever:  
1) **An object goes out of scope**      2) An object comes into scope  
3) new operator is called      4) An object becomes a memory leak
- 6) Unencapsulated programs are harder to debug because:  
1) Data access is restricted      2) Only non-members can access data  
3) **Data access is unrestricted**      4) Data is encapsulated anyway
- 7) Inheritance in software development helps mainly with:  
1) Encapsulated functionality      2) **Component evolution**  
3) Programmer errors      4) Debugging and fault detection
- 8) Derived class members always have access to the following members of the base class:  
1) private data members      2) public & private members  
3) protected members only      4) **public & protected members**
- 9) The rule described in question # 8 applies :  
1) **For all derivations**      2) On Sundays  
3) Only in public derivations      4) Only in private derivations
- 10) When is the derived class constructor called relative to the base class constructor?  
1) They are called together      2) Depends on kind of derivation  
3) Derived class constructor first      4) **Base class constructor first**

- 11) From the “main” function, the members of a class can be used:
- 1) Only if they are functions
  - 2) Only if they are public
  - 3) Only if there is a derived class
  - 4) All the time
- 12) When the compiler tries to find the best match for a member function call on an object, it will select:
- 1) Any global function with that name
  - 2) Most specific member function
  - 3) Most general function possible
  - 4) Any function that looks available
- 13) Dynamic binding in C++ occurs at :
- 1) compile time
  - 2) link time
  - 3) run time
  - 4) Miller time
- 14) Dynamic binding is useful for:
- 1) functions that are overridden
  - 2) functions that are defined once
  - 3) functions that are undefined
  - 4) none of the above
- 15) Heterogeneous collections work well with dynamic binding because:
- 1) Same code works for all types
  - 2) Objects are of different types
  - 3) Code can be copied/modified
  - 4) Runtime performance is not a problem
- 16) The interface of a class refers to its:
- 1) Publicly available functions
  - 2) Private data members
  - 3) Protected members
  - 4) All members
- 17) Recursive functions are those that:
- 1) Call lower-level functions
  - 2) Use classes appropriately
  - 3) Call themselves
  - 4) Terminate in many places
- 18) A memory leak occurs in C++ when
- 1) An object goes out of scope
  - 2) A pointer is deleted
  - 3) Program loses all references to an object
  - 4) Objects have more than 1 pointer
- 19) A list container in STL is indexed and can be accessed using subscript notation:  
TRUE / FALSE
- 20) I need a data structure that is indexed and allows for efficient addition of elements at the beginning and end of the collection. The best STL container to use in this situation is:
- 1) list
  - 2) vector
  - 3) deque
  - 4) any of the above

Questions 21 – 25 use the following declaration:

```
struct unit
{
    int payload;
    unit *nextone;
};
```



21) Declare the variable “collection” appropriately below so that it can point to the first element of the linked list (1 point) and initialize it to represent an empty list (1 point).

```
unit *collection = NULL; // separate assignment of NULL to collection is also fine
```

22) Write the code to add the element 89 to the front of the existing collection as shown above, leaving the remaining list intact. (2 points)

```
unit *temp = new unit();
temp->payload = 89;
// pointers must be assigned in this order to be correct!!
temp->next = collection;
collection = temp;
```

23) Write the code to display the whole collection to the terminal, including declaration of any necessary temporary variables. (2 points)

```
unit *temp = collection;
while (temp != NULL)
{
    cout << temp->payload << endl;
    temp = temp->next;
}
```

24) If you decide to change from a linked list to an STL list collection, provide a declaration of “collection” using STL list template: **(1 point)**

```
list <int> collection;
```

25) How would you add the element 89 to the front of “collection” implemented as an STL list? **(1 points)**

```
collection.push_front(89);
```

**PART II: Question 1:** (12 points) Answer the questions related to the following code:

```
#include <iostream>
using namespace std;

int total = 0, count = 0;
class Vehicle{
public:
    void show ( ) { cout << "My value: " << id << endl; }
    int get_id ( ) { return id; }
    void set_id (int newid) { id = newid; }
    Vehicle (int initid) { id = initid; total = total + 10; }
    Vehicle ( ) { id = 0; total = total + 10; }
    ~Vehicle ( ) { total = total - 10; }
private:
    int id;
};
main ()
{
    Vehicle a(22), b(33), c(99), x(44), y(33), w[2];
    w[1].show( ); // LINE 1
    cout << "Count of vehicles: " << total << endl ; // LINE 2
    Vehicle *xp;
    {
        Vehicle x[2], y(55), w[3];
        xp = new Vehicle(77); xp = new Vehicle(88);
        cout << "Count of vehicles: " << total << endl ; // LINE 3
    }
    delete xp;
    cout << "Count of vehicles: " << total << endl ; // LINE 4
}
```

- a) Explain why it is not legal for the following code to appear in the main program (1 points)  
**int foo = xp->id;**

“id” is defined to be private in the class. So, it is not accessible in main (not a member function)

- b) What is the output produced by LINE 1 of the main program above? (2 points)

**My value: 0**

- b) What is the output produced by LINE 2 of the main program above? (3 points)

**Count of Vehicles: 70**

- c) What is the output produced by LINE 3 of the main program above? (3 points)

**Count of Vehicles: 150**

- d) What is the output produced by LINE 4 of the main program above? (3 points)

**Count of Vehicles: 80**

**PART II: Question 2:** (9 points) Answer the questions related to the following code:

```
#include <string>
#include <iostream>
using namespace std;
class Student
{ public:
    virtual string handle () { return ssn ; }
    void setSSN (string s) { ssn = s; }
    Student () { setSSN("000"); }
private:
    string ssn;
};
class Graduate : public Student
{
public:
    string handle () { return "Grad: " + Student::handle() + " + " + thesis; }
    Graduate (string t) { setSSN("111"); thesis = t ; }
private:
    string thesis;
};
main ()
{
    Student *p;
    Graduate v("Metal Properties");
    v.setSSN("444");
    cout << v.handle() << endl;           // Line # 1
    p = new Graduate("Extreme Programming");
    p->setSSN("555");
    cout << p->handle() << endl;       // Line # 2
}
```

a) What is the output of Line # 1 ? (2 points)

**Grad: 444 + Metal Properties**

b) What is the output of Line # 2 ? (2 points)

**Grad: 555 + Extreme Programming**

Supposing I remove the word "virtual" from the declaration of "handle" in the class Student. **In the changed program**, answer the following questions:

c) What is the output of Line # 1 ? (2 points)

**Grad: 444 + Metal Properties**

d) What is the output of Line # 2 ? (3 points)

**555**

### PART II: Question 3. (14 points)

Class `lottery_numbers` is designed to hold a collection of integer numbers that a person thinks are lucky to play.

```
class lottery_numbers
{
    public:
        // empty out the "good" numbers if any
        void clear_numbers ( );
        list<int> get_numbers ( ) { return numbers ; }
        // will add the provide "int" to numbers, unless it is a duplicate
        void add_number (int);
        // will remove the provided "int" from numbers, unless it is not in the numbers
        void delete_number (int);
    private:
        list<int> numbers;
};
```

a) Write the code for the member function "add\_number" below: **(3 points)**

```
void lottery_numbers::add_number (int n)
{ // add n to "numbers" if "n" is not already in the list; if it is, print an error message
```

```
    list<int>::iterator p = find(numbers.begin(), numbers.end(), n);
    if (p == numbers.end()) numbers.push_front(n); // push_back is also fine
    else cout << "Duplicate number: " << n << endl;
```

```
    // or using STL iterators as below:
```

```
    list<int>::iterator p = numbers.begin();
    while (p != numbers.end())
    {
        if (*p == n) { cout << "Duplicate number: " << n << endl; return; }
        p++;
    }
    numbers.push_front(n); // push_back is also fine
```

```
}
```

b) Write the code for the function “clear\_numbers” below: (3 points)

```
void lottery_numbers::clear_numbers ( )  
{ // empty out any of the numbers in the “numbers” list; numbers must be empty after this
```

```
    numbers.clear();
```

**OR**

```
    list<int>::iterator p = numbers.begin(), temp;
```

```
    while (p != numbers.end() )
```

```
    {
```

```
        temp = p;
```

```
        p++;
```

```
        numbers.erase(temp); // numbers.pop_front() also ok
```

```
    }
```

```
}
```

c) Consider the following declarations for the lottery numbers choices by X and Y.

Write the code to determine whether X and Y have made EXACTLY the same lottery number choices (i.e., all the numbers of X are also numbers of Y; and all the numbers of Y are also numbers of X.): (5 points)

```
// return true if X and Y have the same set of numbers; false otherwise  
bool identical_lottery_choices (lottery_numbers X , lottery_numbers Y)  
{
```

```
    list<int> x = X.get_numbers(), y = Y.get_numbers();
```

```
    if (x.size() != y.size()) return false; // this is not necessary, but it is efficient to do
```

```
    list<int>::iterator p;
```

```
    // check to make sure every number in X is in Y
```

```
    for (p = x.begin(); p != x.end(); p++)
```

```
        if ( find(y.begin(), y.end(), *p) == y.end() ) return false;
```

```
    // check to make sure every number in Y is in X
```

```
    for (p = y.begin(); p != y.end(); p++)
```

```
        if ( find(x.begin(), x.end(), *p) == x.end() ) return false;
```

```
    return true;
```

```
}
```

d) **Using STL algorithms as much as possible**, write the code for the member function "delete\_number" below: **(3 points)**

```
void lottery_numbers::delete_number (int n)
{
// remove n from "numbers" if "n" is in the list; if it is not, print an error message
```

```
list<int>::iterator p = numbers.find(n);
if (p != numbers.end()) numbers.erase(p);
```

OR

```
list<int>::iterator p = numbers.remove(n);
if (p != numbers.end()) numbers.erase(p);
```

```
}
```

## **PART II: Question 4.** (10 points)

If we have the following representation of a list using STL:

```
list<int> collection1;  
vector<int> collection2;
```

- a) Write the code to transfer ALL the data from collection2 to collection1 in the **REVERSE** order (after your code is finished, elements in collection1 must be in reverse order from the order in collection2): **(4 points)**

```
for (int i = 0; i < collection2.size(); i++) collection1.push_front(collection2[i]);  
// Solution can also ITERATORS as below  
for (vector<int>::iterator p = collection2.begin(); p != collection2.end(); p++)  
    collection1.push_front(*p);
```

- b) Write the code to replace the 4<sup>th</sup> element of collection2 (i.e., the element with index 3) with the number 50. **(1 point)**

```
collection2[3] = 50;
```

- c) Assuming “collection1” has more than 3 elements, write the code to replace the 3<sup>rd</sup> element of collection1 (the front of the list is the 1<sup>st</sup> element) with the number 50. **(2 points)**

```
list<int>::iterator p = collection1.begin( );  
p++; p++;  
*p = 50;
```

- d) Complete the body of the function below. It returns “true” if collection1 and collection2 have the same NUMBER of elements. **(3 points)**

```
bool same_size_p (list<int> & collection1, vector<int> & collection2)  
{  
    return (collection1.size()==collection2.size()) // Simplest solution  
    OR  
    if (collection2.size() == collection1.size()) return true;  
    else return false;  
    OR  
    int count = 0;  
    for (list<int>::iterator l = collection1.begin(); l != collection1.end(); l++) count++;  
    if (collection2.size() == count) return true;  
    return false;  
}
```

**PART II: Question 5.** (9 points)

```
#include <iostream>
using namespace std;

const int size = 7;
void mystery (int elements[], int high, int low)
{
    if ( low >= high) return;

    if ( elements[low] >= elements[high] )
    {
        elements[low] = 100; elements[high] = 10;
    }
    else { elements[low] = -100; elements[high] = -10; }

    mystery(elements, high - 1, low + 1);
}

main ( )
{
    int collection[size];
    for (int index = 0; index < size; index++)
        collection[index] = ( index * 2 ) % size;
    for (int index = 0; index < size; index++) cout << collection[index] << " "; // Line 1
    cout << endl;
    mystery(collection, size - 1, 0) ;
    for (int index = 0; index < size; index++) cout << collection[index] << " "; // Line 2
    cout << endl;
}
```

a) Briefly, explain the purpose of the constant “size” in the code above (2 points)

It is useful to declare a constant to represent the size, and use that constant everywhere instead of a literal number. This makes the program much easier to modify/maintain.

b) What is the output of Line 1 of the program above? (2 points)

0 2 4 6 1 3 5

c) What is the output of Line 2 of the program above? (5 points)

-100 -100 100 6 10 -10 -10

## **PART II: Question 6.** (9 points)

Read the following code segment. Then answer the questions that follow.

```
#include <list>
#include <algorithm>
#include <iostream>
using namespace std;

void enigma (int i) { cout << " " << i + i ; }

bool predicate1 (int i) { if ( (i % 4) == 2 ) return true; else return false; }

bool predicate2 (int i) { if ( (i % 3) == 2 ) return true; else return false ; }

main ()
{

    list<int> foo;
    list<int>:: iterator fp;

    for (int i = 1; i <= 3; i++) foo.push_back( i );
    for (int i = 1; i <= 2; i++) foo.push_front( i );

    for_each(foo.begin(), foo.end(), enigma);    // === LINE 1
    cout << endl;

    fp = find_if(foo.begin(), foo.end(), predicate2);

    if (fp != foo.end()) fp++;

    for_each(foo.begin(), fp, enigma);          // === LINE 2
    cout << endl;

    cout << "First Count: " << count(fp, foo.end(), 4) << endl; // === LINE 3

    cout << "Second Count: " << count_if(foo.begin(), fp, predicate1) << endl; // === LINE 4
}
```

- a) In the example shown above, supposing you wanted to change the design to work with vectors rather than lists, would the program work if you simply changed all occurrences of “list” with “vector”? Why or why not? (2 points)

Program will not work; because we are using “push\_front”, which is not valid for vector

- b) What is the output of LINE 1 of the program as shown (1 point)

4 2 2 4 6

c) What is the output of LINE 2 of the program as shown **(2 points)**

**4**

d) What is the output of LINE 3 of the program as shown **(2 points)**

**First Count: 0**

e) What is the output of LINE 4 of the program as shown **(2 points)**

**Second Count: 1**

**PART II: Question 7. (9 points)**

```
#include <vector>
#include <algorithm>
#include <iostream>
using namespace std;

class mystery
{
    public:
    mystery (int x, int y) { first = x; second = y; }
    bool operator () (int a)
    {
        if ( (a * first == second) ) return true;
        return false;
    }
    private:
        int first;
        int second;
};

void print (int x) { cout << " " << x; }

main ()
{
    vector<int> foo;

    for (int x = 1; x <= 10; x++) foo.push_back(x % 4);

    for_each(foo.begin(), foo.end(), print); //=== LINE 1
    cout << endl;
    cout << "Count 1: " << count_if(foo.begin(), foo.end(), mystery(0,0)) ; //=== LINE 2
    cout << endl;

    cout << "Count 2: " << count_if(foo.begin(), foo.end(), mystery(1, 2)); //=== LINE 3
    cout << endl;

    cout << "Count 3: " << count_if(foo.begin(), foo.end(), mystery(3,0)); //=== LINE 4
    cout << endl;
}
```

a) LINE 1 output (2 points)

1 2 3 0 1 2 3 0 1 2

b) LINE 2 output (3 points)

Count 1: 10

c) LINE 3 output (2 points)

Count 2: 3

d) LINE 4 output (2 points)

Count 3: 2