A Bunch of SYSC 2002 Exam Questions from Many Different Exams

Question 1 (11 marks)
For this question, assume that all necessary #includes are present.

a) Given the code below identify **three** mistakes **and** briefly explain each problem. [3 marks]

```
line 1:       int a, c;
line 2:       int b, *f, *g;
line 3:       *f = 2;
line 4:       f = &b;
line 5:       *f = 4;
line 6:       *g = new int;
line 7:       *f = a;
line 8:       *f = &c;
```

b) Give the output produced by the following: [2 marks]

```
int *a, b, *c, d, *e;
b = 2;
d = 1;
a = &d;
c = &d;
a = &b;
e = a;
*c = 9;
*a = 7;
cout << *e << " " << d << endl;
```

c) Give the output produced by the code in part d). [4 marks]
d) This code has a memory leak. Clearly indicate **exactly** how this should be fixed. [2 marks]

```
int *f, i, j;
IntBag *g;
f = new int;
g = new IntBag[4];
*f = 8;
for ( i=0; i<4; i++ ) {
   for ( j=0; j<i; j++ ) {
      g[i].add(j);
   }
}
cout << *f << " " << g[2].size() << endl;
(*f)--;
g = new IntBag[6];
for ( i=0; i<6; i++ ) {
   for ( j=0; j<4; j++ ) {
      (*(g+i)).add(5*i);
   }
}
cout << *f << " " << g[0].size() << endl;
```

Question 3 (7 Marks)

Below you will find an extract from a template Queue class similar to that studied in the lectures. Here the queue is implemented as a ring array.

```
template <class T>
class Queue {
private:
   T *ring;
   int capacity, count, head, tail;

public:
  Queue (int capacity); // creates empty queue (quits if capacity<1)

  Queue (const Queue &otherQueue); // copy constructor

  ~Queue (); // destructor

  void enqueue (T item); // adds item to queue, quits if full

  T dequeue (); // removes value from queue; quits if empty

  T look () const; // returns first value in queue; quits if empty

  int size() const; // returns number of elements in queue

}; // end of extract from template Queue class
```

```
Queue<String2002> q(5); // start of application program fragment
q.enqueue("a string");
q.enqueue("another string");
cout << q.look() << "\n";
cout << q.dequeue() << "\n";
q.enqueue("SYSC 2002");
cout << q.size() << "\n";
cout << q.dequeue() << "\n";
q.enqueue("yet another string");
cout << q.look() << "\n";
```

a) Give the output of the application program fragment above. [3 marks]

b) After all the statements above have been executed, show **all** the fields (member variables) along with their values. This must include a diagram of the ring array, showing all details, **including** the index of each array element. [4 marks]

Question 4 (7 Marks)

On the next page you will find an extract from a template Stack class, similar to that studied in the lectures. Here the stack is implemented as a linked list.

```
template <class T>
class Stack {
private:
  class SNode {
  public: // a very open class
```

```
    T data;
    SNode *next;
    SNode (T data, SNode *next) {
       this -> data = data; this -> next = next;
    }
  }; // end of class SNode
  SNode *head; // head pointer for the linked list
  bool elastic; // true if stack is elastic (has infinite capacity)
  int capacity; // stack capacity (only meaningful if not elastic)
  int count; // number of values on stack

public:
  Stack (); // creates empty elastic stack

  Stack (int capacity); // creates empty non-elastic stack with given
                        // capacity (quits if capacity < 1)

  Stack (const Stack &otherStack); // copy constructor

  ~Stack (); // destructor

  void push (T value); // pushes value onto stack; quits if full

  T pop (); // pops value from stack; quits if empty

  T peek () const; // returns top value of stack; quits if empty

  int size() const; // returns number of elements in stack

}; // end of extract from template Stack class
```

```
Stack<Date> s(5); // start of application program fragment
Date d1, d2, d3(1,1,2010), d4(11,12,2009);
s.push(Date(12,12,2012));
s.push(d1);
s.pop().write(cout);
cout << endl;
s.peek().write(cout);
cout << endl;
s.push(d4);
s.push(d2);
s.push(d3);
cout << s.size() << " ";
s.peek().write(cout);
cout << " ";
s.pop().write(cout);
cout << endl;
```

a) Give the output of the application program fragment above. [3 marks]

b) After all the statements above have been executed, show **all** the fields (member variables) along with their values. This must include a diagram of the linked list, showing **all** pointers, including any NULL pointers, if applicable. For any dates, just write the day, month, and year (you do **not** need to show the dayNumber). [4 marks]

Question 5 (5 marks)

Below is a header file for a variation of the IntBag class implemented as a dynamically allocated array.

```
class IntBag {

private:
  int *data; // pointer to dynamically allocated array
  int capacity; // the size of this array
  int count;  // number of values currently in the bag

public:
  // Constructs an empty bag having the default capacity.
  IntBag ();

  // Constructs an empty bag having the specified capacity.
  // Quits if the capacity is unreasonable.
  IntBag (int capacity);

  // Copy constructor.
  IntBag (const IntBag &otherBag); // copy constructor

  // Destructor
  ~IntBag (); // destructor

  // Adds a value to the bag.  Doubles the capacity if bag full.
  void add (int value);

  // Removes one occurrence of the specified value from the bag.
  // Returns true on success (value existed) and false otherwise.
  bool remove (int value);

  // Returns the number of times that the value occurs in the bag.
  int countOccurences (int value) const;

  int size () const { return count; } // number of values in the bag

  int getCapacity () const { return capacity; } // capacity of bag

  // Overloaded assignment operator.  Quits if the destination is
  // not large enough to contain all the values in the source bag.
    IntBag& operator= (const IntBag &otherBag);
};
```

Write the implementation of the add method. Note that the capacity is **doubled** if the bag is full.

Question 6 (10 marks)

Below you will find the header file (List.h) for a variation of the List class of integers represented by a linked list.

```
class List {
private:

  class Node {
  public:
    int data;
    Node *next;
    Node (int data, Node *next) {
      Node::data = data; Node::next = next;
    }
  };

  Node *head;

public:
  // default constructor: creates empty list
  List ();

  // copy constructor
  List (const List &otherList);

  // destructor: destroys list
  ~List ();

  // outputs list contents to cout
  void outputList () const;

  // adds a value at the beginning of the list
  void insert (int value);

  // assigns one linked list to another
  List& operator= (const List &otherList);
};
```

In this question you are going to define and implement a new method, `addFiveToEveryValueOverFifty`. This method adds 5 to all values in the original list that are greater than 50 (i.e. overwrites every value in the list that is larger than fifty with the original value plus five). This method has no arguments and returns nothing.

a) Write an **iterative** implementation of `addFiveToEveryValueOverFifty`. [5 marks]

b) Write a **recursive** implementation of `addFiveToEveryValueOverFifty`. [5 marks]

Question 1 (8 marks)
Suppose that somebody owes X dollars. At the end of every year the debt is recalculated as explained here:
1. First the debt is increased by X * (R / 100), where R is the interest rate.
2. Secondly, the debt is then reduced by the annual payment of Y dollars.

Write a RECURSIVE function that, given X, R, and Y, returns the number of years it will take to pay off the debt. You may assume that the values are such that the debt will eventually get paid off.

Hint: How much debt remains after one year? How many years will it take to pay off this debt?

Question 4 (8 marks)
Here is the class definition for an elastic Queue implemented using a ring array:

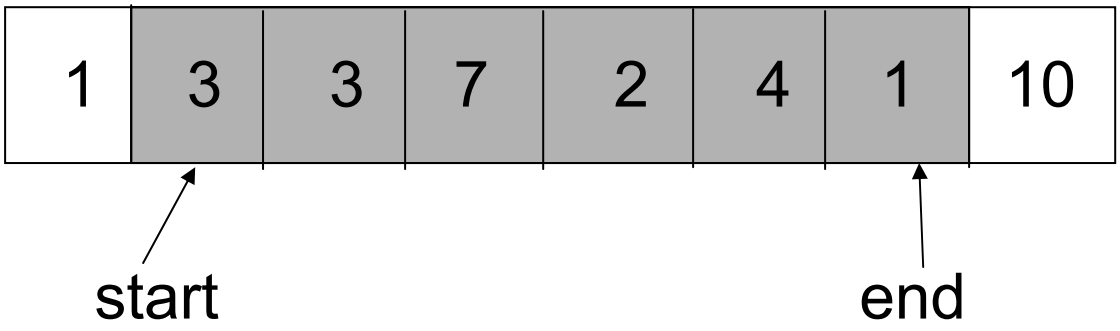
```
template <class T>
class Queue {
   private:
     T *ring; // dynamically allocated ring array
     int capacity, count, head, tail;
   public:
     Queue();
     void enqueue (T item); // adds the item to the queue, enlarging it if
                            // necessary
     T dequeue (); // throws "overflow_error" if queue empty
     int size (); // returns number of items in queue
     T look (); // as for dequeue but item is left in queue
};

Here is a partial implementation of enqueue:
template <class T>
void Queue<T>::enqueue (T item) {
   int i;
   T *temp;
   if (count == capacity) { // increase the capacity of the ring array by 10
      // this code has been omitted
   }
   ring[tail] = item; count++;
   tail = (tail + 1) % capacity;
}
```

The omitted code increases the capacity of the queue by 10. Basically it has to enlarge the array, somehow preserve the original data, and leave "head" and "tail" containing appropriate values. You are to write the omitted code.

Hint: We only enlarge the ring array if it is full. What is the relationship between head and tail if the queue is full? Should this still be the case after the array is enlarged?

Question 2 (6 marks)
In the space provided in your answer booklet, write a **recursive** function that finds and returns the largest value in some portion of an **unsorted** array. The area of interest starts at element "start" and runs through element through "end" (as illustrated below).



A function header has been supplied:
```
int findLargest (int array[], int start, int end)
```

You should work on the assumption that "start" will never be greater than "end".

Question 4 (6 marks)
This question assumes the stack and queue classes outlined below. Note that these class definitions are not complete. Instead you have only been given what you need to answer this question.

```
template <class T>
class Stack {
   public:
      Stack(); // creates an elastic stack
      push (const T &data);
      T pop (); // quits if the stack is empty
      int size (); // returns the number of values on the stack
      bool isEmpty(); // returns true if the stack is empty
};

template <class T>
class Queue {
   public:
      Queue(); // creates an elastic queue
      enqueue (const T &data);
      T dequeue (); // quits if the queue is empty
      int size (); // returns the number of values in the queue
      bool isEmpty(); // returns true if the queue is empty
};
```

Write a function that, given a queue of integer values, reverses the order of the queue's contents. Your function must conform to the sample call:.

```
Queue<int> queue;
// assume that some values get enqueued here . . .
reverseQueue (queue); // reverse the order of the values on the queue
// the value that was at the head of the queue is now at the end of the
// queue, and so on
```

Question 1 (3 Marks)
Catalan numbers are defined for non-negative integers as follows:
Catalan(0) = 1
Catalan(n) = Catalan(n-1)*(4n+2)/(n+1), for n>0

Write a **recursive** function that calculates Catalan(n). Make sure that your function does something reasonable for **all** integer arguments.

Question 2 (4 Marks)
Write a function, getMean, which given the head pointer to a linked list of integers and the Node class defined below, returns the **mean** of the values stored in the list. Note that the mean (sometimes called the average) is defined to be the sum of the integers divided by the number of integers in the list. If the list is empty, your function should quit. Your function may be iterative **or** recursive. If you choose to call any other functions, you must also write their implementations.

```
class Node {
   public:
      int value;
      Node *next;
}
```

Question 3 (2 Marks)
Assuming that we start with an empty stack of integers, s, give the output of the following code fragment:

```
s.push(4);
s.push(5);
cout << s.pop() << endl;
s.push(12);
s.push(3);
s.pop();
cout << s.pop() << endl;
```

Question 6 (14 Marks)
In this question you are implementing a class that models customers lining up and paying for their purchases at a grocery store. A grocery store has a number of lines, called "checkouts". Each one of these is a queue (FIFO). When a customer is ready to pay, he or she joins the checkout with the shortest queue.

For example, in this grocery store we have four checkouts:

- checkout[0] has 3 customers: fred, mary, joe
- checkout[1] is empty (no customers)
- checkout[2] has 2 customers: sue, john
- checkout[3] has 1 customer: louise

The next customer would join the queue for checkout[1] (as it is the shortest).

Note that we do not know (or care) how the queues are implemented (e.g. they could be linked lists or arrays), all we know is that we enqueue at the right, and dequeue at the left.

Here's a partial header file for our Grocery Store:

```
class GroceryStore {
  private:
    // a dynamically allocated array of checkout queues (one queue per
    // checkout line). Each queue is made up of customer names,
    // represented by strings.
    Queue<string> *checkout;
    int numCheckouts; // the number of checkouts

    // returns the array index of the checkout with the shortest queue.
    // (The shortest queue is the one with the fewest customers.)
    // If there is a tie for shortest queue, then, of the tied
    // checkouts, it returns the checkout with the lowest index.
    // For example, if checkout[1] and checkout[2] were the only two
    // empty queues, this method would return 1.
    int findShortestQueue();

  public:
     // creates a grocery store with "number" checkouts. Quits
     // if "number" is less than 1.
     GroceryStore(int number);

     // adds the given customer to the shortest available queue.
     void lineUp(const string &name);

     // serves the customer at the front of the given queue (i.e.
     // this customer buys his groceries and is removed from the queue).
     // Prints a message giving queue number and the customer name.
     // Quits if this queue does not exist or is empty.
     void serve(int queue);

};
```

Here's everything that you need to know about the Queue template class:

```
template <class T>
class Queue {
  public:
    Queue (); // creates queue with infinite capacity
    void enqueue (T item); // adds item to the end of the queue
    T dequeue (); // removes and returns item from the front of
                  // the queue. Quits if queue is empty.
    int size (); // returns number of items in queue
    T look (); // as for dequeue but item is left in queue
```

```
};
```

a) Write the GroceryStore constructor exactly as it would appear in GroceryStore.cpp. (3 marks)

b) Write the findShortestQueue private member method exactly as it would appear in GroceryStore.cpp. (5 marks)

c) Write the lineUp method exactly as it would appear in GroceryStore.cpp. Hint: This method should use findShortestQueue. (3 marks)

d) Give the output of this application program code fragment: (3 marks)

```
GroceryStore gs(3);
gs.lineUp("dan");
gs.lineUp("siva");
gs.lineUp("joe");
gs.serve(1);
gs.lineUp("lynn");
gs.lineUp("fran");
gs.serve(2);
gs.serve(0);
gs.serve(1);
```