These are some review questions to test your understanding of the material. Some of these questions may appear on an exam.

1 Stacks and Queues

Please see the definition of `Stack` on page 3 and of `Queue` on page 4. These are the definitions from the text with minor modifications noted.

1.1 Suppose that `q` is an object of the `Queue` class and was constructed as

```
Queue<char> q; // Queue of size 5, initially empty
```

For the following problems, assume that `q` is initially empty. Show the contents of `element` and the values of data members `front` and `back` initially and after each statement has executed. Indicate any errors that occur.

1.
```
              initial:     _ _ _ _ _    front=     back=

q.enqueue('A');            _ _ _ _ _    front=     back=

q.enqueue('B');            _ _ _ _ _    front=     back=

q.enqueue('C');            _ _ _ _ _    front=     back=

char ch = q.Dequeue();     _ _ _ _ _    front=     back=

q.enqueue(ch);             _ _ _ _ _    front=     back=
```
2.
```
              initial:     _ _ _ _ _    front=     back=

q.enqueue('X');            _ _ _ _ _    front=     back=

q.enqueue('Y');            _ _ _ _ _    front=     back=

q.enqueue('Z');            _ _ _ _ _    front=     back=
while (!q.Empty())
  {
    char ch = q.Dequeue();  _ _ _ _ _    front=     back=  // each time
  }
```

```
3.            initial:   _ _ _ _ _    front=    back=

   char ch = 'q';
   for (int i = 1; i <= 3; ++i)       // show result for each iteration
     {
       q.enqueue(ch);      _ _ _ _ _    front=    back=

       ch++;               _ _ _ _ _    front=    back=

       q.enqueue(ch);      _ _ _ _ _    front=    back=

       q.Dequeue();        _ _ _ _ _    front=    back=
     }
```

1.2 Use the given operations for `Stack` for this problem. Write a `C++` function

```
template <class Object>
Stack<Object> copyStack(Stack<Object> & stk)
```

that returns a `Stack` containing the elements of `stk`, in the same order as in `stk`.

1.3 Use the given operations for `Queue` and `Stack` for this problem.
Write a `C++` program that determines if a given string is a palindrome (*i.e.* reads the same forward and backward). Your program should print (to `cout`) "palindrome" if the string is a palindrome and "not-palindrome" if it is not. You may assume that the string to be tested is initially stored in a `null`-terminated array of `char`. You are to `push` each character onto a stack and `enqueue` it onto a queue. The test is to use only operations on the queue and the stack.

1.4 Describe (pseudo-code is fine) how the operations `isEmpty`, `pop`, and `push` are implemented in the text's array implementation of `Stack` (shown on page 3).

1.5 Describe (pseudo-code is fine) how the operations `isEmpty`, `pop`, and `push` are implemented in the `List`-based implementation of `Stack` given in the lecture notes.

1.6 Describe the advantages and disadvantages of the text's array and the lecture note's list implementations of the *stack* ADT. Consider the asymptotic behavior for each of the operations `isEmpty`, `pop`, and `push`. Also consider the storage requirements for each implementation.

Definition of `Stack` Class

Note: this is directly out of the text. The meanings of the operations are the same as in the text.

```
template <class Object>
class Stack
{
  public:
    explicit Stack( int capacity = 10 );

    bool isEmpty( ) const;
    bool isFull( ) const;
    const Object & top( ) const;

    void makeEmpty( );
    void pop( );
    void push( const Object & x );
    Object topAndPop( );

  private:
    vector<Object> theArray;
    int            topOfStack;
};
```

Definition of Queue Class

Note: this differs from the text only in that the initial size is 5, not 10. The meanings of the operations are the same as in the text.

```
template <class Object>
class Queue
{
  public:
    explicit Queue( int capacity = 5 );

    bool isEmpty( ) const;
    bool isFull( ) const;
    const Object & getFront( ) const;

    void makeEmpty( );
    Object dequeue( );
    void enqueue( const Object & x );

  private:
    vector<Object> theArray;
    int            currentSize;
    int            front;
    int            back;

    void increment( int & x );
};
```