Name:

Student Number:

Please read the following guidelines carefully!

• Please write your name on the front and back of the exam. The latter is to help us return the exams.

• This examination has 4 questions. There are a total of 9 pages, DOUBLE-SIDED.

• Any question you leave blank or clearly cross out your work and write “I don’t know” is worth 10% of the marks.

Take a deep breath.

This is your chance to show us

How much you’ve learned.

We WANT to give you the credit

That you’ve earned.

A number does not define you.

Good luck!
1. The following questions test your understanding of the terminology and concepts from the first four weeks of this course. You may answer in either point form or full sentences; **you do not need to write much to get full marks**! It’s strongly recommended that you use examples to illustrate your points.

(a) [2] Explain the difference between a **stack** and a **queue**.

(b) [2] Consider the built-in Python list (**NOT** the linked list). Why is it much faster to insert an element at the end of the list than at the beginning?

(c) [2] Recall from lecture the iterative, node-based implementation of linked lists (which we process using loops rather than recursion). Is it faster to insert an element at the beginning or end of the list? Why?
(d) [2] The following classes are defined using **inheritance**.

```python
class A:
    def __init__(self, x):
        self.x = x

def show(self):
    print(self.x)

class B(A):
    def show(self):
        print("I’m a B!")

def noshow(self):
    print("shhh")
```

Assume that we’ve loaded this source code into the interpreter, and run the following commands successfully.

```python
>>> a = A("Hi")
>>> b = B("Bye")
```

**Clearly explain** what happens when each of the following commands is run next.

```python
>>> a.show()
```

```python
>>> b.show()
```

```python
>>> a.noshow()
```

```python
>>> b.noshow()
```
2. Write a function that takes two stacks, and returns a new stack with the items of the first stack, and then above them the items of the second stack, in their original order.

That is, the top of the new stack is the top of the second stack, and the bottom of the new stack is the bottom of the first stack.

You may only use the Stack ADT methods in this question; no other data structures (like Python lists) are allowed.

Finally, this is a non-mutating function: when the function ends, it should return a new stack, but the original two stacks should be unchanged!

```python
def combine(stack1, stack2):
    """ (Stack, Stack) -> Stack
    """
    # YOUR CODE GOES HERE
```
3. Consider the following incorrect method for inserting a new item at the second position in a linked list.

```python
def insert_second(self, item):
    """Insert item at the second position of this list.""
    >>> lst = LinkedList([1, 2, 3])  # [1 -> 2 -> 3]
    >>> lst.insert_second(10)  # [1 -> 10 -> 2 -> 3]
    if self.first is None:
        raise IndexError
    else:
        new_node = Node(item)
        self.first.next = new_node
```

Explain what goes wrong when you try to use this method on a linked list [1 -> 2 -> 3] to insert 10 at the second position.

Then, rewrite the else block to fix the problem.

```python
def insert_second(self, item):
    """Insert item at the second position of this list.""
    >>> lst = LinkedList([1, 2, 3])  # [1 -> 2 -> 3]
    >>> lst.insert_second(10)  # [1 -> 10 -> 2 -> 3]
    if self.first is None:
        raise Exception
    else:
        # YOUR CODE GOES HERE
```
4. [5] A very common operation to do on lists is filter them according to some property, e.g. “the videos about recursion” or “the students who are in first-year”.

Python has a built-in filter function that does this for regular lists; your task in this question is to implement a simpler version for the node-based linked list.

**Your function should create new linked list – the original linked list should remain unchanged!**

You **may not** use any LinkedList methods we developed in class, other than the constructor. You may, however, use a built-in Python list if you’d like.

```python
def filter_positive(lst):
    """ (LinkedList of int) -> LinkedList of int
    Return a new LinkedList whose items are the ones in lst that have value > 0. The items must appear in the same order they did in lst.
    >>> lst = LinkedList([3, -10, 4, 0]) # [3 -> -10 -> 4 -> 0]
    >>> pos = lst.filter_positive() # [3 -> 4]
    """

    # YOUR CODE GOES HERE
```
**Bonus Question [2]**

**Warning**: this is a difficult question, and will be marked harshly. Only attempt it if you have finished all of the other questions!

One of the major shortcomings of our linked list class is that it’s only possible to move forwards at a node, but not backwards. A **doubly-linked list** is a linked list of nodes where the nodes store a reference to both the next node and previous node in the list. Storing the “previous” links enables the extra flexibility of moving backwards and forwards through a list, at the cost of a bit of extra memory.

Write the analogous classes and constructors for the `DoubleNode` and `DoublyLinkedList` class to the standard code found on page 5. Note that the parameters to the constructors (an object and a list) must stay the same.

The core of this question is converting a Python list into a doubly-linked list.
Use this page for rough work.
<table>
<thead>
<tr>
<th>Grade</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>Total</th>
<th>Bonus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out Of</td>
<td>8</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>22</td>
<td>2</td>
</tr>
</tbody>
</table>