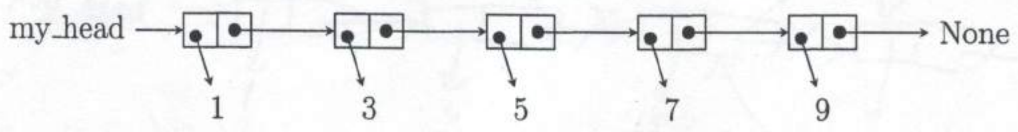
```
def mystery(head, n):
    """





    """

    curr = head
    prev = None
    found_node = None

    #
    while curr != None and curr.value != n:
        prev = curr
        curr = curr.next_node

    #
    if curr != None:
        found_node = curr

        #
        while curr.next_node != None:
            curr = curr.next_node

        #
        if found_node != curr:
            #

            #
            if prev == None:
                head = head.next_node
            else:
                prev.next_node = found_node.next_node

            #
            curr.next_node = found_node
            found_node.next_node = None

    return head
```

Nick wrote a really cool function that did something to a linked list of integers (Represented by the standard IntNode class used in lecture). But then, the *Code Mangler* struck!!! All of Nick's docstrings and internal comments were stolen and the name of the function was changed. What's left of Nick's code is on the previous page, we have also included a copy at the back of the test, along with the IntNode class, which you may detach for rough work.

Question 1. [20 MARKS]

Part (a) [3 MARKS]

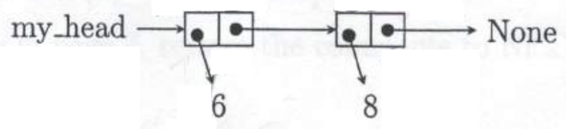
Consider the following diagram illustrating a state of memory:



In the space below, draw the memory diagram after running the following line of code.
`my_head = mystery(my_head, 24)`

Part (b) [3 MARKS]

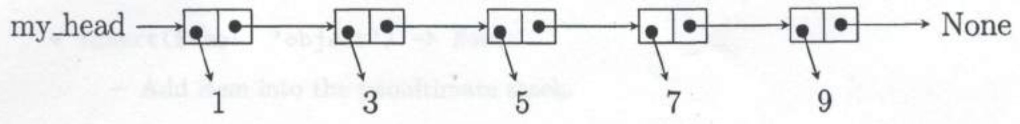
Consider the following diagram illustrating a state of memory:



In the space below, draw the memory diagram after running the following line of code.
`my_head = mystery(my_head, 6)`

Part (c) [3 MARKS]

Consider the following diagram illustrating a state of memory:



In the space below, draw the memory diagram after running the following line of code.
`my_head = mystery(my_head, 5)`

Part (d) [1 MARK]

Give a better name for Nick's function:

Part (e) [10 MARKS]

On page 2, restore the comments to Nick's code.

Question 2. [15 MARKS]

A `penultimate` stack is a container of objects with the following operations:

- `insert(item: 'object') -> None:`
  - Add item into the penultimate stack.
- `extract() -> item: 'object':`
  - If the penultimate stack has two or more items in it, then remove and return the second last item that was added.
  - If the penultimate statck has exactly one item in it, then remove and return it.
  - REQ: The penultimate stack must be nonempty.
- `is_empty() -> bool:`
  - Return True iff the penultimate stack is empty.

Now recall the BANANA game from your homework, where the object is to transform a source word into a goal word using only the container provided. Remember that we have three options at each stage.

`Move`: Add the next letter of source word onto the end of goal word.
`Put`: Put the next letter from source word into the container.
`Get`: Remove a letter from the container and add it to the end of goal word.

Part (a) [5 MARKS]

For each of the following permutations of BANANA, is it possible to create it using a penultimate stack (starting with BANANA as source word)? If Yes, give the series of moves which will achieve this.

ANANAB

NNBAAA

ANANBA

```
# A stack class
# Uses a list for the stack.
# Saved in a file called liststack.py

class Stack():
    def __init__(self: 'Stack') -> None:
        self._contents = []

    def push(self: 'Stack', item: 'object') -> None:
        self._contents.append(item)

    def pop(self: 'Stack') -> object:
        return self._contents.pop()

    def is_empty(self: 'Stack') -> bool:
        return (len(self._contents) == 0)
```

Part (b) [10 MARKS]

Use the stack implementation on the previous page to complete the following implementation of a PenultimateStack class.

```
from liststack import Stack

class PenultimateStack():
    def __init__(self):



    def insert(self, item):



    def extract(self):



    def is_empty(self):
```

Question 3. [15 MARKS]

In the space below, draw a UML diagram that meets the following requirements:

- We need to represent books and authors. Authors should have a first and last name and a mailing address. Sometimes books change their titles and number of pages, and we often need to add just one extra page, so it would be good to have a simple way of doing that. Given a book, I should be able to get all of its authors, and given an author I want to be able to get all the books they've written.
- Books can be paperback or hardcover, and hardcover books can either be regular or special edition. Paperbacks cost less, and a special edition's price depends on whether or not it has a certificate with it. The number of copies we print will also depend on the type of book, there's a formula for paperback based just on the number of pages, but for hardcover it depends on the author, and for special editions, we always print a fixed number.
- Authors are either contractors (who have an agent and are paid solely on the number of pages they've written), or salaried (who have an annual salary based on the number of years they've worked with the company). We need to be able to see for any given author how much they've been paid over the years