THE UNIVERSITY OF AUCKLAND

SECOND SEMESTER, 2014
Campus: City

COMPUTER SCIENCE

TEST

Principles of Computer Science

(Time Allowed: 75 minutes)

Note:

- The use of calculators is NOT permitted.
- Compare the term test version number on the Teleform sheet supplied with the version number above. If they do not match, ask the supervisor for a new sheet.
- Enter your name and student ID on the Teleform sheet. Your name should be entered left aligned. If your name is longer than the number of boxes provided, truncate it.
- Answer all **Multiple-choice** questions on the Teleform answer sheet provided. Answer Section **B** in the space provided in this booklet. Attempt all questions.
- Use a dark pencil to mark your answers in the multiple choice answer boxes on the Teleform sheet. Check that the question number on the sheet corresponds to the question number in this question/answer book. If you spoil your sheet, ask the supervisor for a replacement.
- Write your answers in the space provided in the short answer section. Write as clearly as possible. The space provided will generally be sufficient but is not necessarily an indication of the expected length. Extra space is provided at the end of this exam book.

Surname:	
First Name(s):	
Student ID:	
Login Name(UPI):	
Lab Time:	

MARKERS ONLY

Question		Mark	Out Of
1 – 18	Multiple Choice		20
19 – 25	Written		30
	TOTAL		50

Question 1
[1 mark] Which of the following functions will **NOT** test to see if a list is a palindrome?

(a)
```
def is_palindrome(the_list):
    for x in the_list:
        for y in the_list[::-1]:
            if x != y:
                return False
    return True
```
(b)
```
def is_palindrome(the_list):
    return the_list == the_list[::-1]
```
(c)
```
def is_palindrome(the_list):
    copy = the_list.copy()
    copy.reverse()
    return the_list == copy
```
(d)
```
def is_palindrome(the_list):
    size = len(the_list)
    for x in range(size):
        if the_list[x] != the_list[size-x-1]:
            return False
    return True
```
(e)
```
def is_palindrome(the_list):
    backwards = []
    for x in the_list:
        backwards = [x] + backwards
    return backwards == the_list
```

The `Right_Angle_Triangle` class below is used by the following two questions.

```
import math

class Right_Angle_Triangle:
    """Represents a right angle triangle.  The length of the two shortest
       sides are required.
    """

    def __init__(self, a, b):
        #Store the length of the two shortest sides
        self.a = a
        self.b = b

    def perimeter(self):
        #Calculate the sum of the three edges
        hypotenuse = math.sqrt(self.a ** 2 + self.b ** 2)
        return self.a + self.b + hypotenuse

    def area(self):
        #Area of a triangle is half height times the base
        return self.a / 2 * self.b

    def scale(self, factor):
        #Alters the size of the triangle
        self.a = self.a * factor
        self.b = self.b * factor
    def __gt__(self, other):
        #Uses the area to compare size
        return self.area() > other.area()

    def __lt__(self, other):
        #Uses the area to compare size
        return self.area() < other.area()

    def __eq__(self, other):
        #Compares the edges of the two triangles
        return self.a == other.a and self.b == other.b
```

Question 2
[1 mark] Which of the following code fragments will cause an error?

(a)
```
s = Right_Angle_Triangle(3, 4)
print(s.scale(4))
```

(b)
```
s = Right_Angle_Triangle(3, 4)
bigger = s > s
print(bigger)
```

(c)
```
s = Right_Angle_Triangle(3, 4)
print(s.perimeter())
```

(d)
```
s = Right_Angle_Triangle(3, 4)
print(s.area())
```

(e)
```
s = Right_Angle_Triangle(3, 4)
t = s.scale(4)
print(t.area())
```

Question 3
[1 mark] What is the output of the following code?

```
s = Right_Angle_Triangle(3, 4)
t = Right_Angle_Triangle(4, 3)
print(s > t, s < t, s == t)
```

(a) `True, True, False`

(b) `False, True, False`

(c) `True, False, True`

(d) **`False, False, False`**

(e) `False, False, True`

Question 4
[1 mark] What is the output of the following code?

```
a = [1, 2, 3]
b = [1, 2, 3]
c = a
print(a is b, a == b, a is c, a == c)
```

(a) `True False False False`

(b) `True True False`

(c) **`False True True True`**

(d) `False True False True`

(e) `True False True False`

Question 5
[1 mark] What is the output of the following code?

```
s = "Hello"
t = "World"
print("{1}-{0}-{2}-{0}".format(s, t, s, t))
```

(a) `Hello-World-Hello-World`

(b) `HelloWorld--HelloWorldHelloWorld-`

(c) `World-Hello-WorldWorld-Hello`

(d) `Hello-World-HelloHello-World`

(e) `World-Hello-Hello-Hello`

The `check_exceptions` function below is used by the following two questions:

```
def check_exceptions(n):
    result = 3 / n #possible exception
    try:
        result = 'Output: ' + result #possible exception
        print(result)
    except:
        print('Exception')
    finally:
        print('Finally')
    print('Exit function')
```

Question 6
[1 mark] What is the output from the following code?

```
try:
    check_exceptions(0)
except:
    print('Done')
```

(a)
```
Finally
Done
```

(b) `Done`

(c)
```
Output: 3
Finally
Exit function
```

(d)
```
Finally
Exit function
Done
```

(e)
```
Exception
Finally
Exit function
```

Question 7

[1 mark] What is the output from the following code?

```
try:
    check_exceptions(3)
except:
    print('Done')
```

(a)
```
Output: 1
Finally
Exit function
```
(b)
```
Exeption
Done
```
(c)
```
Exception
Finally
Exit function
```
(d)
```
Output: 1
Exit function
```
(e)
```
Done
```

Question 8

[1 mark] What is the output of the following code?

```
class simple_maze:
    def __init__(self):
        self.maze = {}

    def add_connection(self, src, dst):
        if src in self.maze:
            self.maze[src].append(dst)
        else:
            self.maze[src] = [dst]
        if dst in self.maze:
            self.maze[dst].append(src)
        else:
            self.maze[dst] = [src]

m = simple_maze()
m.add_connection(0, 2)
m.add_connection(1, 2)
print(m.maze)
```

(a) `{0: [2], 1: [2]}`

(b) `{0: [2], 1: [2], 2: [0], 2:[1]}`

(c) `{0: [1, 2], 1: [0, 2], 2: [0, 1]}`

(d) `{0: [2], 1: [2], 2: [0, 1]}`

(e) `{}`

The website class below is used by the following two questions.

```
class website:
    def __init__(self, d):
        try:
            if len(d) < 1:
                raise ValueError()
            else:
                self.links = d
        except:
            raise ValueError('The website must contain at least one page.')

    def add_link(self, src, dst):
        if src not in self.links:
            raise ValueError("That source page doesn't exist.")
        if dst not in self.links:
            raise ValueError("That destination page doesn't exist.")
        if dst in self.links[src]:
            raise ValueError("That link already exists.")
        self.links[src] += dst

    def add_page(self, label):
        if label in self.links:
            raise ValueError("That page already exists.")
        self.links[label] = []
```

Question 9

[1 mark] What is the output of the following code?

```
before = {
    'A' : ['B'],
    'B' : ['C'],
    'C' : []
    }
w = website(before)
try:
    w.add_link('D', 'C')
except Exception as e:
    print(e)
```

(a) `No output is printed.`

(b) **`That source page doesn't exist.`**

(c) `That page already exists.`

(d) `That link already exists.`

(e) `That destination page doesn't exist.`

Question 10
[1 mark] What is the output of the following code?

```
before = {
    'A' : ['B'],
    'B' : ['C'],
    'C' : []
    }
w = website(before)
try:
    w.add_link('B', 'C')
except Exception as e:
    print(e)
```

(a) `No output is printed.`
(b) `That destination page doesn't exist.`
(c) `That source page doesn't exist.`
(d) `That link already exists.`
(e) `That page already exists.`

Question 11
[1 mark] What is the big-O complexity of the following function?

```
def complexity_01(my_list):
    n = len(my_list)
    i = 5
    while i < n:
        print(my_list[i])
        i += 1
```

(a) O(log n)
(b) O(n)
(c) $O(n^2)$
(d) O(1)
(e) None of the above

Question 12
[1.5 marks] What is the big-O complexity of the following function?

```
def complexity_02(my_list1):
    my_list2 = []
    n = len(my_list1)
    for i in range(n):
        my_list2.insert(0, my_list1[i])
    return my_list2
```

(a) O(log n)
(b) O(n log n)
(c) O(n)
(d) $O(n^2)$
(e) None of the above

Question 13

[1 mark] What is the big-O complexity of the following function?

```
def complexity_03(my_list1):
    my_list2 = [2, 4, 6, 8, 10]
    count = 0
    number = 0
    for i in my_list1:
        if i == my_list2[number] and number < len(my_list2):
            count += 1
            number += 1
    return count
```

(a) **O(n)**

(b) O(n^2)

(c) O(n log n)

(d) O(log n)

(e) None of the above

Question 14

[1.5 marks] What is the big-O complexity of the following function?

```
def complexity_04(my_list1):
    my_list2 = []
    n = len(my_list1)
    for i in range(1, n, 2):
        my_list2.append(my_list1[i])
    return my_list2
```

(a) O(n^2)

(b) O(log n)

(c) O(n log n)

(d) **O(n)**

(e) None of the above

Question 15

[1 mark] Using the algorithm studied in class to check for balanced braces, which of the following is **TRUE** if the string has balanced braces and the end of the string has been reached?

(a) the stack contains one "{"

(b) **the stack is empty**

(c) the stack contains one "{" and one "}"

(d) the stack contains one "}"

(e) None of the above

CONTINUED

Question 16
[1.5 marks] Which of the following is **FALSE** about converting **infix** expressions to **postfix** expressions?

(a) the operator always moves "to the right" with respect to the operands
(b) all parentheses are removed
(c) **the operators always stay in the same order with respect to one another**
(d) the **operands** always stay in the same order with respect to one another
(e) None of the above

Question 17
[1.5 marks] Which of the following is the postfix form of the infix expression: `4 * 2 + 5`?

(a) 4 2 5 * +
(b) **4 2 * 5 +**
(c) 5 2 + 4 *
(d) 2 4 5 * +
(e) None of the above

Question 18
[1 mark] Complete the following statement so that it is true:
"The `pop` operation throws an `IndexError:` when it tries to ...".

(a) check if an already empty stack is empty
(b) add an item to an empty stack
(c) **delete an item from an empty stack**
(d) check the size of an empty stack
(e) None of the above

SECTION B

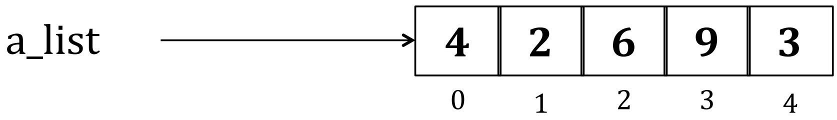
Answer all questions in this section in the space provided. If you run out of space then please use the Overflow Sheet and indicate in the allotted space that you have used the Overflow Sheet.

Question 19: [5 marks]

As shown in lecture 02, the structure of the list created by the following code:

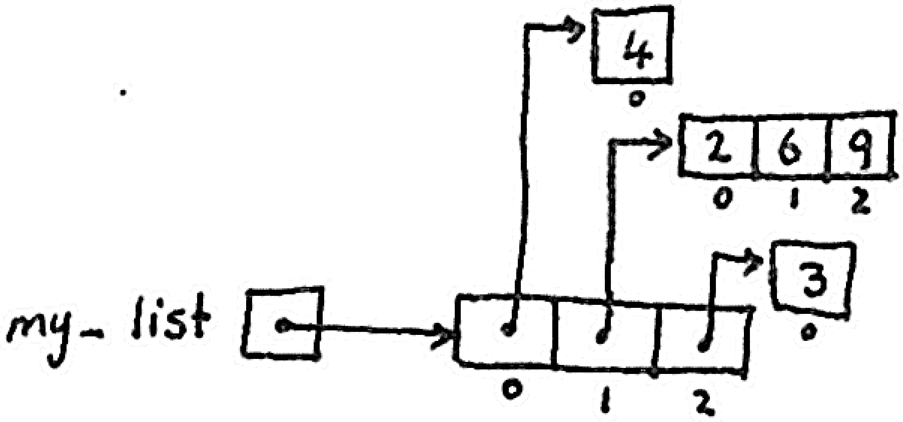
```
a_list =  [4, 2, 6, 9, 3]
```

can be represented with the diagram:



Draw a diagram to show the structure of the list created by the code:

```
my_list = [[4], [2, 6, 9], [3]]
```



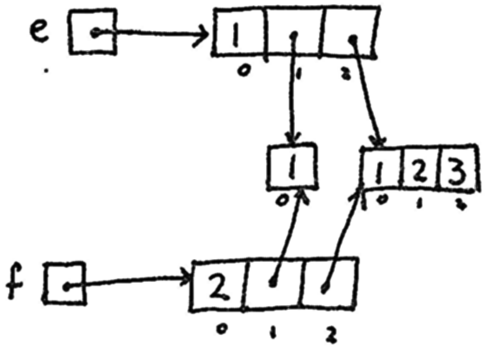
(5 marks)

Question 20: [5 marks]

In lecture 03, you were shown diagrams that illustrated how different lists were represented in memory. Use this approach to draw a diagram showing how the lists labelled **e** and **f** are represented in memory.

```
a = [1]
b = a
c = [1, 2, 3]
d = c

e = [1, a, c]
f = [2, b, d]
```

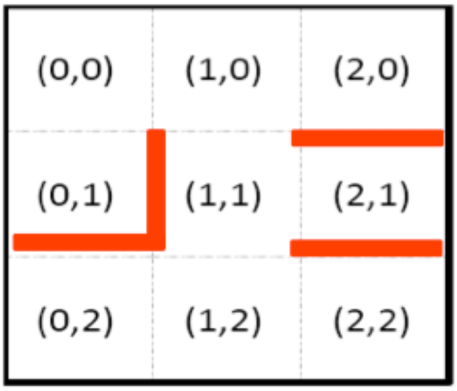


(5 marks)

Question 21: [5 marks]

Given the following dictionary representing a maze in the same format as used in Assignment 02, draw the maze on the grid provided. By drawing thick lines between adjacent cells that should not have a connection between them, clearly indicate where the walls are.

```
maze = {
 (0, 0) : [(0, 1), (1, 0)],
(0, 1) : [(0,0)],
(0, 2) : [(1, 2)],
(1, 0) : [(0,0), (1, 1), (2, 0)],
(1, 1) : [(1, 0), (1, 2), (2, 1)],
(1, 2) : [(0, 2), (1, 1), (2, 2)],
(2, 0) : [(1, 0)],
(2, 1) : [(1, 1)],
(2, 2) : [(1, 2)]
}
```



(5 marks)

Question 22 [3 marks]

Complete the following table regarding the operations of Python `Lists`. You need to complete the four missing bits of the **Description** column and the four parts in the **Complexity** column.

Python Lists	Description	Complexity
`append(item)`	adds an item to the **end** of the list	**O(1)**
`pop(0)`	removes the item at the **beginning** of the list	**O(n)**
`pop()`	removes the item at the **end** of the list	**O(1)**
`insert(0, item)`	adds an item to the **beginning** of the list	**O(n)**

(3 marks)

Question 23: [3 marks]

Complete the definition of the `Stack` class below.

```
class Stack:
    def __init__(self):
        self.items = []

    def is_empty(self):
        return self.items == []

    def push(self, item):   #has O(1) complexity
        self.items.append(item)
```

(1.5 marks)

```
    def pop(self):          #has O(1) complexity
        return self.items.pop()
```

(1.5 marks)

```
    def peek(self):
        return self.items[len(self.items) - 1]

    def size(self):
        return len(self.items)
```

Question 24: [4 marks]

Give the output produced by the following code fragment.

```
from Stack import Stack

s = Stack()

for i in range(10):
    if i % 2 == 0:
        s.push(i)

while not s.is_empty():
    print(s.pop())
```

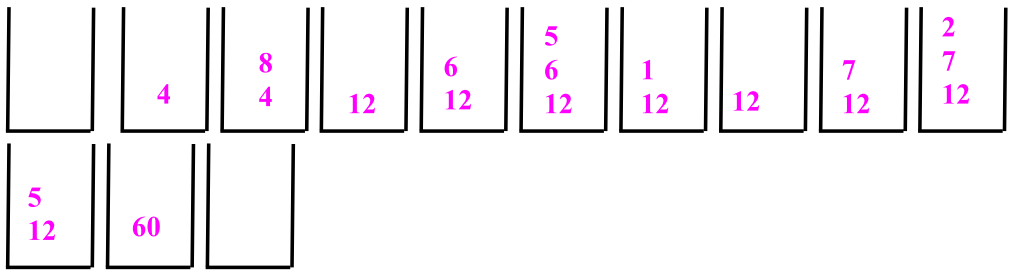
```
8
6
4
2
0
```

(4 marks)

Question 25: [5 marks]

Evaluate the following postfix expression using a stack structure. Show the status of the stack after each step.

4 8 + 6 5 – * 7 2 — *



(5 marks)