Office Use Only

MONASH University

ANSWER BOOKLET
Semester Two 2017
Examination Period

Faculty of Information Technology

EXAM CODES: **FIT1008**

TITLE OF PAPER: **Introduction to Computer Science – Answer Booklet**
EXAM DURATION: 2 hours writing time
READING AND NOTING TIME: 30 minutes

THIS PAPER IS FOR STUDENTS STUDYING AT: (tick where applicable)

☐ Berwick	✓ Clayton	✓ Malaysia	☐ Off Campus Learning	☐ Open Learning
☐ Caulfield	☐ Gippsland	☐ Peninsula	☐ Monash Extension	☐ Sth Africa
☐ Parkville	☐ Other (specify)			

During an exam, you must not have in your possession any item/material that has not been authorised for your exam. This includes books, notes, paper, electronic device/s, mobile phone, smart watch/device, calculator, pencil case, or writing on any part of your body. Any authorised items are listed below. Items/materials on your desk, chair, in your clothing or otherwise on your person will be deemed to be in your possession.

No examination materials are to be removed from the room. This includes retaining, copying, memorising or noting down content of exam material for personal use or to share with any other person by any means following your exam.
Failure to comply with the above instructions, or attempting to cheat or cheating in an exam is a discipline offence under Part 7 of the Monash University (Council) Regulations.

AUTHORISED MATERIALS

OPEN BOOK	☐ **YES**	✓ **NO**
CALCULATORS	☐ **YES**	✓ **NO**
SPECIFICALLY PERMITTED ITEMS	☐ **YES**	✓ **NO**

Table for Office Use Only

Page		Marks	Page		Marks
3		4	21		2
5		6	23		10
7		5	25		5
9		5	27		5
11		5	29		10
13		5	31		10
15		3	33		7
17		7	35		3
19		8	**Total**		100

Candidates must complete this section if required to write answers within this paper

STUDENT ID: __ __ __ __ __ __ __ __ __ DESK NUMBER: __ __ __ __ __

This page is intentionally left blank, use if needed but it will not be marked.

Question 1 – Short questions [15 marks = 10 x 1.5]

In this part you are required to answer ten short questions. Your answer should be concise. As a guideline, it should require no more space than the one provided.

(1) What is the purpose of using `jal` instead of `j` when working with functions in MIPS?

(2) How can we avoid primary clustering in open addressing collision resolution? Explain your answer.

(3) The best case time complexity for a method that traverses a list of elements can never be when the list is empty. Explain why.

(4) Mention and explain one disadvantage of using a Binary Search Tree over a Hash Table for implementing a Dictionary ADT.

7.5

(5) Can you modify a simple Linked List implementation to achieve constant time append to the end operations? Explain your answer.

(6) What is the difference between `is` and `==` in Python? Explain.

(7) What is the main advantage of a circular queue implementation over a linear one? Explain.

(8) When does the worst case time complexity arise in Bubble sort? Explain your answer.

(9) If the main concern is minimising the number of swaps, which of the non-recursive sorting algorithms (covered in the lectures) should be chosen and why?

(10) Draw a tree with two nodes that is both a Binary Search Tree and a maxHeap. Explain your choice.

This page is intentionally left blank, use if needed but it will not be marked.

7.5

Question 2 – Array Containers [10 marks = 3 + 2 + 5]

The following code gives a partial implementation of a `DoubleStack` class, which holds two Stacks in **a single array**, one starting from the front and one from the back. The class should have two push operations (`push1` and `push2`), two pop operations (`pop1` and `pop2`), and three length operations: `len1`, `len2` and `len`, which give the number of items on stack 1, on stack 2, and on the double stack, respectively. As it can be inferred from the partial implementation, the left-hand side of the array is used for the first stack, while the right-hand side of the array will be used for the second stack. The double stack will be full only when the total number of elements equals the maximum capacity of the array, as given on the constructor.

```
class DoubleStack:

    def __init__(self, max_capacity=30):
        self.array = [None]*max_capacity
        self.top1 = 0
        self.top2 = max_capacity-1

    def push1(self, item):
        if len(self) = len(self.array):
            raise Exception('Stack is full')
        else:
            self.array[self.top1] = item
            self.top1 += 1

  def pop2(self):
        assert self.len2() > 0
        item = self.array[self.top2]
        self.top2 +=1
        return item

  def len1(self):
        return self.top1
```

(a) Implement method `push2(self, item)`, which pushes `item` onto the second stack. This method should raise an `Exception` if there is no space available.

This page is intentionally left blank, use if needed but it will not be marked.

3

(b) Implement method `len2(self)`, which returns the number of items in the second stack.

(c) Implement method `__str__(self)`, which returns a string representing the elements of the two stacks separated by an & symbol. For example, the following string `"1 2 3 & 0 -3 "` would be returned for a double stack where the first stack contained the elements 1, 2, 3 (where 3 is on top) and the second contained elements 0 and -3 (where 0 is on top). Make sure you do not modify the stacks.

This page is intentionally left blank, use if needed but it will not be marked.

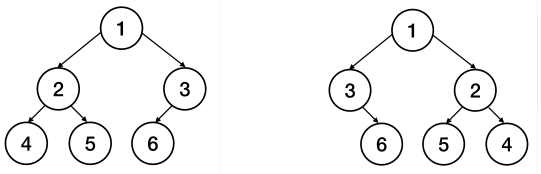
3

Question 3 – Binary Trees [12 marks = 5 + 2 + 2 + 3]

Consider the partial implementation of a binary tree given below, which uses the TreeNode class:

```
class TreeNode:

    def __init__(self, item=None, left=None, right=None):
        self.item = item
        self.left = left
        self.right = right

class BinaryTree:

    def __init__(self):
        self.root = None

    def is_empty(self):
        return self.root is None
```

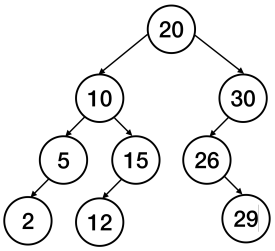
(a) Implement the method `mirror(self)` within the `BinaryTree` class, which modifies the tree to make it the mirror image of itself. For example, if the method is applied to the binary tree on the left, it will modify it to be the one on the right, and vice-versa.



This page is intentionally left blank, use if needed but it will not be marked.

5

(b) Assume you want to delete the root node in the Binary Search Tree shown on the right. Briefly enumerate the steps required to do so for this particular Tree, and draw the resulting Binary Search Tree.



(c) Consider an empty Binary Search Tree to which you add the integers 5, 1, 18, 6, 2, 0, 4, 10 in that order. Draw the tree obtained after the last insertion.

This page is intentionally left blank, use if needed but it will not be marked.

5

Question 4 – MIPS [8 marks = 3 + 5]

(a) Consider the following Python code that constructs and returns a string:

```
def hello(name):
    out = "Hello␣" + name + ",␣great␣to␣meet␣you."
    return out
```

In MIPS, local variables are created (allocated) during function entry and destroyed (deallocated) upon function exit. That is why they are stored in the System Stack. However, the MIPS translation of Python code to `x = hello("Maria"), print(x)` will successfully print `Hello Maria, great to meet you`. How is this possible given variable `out` has been deallocated by the time the string is printed? Briefly explain in terms of memory allocation.

This page is intentionally left blank, use if needed but it will not be marked.

3

(b) In the space next to the Python code, complete the MIPS program so that it constitutes a **faithful** translation of the original Python program, where `sec(a)` is a call to a function defined elsewhere. Remember to comment the code!

Python code	MIPS code	Comments

`def first(i, a):`
 	
`    result = i + sec(a)`
 	
`    return result`
 	
.		

This page is intentionally left blank, use if needed but it will not be marked.

3

Question 5 – Iterators [10 marks = 8 + 2]

In this question, you will be asked about iterators and their usage.

(a) Write up a class `FactorIterator` that can be used to iterate over all the factors of a given number n. That is, the code

```
x = FactorIterator(8)
it = iter(x)
next(it)
next(it)
next(it)
next(it)
```

will bound `it` to an iterator and print numbers 1, 2, 4 and 8, in that order.

This page is intentionally left blank, use if needed but it will not be marked.

8

(b) Write a Python function `odd_factors(n)` that *uses the iterator you just defined* to print all the odd factors of number `n` (e.g.,. for n=900, it would print 3,5,9, and 15)

This page is intentionally left blank, use if needed but it will not be marked.

8

Question 6 – Classes, Objects and Namespaces [5 marks]

Provide next to the Python code below, the result of each of the print statements (marked with comments from #1 to #11). If the result is an error, please explain why.

```
1  class myclass:
2      def __init__(self,x):
3          self.x = x
4
5      def print(self):
6          print(self.x)
7
8      def a(self):
9          self.x = self.x + 1
10
11     def b(self):
12         self.x = x + 2
13
14     def c(self):
15         x = self.x + 3
16
17 def a(x):
18     x = x - 1
19
20 def b():
21     x = x + 1
22
23 x = 1
24 myobject = myclass(x)
25 myobject.print() #1
26 x = 2
27 myobject.print() #2
28 myobject.x = 1
29 myobject.a()
30 myobject.print() #3
31 myobject.b()
32 myobject.print() #4
33 myobject.c()
34 myobject.print() #5
35 a(x)
36 print(x) #6
37 myclass.x = 1
38 myclass.print(myobject) #7
39 myobject.x = 2
40 myclass.print(myobject) #8
41 myobject.c()
42 print(myclass.x) #9
43 yourobject = myclass(myobject.x)
44 yourobject.a()
45 myobject.print() #10
46 b()
47 print(x) #11
```

This page is intentionally left blank, use if needed but it will not be marked.

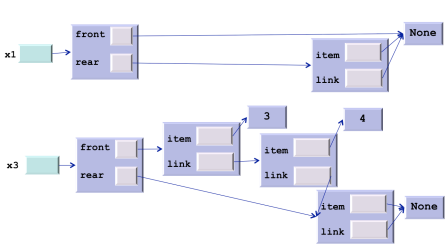
5

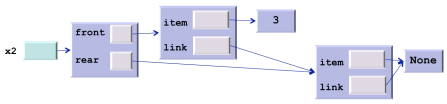
Question 7 – Linked Queues [10 marks = 4 + 4 + 2]

An invariant of the array-based queue implementation we have seen in the lectures, is that `self.rear` always points to the first empty slot at the rear of the queue. To replicate this `self.rear` invariant in the linked queue, consider the partial implementation given below, where the rear of the queue points to an empty but already created node, that is ready to be filled :

```
class Node:

    def __init__(self, item=None, link=None):
        self.item = item
        self.link = link

class LinkedQueue:

    def __init__(self):
        self.front = None
        self.rear = Node()

    def is_empty(self):
        return self.front is None
```

For example, the figures below show an empty queue (`x1`), a queue with one element (`x2`) and one with two elements (`x3`).





(a) Implement the usual queue method `append(self, item)` within the `LinkedQueue` class above.

This page is intentionally left blank, use if needed but it will not be marked.

4

(b) Likewise, implement the method `serve(self)` within the `LinkedQueue` class above. The method should accommodate all boundary cases including serving the last element in the queue.

This page is intentionally left blank, use if needed but it will not be marked.

4

Question 8 – Sorting [8 marks = 6 + 2]

In this question you must give a proof for each answer.
You are a lecturer in charge of a unit with m students and k tutors. Your tutors have marked $\frac{m}{k}$ exam papers each, and you need to sort all papers according to their marks. We suppose that comparing marks (and thus comparing exams) takes constant time.

(a) You propose the following distribution of work: each tutor will sort their $\frac{m}{k}$ papers, and you, the lecturer, will take those k stacks of papers and merge them into a single sorted stack of m papers. We suppose that each tutor requires exactly p^2 operations to sort p papers, and that you require exactly $k * p$ operations to merge those k stacks of p papers into a single sorted stack.

(a) What is the total number of operations required for this algorithm?

(b) For m fixed, what is the number k of tutors that minimises the total number of operations required?

(c) For m fixed, what is the number k of tutors that gives each tutor the same amount of work (i.e. operations) as the lecturer?

This page is intentionally left blank, use if needed but it will not be marked.

6

This page is intentionally left blank, use if needed but it will not be marked.

Question 9 – Priority Queues and Heaps [6 marks]

This question is about priority queues and heaps. Consider a priority queue implemented by a max Heap, where its elements are key value pairs stored using a tuple:

```
class MaxHeap:
       def __init__(self, max_capacity):
            self.count = 0
            self.the_array = build_array(max_capacity+1)
       # code with the rest of the functions provided by the class
```

Assume `maxHeap` has the usual methods, including `rise(self,index)` and `sink(self,index)`, which correctly rise and sink the element at array cell `index`. Write a method `changePriority(self,k,v)` for the `maxHeap` class above, which first finds the element with the given value v (note that all values are unique) and then changes the priority (key) of that element to some new value k, raising an exception if not found.

10

This page is intentionally left blank, use if needed but it will not be marked.

Question 10 – Hash Tables [11 marks = 2 + 7 + 2]

Consider a hash table implementation where collisions are handled with quadratic probing. Also, the implementation uses method `rehash(self)` that is called after adding an element to the table if the load factor reaches or exceeds 1/2. The method will double the size of the table and reinsert each key.

(a) Assuming the initial size of the table is 4, what is the size of the table after inserting 12 different keys into the above hash table. Explain.

(b) Draw the content of the hash table after inserting keys 30, 18, 34, 42, 39 (with associated hashed value given by the table below), assuming the initial size of the table is 3 and there is already one key (14) inside as shown below. Note that the method `rehash(self)` will need to be called while you are adding the keys, as described above.

Table Size	Key	Hashed Value
3	14	0
	30	1
	18	1
6	14	3
	30	1
	18	1
	34	5
	42	1
	39	4
12	14	9
	30	1
	18	1
	34	5
	42	1
	39	10

0	1	2
14		

7

This page is intentionally left blank, use if needed but it will not be marked.

(c) Suppose that, instead of using quadratic probing to handle collision, the hash table uses separate chaining implementation using sorted linked list. What would be the best-case and worst-case time complexity of this implementation during insertion? Explain.

END OF EXAM.

7