**Unit 3 Algorithmics**

**Basic ADTs and Signatures**

1. A polygon on a Cartesian plane is represented by its vertices given as coordinates. Each polygon is assigned a unique name, e.g. “Steve’s Dream Pentagon”.
2. Describe a combination of ADTs to hold the complete collection of polygons. It can help to give examples of how these are to be used.
3. Write the signature for a function to:
4. Add a polygon to the collection based on its name and coordinates
5. Check to see if a polygon exists with a given name
6. Translate a particular polygon x units to the right and y units up.
7. Google Tasks allows the user to define a number of to-do lists. Each list has a number of tasks. For each task, its name and the date it is due is recorded.
   1. Describe a combination of ADTs to capture this information. The user wants to be able to see tasks in order of priority.
   2. Write the signature for a function to:
      1. Add a task
      2. Remove a task
      3. List the next 5 tasks due from a given list
      4. Check to see how many tasks overall are due today
   3. Write pseudocode for a function to list the next 5 tasks due. Your function should avoid using unnecessary storage space in memory.
8. A warehouse manager needs an inventory management system. For each type of item, the manager needs to know its name, price, quantity in stock, and a list of parts it is commonly sold with.
   1. Describe a combination of ADTs to capture this information.
   2. Write the signature for a function to:
      1. Add a new part
      2. Increase the quantity held of an existing part
      3. Check to see if, for a given part, all of the parts with which it is commonly sold are in stock.
   3. Write pseudocode for b iii.

Suggested Solutions

1.

Each coordinate pair makes up a list (or could be an array as it always has a length of 2.)

We store a list of these lists for each shape.

Then it makes sense to have a dictionary of shapes, where the key is the name of the shape (since this is unique) and the value is the lists of pairs of coordinates. This also makes looking up a shape’s coordinates based on its name easier, though this isn’t specified as a requirement so you could just have a list of lists of lists.

Add: dictionary x string x list 🡪 dictionary

(We specify the dictionary even though it’s assumed we are only working with one dictionary overall. Also you could say “list of lists” in place of “list”).

Check: dictionary x string 🡪 Boolean

Translate: dictionary x string x int x int 🡪 dictionary

Note that we have to state the data types, not what they represent; however it doesn’t hurt to add them, in which case underneath each line you could write:

Add: dictionary x name x list of coords 🡪 dictionary

Check: dictionary x name 🡪 Boolean

Translate: dictionary x name x up x right 🡪 dictionary.

2.

Each to-do list is recorded as a min priority queue with the date, where earlier dates are evaluated as smaller values. Each task is an item in the PQ. Together the to-do lists could be stored in a dictionary with the list name as the key and the min PQ as the value. (Doesn’t have to be a dictionary, could be a list of lists.)

Add: string x string x int x dictionary 🡪 dictionary

Task\_name x list\_name x date x dictionary 🡪 dictionary

Remove: string x dictionary 🡪 dictionary

Task\_name x dictionary 🡪 dictionary

ListNextFive: string x dictionary 🡪 list

List\_name x dictionary 🡪 list of tasks

CheckToday: dictionary 🡪 int

define listNextFive:

# Need to check each list to find the five tasks overall with

# the lowest priority.

# This version cycles through the tasks\_lists stored in

# all\_lists to find the task with the smallest priority, then

# adds this to nextfive.

nextfive = empty min priority queue

while length of nextfive < 5:

min = infinity

for task\_list in all\_lists:

nextitem = peek at top item of task\_list and dequeue

if weight of nextitem < min:

min = weight of nextitem

mintask = name of nextitem

end for

add mintask to nextfive with weight min

end while

return nextfive

Note that the question requires you to use minimum storage space, so you can’t just cycle through every task list, add all tasks to one giant list and then choose the top five. Also you might have added code to identify which sublist each task was from.

3.

A dictionary of lists, where the key is the item name and the value is a list containing the other details.

addPart: dictionary x string x int x int x list 🡪 dictionary

increaseQty: dictionary x string x int 🡪 dictionary

checkOthers: dictionary x string 🡪 Boolean

Algorithm CheckParts(mypart):

for each part in mypart.associated\_parts\_list:

if part.qty\_in\_stock = 0 then return False

return True