Chawakorn Chaichanawirote
ID: 59340500013
Tachadol Suthisomboon
ID: 59340500025

FRA142
Smart Warehouse Phase 1 Report

Software Design

StackClass is derived from the LinkedList class with two extra functions: push and pop

i. StackClass.push is a LinkedList.add with the position parameter equal to the size of the stack, meaning that push will always add on top of the stack
ii. StackClass.pop basically assigns a new tail to be the one previous from the current tail

Queue is also derived from LinkedList with different add and remove funtions from a LinkedList

i. Queue.add is the same as StackClass.add, as it will always add on top
ii. Queue.remove is the opposite of StackClass.pop, shortening the list by replacing the head

Test cases can be found in test.py in the appendix. It iterates through a for loop adding to and removing items from a stack and a queue, giving opposite results.

Warehouse Progress

For the warehouse we chose to use arrays and queues as the data structures to utilize for phase 1.

Step 1: Initialization

Create warehouse, conveyor belt, and a command queue

Step 2: Input

Take in input and store in a command queue, so that commands can be

```
print ("Conveyor Belt Generated")
print ("Input your commands")
print (" 0XXXX \n"
       "Retrieve a product with ID XXXX \n"
       "1XXXX \n"
       "Store a product with ID XXXX \n"
       "2XY00 \n"
       "Sort warehouse X at row Y \n"
       "30000 \n"
       "Retrieve a product from the conveyor belt \n"
       "40000 \n"
       "Output information on all of the warehouses \n"
       "5XXXX \n"
       "Search for a product ID XXXX \n"
       "9XXXXYYYY \n"
       "Manually put a product ID XXXX at position YYYY \n")
newcom = True
while newcom:
    comm = input("Please enter command\n")
    commandQue.add(Node(comm))
    yesno = ''
    while yesno != 'y' and yesno != 'n':
        yesno = input("Would you like to enter another command? y/n\n")
        yesno = yesno.lower()
    if yesno == 'n':
        newcom = False
```

executed in correct order. Repeated inputs are achieved using while loops

```
for i in range(commandQue.size):
    executeCommand(commandQue.remove())

def executeCommand(Command):
    Command = Command.lower()
    if len(Command) == 0: #this is a quick fix to avoid index range errors
        Command = " "
    if Command[0] in ['0','1','2','3','4','5','6','7','8','9']:
```

Step 3: Execution

Execute commands from the command queue by interpreting the input, separating by type. The algorithms for each command are as follows.

Command type 0XXXX:

1. Find the X and Y coordinates from the slot number(Call FindX, FindY)
2. Check if the slot we have to retrieve the product from is empty(Warehouse1[x][y] == None)
3. Check if the belt has space available(belt.size <10)
4. Add the product to the conveyor belt. (belt.add(Node(Warehouse1[x][y])

Command type 1XXXX:

1. Find X and Y coordinates
2. Check if the slot we have to store the product in is empty
3. Store the product in the warehouse array

Command type 30000:

1. Check if the belt is empty(belt.size != 0)
2. Retrieve product(belt.remove())

Command type 40000:

1. Iterate through each matrix dimension with for loops to check whether each slot is empty or not
2. When not empty, add the product ID to the output txt and increase the count
3. Print out the output txt and the count

```
elif Command[0] == '4':
    print ("Warehouse A")
    print("row 1")
    txt = "products in row1 :"
    count = 0
    for x in range(10):
        for y in range(10):
            if Warehouse1 [x][y] != None:
                txt = txt + " A1" + str(x) + str(y) +"  "
                count +=1
    print (txt)
    print ("Total products: " + str(count))
```

Command type 9XXXXYYYY

1. Find X and Y for each Slot
2. Check if the slot we are moving the product from is empty
3. Check if the slot we are moving the product to is empty
4. Move the product(Warehouse1[newx][newy] = Warehouse1[x][y])

Please refer to the appendix for further information.