

DSA Project

by Lim SI NI

Submission date: 15-Jan-2024 06:55PM (UTC-0800)

Submission ID: 2271664091

File name: DSA_Project_Report_Center_Point_1.pdf (679.36K)

Word count: 2868

Character count: 12867



UTM
UNIVERSITI TEKNOLOGI MALAYSIA

FACULTY OF COMPUTING
UTM Johor Bahru

SESSION 2023/2024 SEMESTER 1

SECJ - DATA STRUCTURE AND ALGORITHMS

Project

Group Name: Center Point

Topics: Inventory Management System

Lecturer: MDM LIZAWATI BINTI MI YUSUF

GROUP MEMBERS:

No.	Name	Matric No.
1	LIM SI NI	A22EC0070
2	ONG KAI XUEN	A22EC0100
3	SOH FEI ZHEN	A22EC0272

Table of Content

Table of Content	2
1. Objective	3
2. Synopsis	4
3. Problem Analysis	5
4. System Design	6
4.1. Pseudocode	6
4.2. Flow Chart	14
4.3. Class Diagram	17
5. Data Structure Concept Implementation	18
6. User Manual/Guide	23
7. Conclusion	29

1. Objective

In our data structure and algorithm project, the main objectives of developing the Inventory management system are :

- Develop an Inventory Management System using queue
- Manage inventory in a warehouse by features
 1. Add Inventory
 2. Delete Inventory
 3. Sort Inventory
 4. Find Inventory
 5. Display Inventory (based on category or display all)
 6. Update current inventory information
 7. Check low stock inventory

2. Synopsis

The inventory management system is essential for one business or organization to track or manage its warehouse. Our inventory management system is designed to organize the inventory. User can organize the inventory by adding the inventory, deleting the inventory, sorting the inventory by its ID in ascending order, and finding the inventory by item or by its category. Since we are using a queue data structure implementation concept, user can only add the inventory at the last position in the list, and delete the first element in the inventory. In addition, user can choose to display the inventory list whether by its category or display full inventory list. Then, we also implement extra features such as updating inventory and checking stock item. User can update the inventory information by entering the item code, then choosing the part they would like to update such as name, type, quantity, or price. Then, the low-stock inventory feature helps the user to check and analyze all the low-stock items and display as a list. The item which has a quantity of less than 10 will be counted as low-stock inventory.

3. Problem Analysis

Regarding warehouse management, the current manual inventory system often causes issues related to efficiency and accuracy. This mode of operation is not automated to achieve maximum resource optimization, and it may often cause errors and inefficiencies. To overcome these challenges, our Inventory Management System aims at using a queue data structure for an organized and automated solution.

This system simplifies the warehousing process through the ability to add and delete inventory items by the FIFO (First In First Out) rule. It also aims to improve the organization by implementing a sorting feature to allow quick searches and offering a user-friendly interface to view and edit inventory data. The real-time updates and a low-stock alert function, it enable timely actions to avoid shortages. Through these goals and characteristics, the Inventory Management System will provide efficiency and accuracy for every warehouse management.

4. System Design

4.1. Pseudocode

4.1.1. Add Inventory

1. Start
2. Get inventory code
 - 2.1 check code format
 - 2.1.1 if code length >4
 - 2.1.1.1 set codeformat== false
 - 2.1.2 if first character is not 'I'
 - 2.1.2.1 set codeformat== false
 - 2.2 check duplicate code
 - 2.2.1 if code is empty
 - 2.2.1.1 set duplicateCode ==false
 - 2.2.2 Set temp=head
 - 2.2.3 Do
 - 2.2.3.1 {if (temp->getCode() == code)
 - 2.2.3.1.1 set duplicateCode ==true
 - 2.2.3.1.2 temp = temp->next;
 - 2.2.3.1.3 } while (temp != back->next);
 - 2.2.4 set duplicateCode ==false
 - 2.3 If (codeformat is true AND duplicateCode is false)
 - 2.3.1 go to step 2.5
 - 2.4 Else
 - 2.4.1 if (codeformat==false)
 - 2.4.1.1 cout << " Re-enter your code (IXXX) where X should be a digit."
 - 2.4.1.2 go to step 2
 - 2.4.2 else if (duplicateCode==true)
 - 2.4.2.1 cout << " The code already exists."
 - 2.4.2.2 go to step 2
 - 2.5 Change code to upper case
3. Get inventory name
 - 3.1 check duplicate name
 - 3.1.1 if (isEmpty())
 - 3.1.1.1 set duplicateName== false;
 - 3.1.2 Do
 - 3.1.2.1 if (temp->getName() == name)
 - 3.1.2.1.1 set duplicateName== true
 - 3.1.2.1.2 temp = temp->next
 - 3.1.2.1.3 } while (temp != back->next);

```

    3.1.3 set duplicateName== false;
3.2 If duplicateName==false
    3.2.1 Go to step 4
3.3 Else
    3.3.1 cout << "Invalid name! The name already exists.
        Re-enter."
    3.3.2 Go to step 3
4. Get inventory type
    4.1 change inventory type to upper case
5. Get inventory quantity
    5.1 while quantity<0
        5.1.1 cout << "Invalid quantity! Reenter! "
        5.1.2 go to step 5
6. Get inventory price
7. Set *newInventory = new Inventory(code, name, type, quantity,
    price)
8. if list is empty
    8.1 Set back = newInventory
    8.2 Set newInventory->next = back
9. Else
    9.1 Set newInventory->next = back->next
    9.2 Set back->next = newInventory
    9.3 Set back = newInventory
10. Display congrats message
11. End

```

4.1.2. Delete Inventory

```

1. Start
2. If list is empty
    2.1 cout << "No nodes left!!"
3. Else
    3.1 Set Inventory *temp = back->next;
    3.2 if (back->next != back)
        3.2.1 Set back->next = temp->next
    3.3 else
        3.3.1 Set back = NULL;
    3.4 Set temp->next =NULL
    3.5 delete temp;
4. Display congrats message
5. End

```


- 4.1.3. Sort the Inventory
 1. Start
 2. If queue is empty
 - 2.1. Print "The queue is empty"
 3. Else
 - 3.1. Set *first=back->next
 - 3.2. Set *curr=back->next
 - 3.3. Set *sBack=new Inventory("", "", "", 0, 0)
 - 3.4. Set sBack->next = sBack
 - 3.5. Do
 - 3.5.1. Set *Next = curr->next
 - 3.5.2. if (curr->getCode() < sBack->next->getCode())
 - 3.5.2.1. curr->next = sBack->next
 - 3.5.2.2. sBack->next = curr
 - 3.5.3. Else
 - 3.5.3.1. Set *temp = sBack->next->next
 - 3.5.3.2. Set *prev = sBack->next
 - 3.5.3.3. while (temp != sBack->next && curr->getCode() > temp->getCode())
 - 3.5.3.3.1. prev = temp
 - 3.5.3.3.2. temp = temp->next
 - 3.5.3.4. if (temp == sBack->next)
 - 3.5.3.4.1. curr->next = sBack->next
 - 3.5.3.4.2. sBack->next = curr
 - 3.5.3.4.3. sBack = curr
 - 3.5.3.5. Else
 - 3.5.3.5.1. prev->next = curr
 - 3.5.3.5.2. curr->next = temp
 - 3.5.4. curr=Next
 - 3.5.5. while (curr != first)
 - 3.6. sBack->next = sBack->next->next
 - 3.7. back = sBack
 4. Print "Inventory List is sorted by Inventory Code in Ascending\n"
 5. Print queue
 6. End

4.1.4. Find an Inventory

1. Start
2. Set *temp = back->next
3. Set found = 0
4. Print find menu
5. Get find choice
6. if (fChoice == 1)
 - 6.1. Get search key
 - 6.2. Do
 - 6.2.1. if (temp->getCode() == sKey)
 - 6.2.1.1. found=true
 - 6.2.1.2. Print the inventory detail
 - 6.2.2. tem=temp->next
 - 6.3. while (temp != back->next && found == false)
7. else if (fChoice == 2)
 - 7.1. Get search key
 - 7.2. Do
 - 7.2.1. if (temp->getCode() == sKey)
 - 7.2.1.1. found=true
 - 7.2.1.2. Print the inventory detail
 - 7.2.2. tem=temp->next
 - 7.3. while (temp != back->next && found == false)
8. Else
 - 8.1. Go to step 10
9. If(found==false)
 - 9.1. Print "The entered value is invalid"
10. End

4.1.5. Display Inventory List based on Category

1. Start
2. Get category
3. Set found=false
4. Set *temp=back->next
5. Do
 - 5.1. if (temp->getType() == cat)
 - 5.1.1. Show temp->getCode, temp->getName,
temp->getType, temp->getQuantity, temp->getPrice
 - 5.1.2. found=true
 - 5.2. temp = temp->next
6. while (temp != back->next)
7. if (!found)
 - 7.1. Print "There is no such category in inventory list. Pls try again"
8. End

4.1.6. Display Full Inventory List

1. Start
2. Set *temp = back->next
3. Do
 - 3.1 Display temp->getCode()
 - 3.2 Display temp->getName()
 - 3.3 Display temp->getType()
 - 3.4 Display temp->getQuantity()
 - 3.5 Display temp->getPrice()
 - 3.6 Set temp = temp->next;
 - 3.6 while (temp != back->next)
4. End

4.1.7. Update an Inventory

1. Start
2. Get iCode, iName, iType, choice, iQuantity, iPrice
3. Set temp = back->next
4. Set found = false
5. Do
 - 5.1. If (temp->getCode == iCode)
 - 5.1.1. found = true
 - 5.1.2. printDetail(temp)
 - 5.1.3. Break
 - 5.2. temp = temp->next
6. While (temp != back->next)
7. If (!found)
 - 7.1. Show "The entered inventory code is invalid!"
 - 7.2. return
8. Set choice = updateMenu()
9. If (choice == 1)
 - 9.1. Get iName
 - 9.2. iName = changeToUpper(iName)
 - 9.3. Set dup = false
 - 9.4. Set findDup = back->next
 - 9.5. Do
 - 9.5.1. If (iName == findDup->getName())
 - 9.5.1.1. dup = true
 - 9.5.1.2. break
 - 9.5.2. findDup = findDup->next
 - 9.6. While (findDup != back->next)
 - 9.7. If (dup)
 - 9.7.1. Show "The entered inventory name is duplicate!"
"Unsuccessfully update!"
 - 9.8. Else
 - 9.8.1. temp->putName(iName)
 - 9.8.2. Show "The inventory with code [" + iCode + "] is updated!"
 - 9.8.3. printDetail(temp)
 - 9.9. Break
10. Else if (choice == 2)
 - 10.1. Get iType
 - 10.2. iType = changeToUpper(iType)
 - 10.3. Set temp->putType(iType)

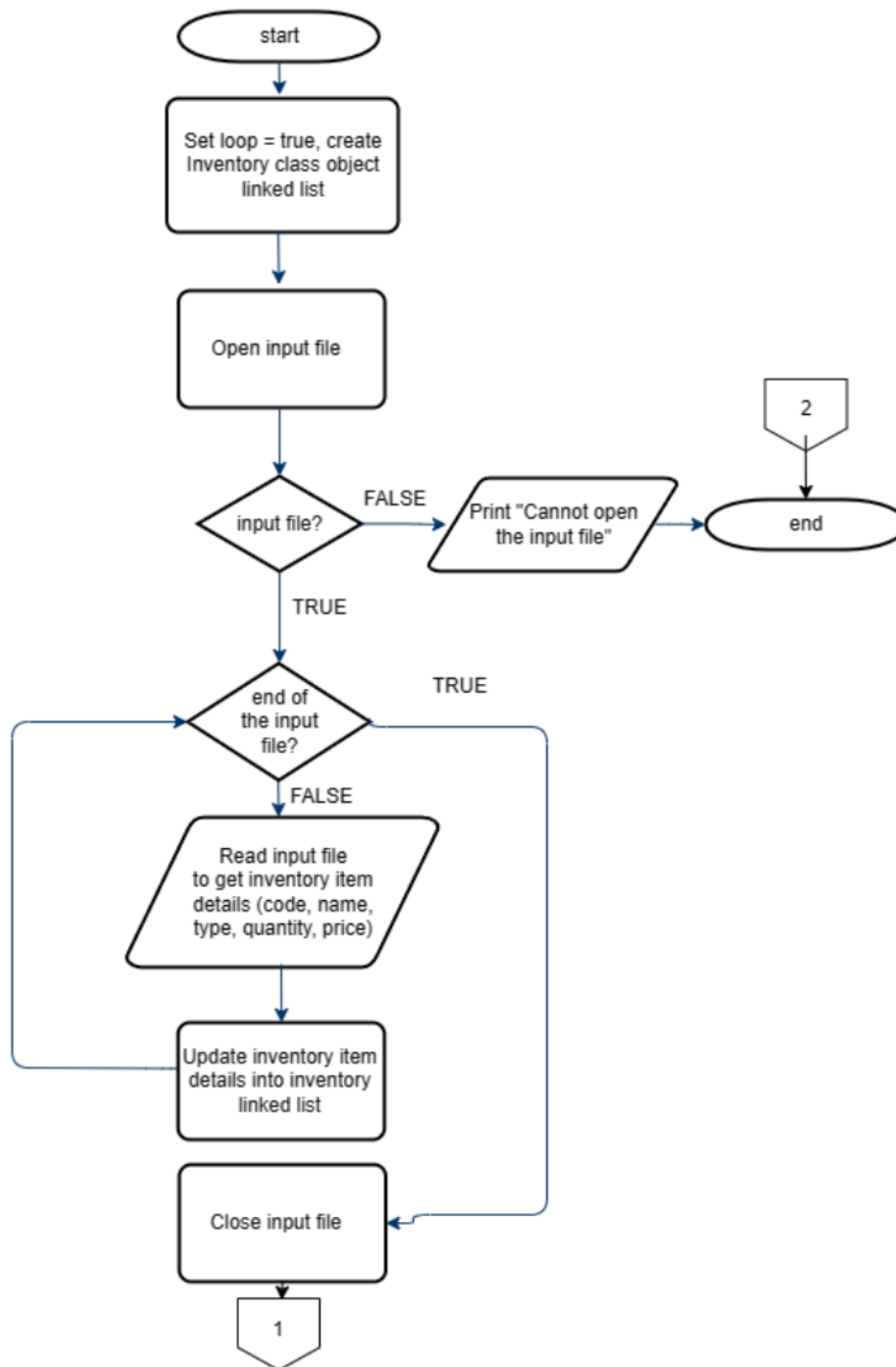
```

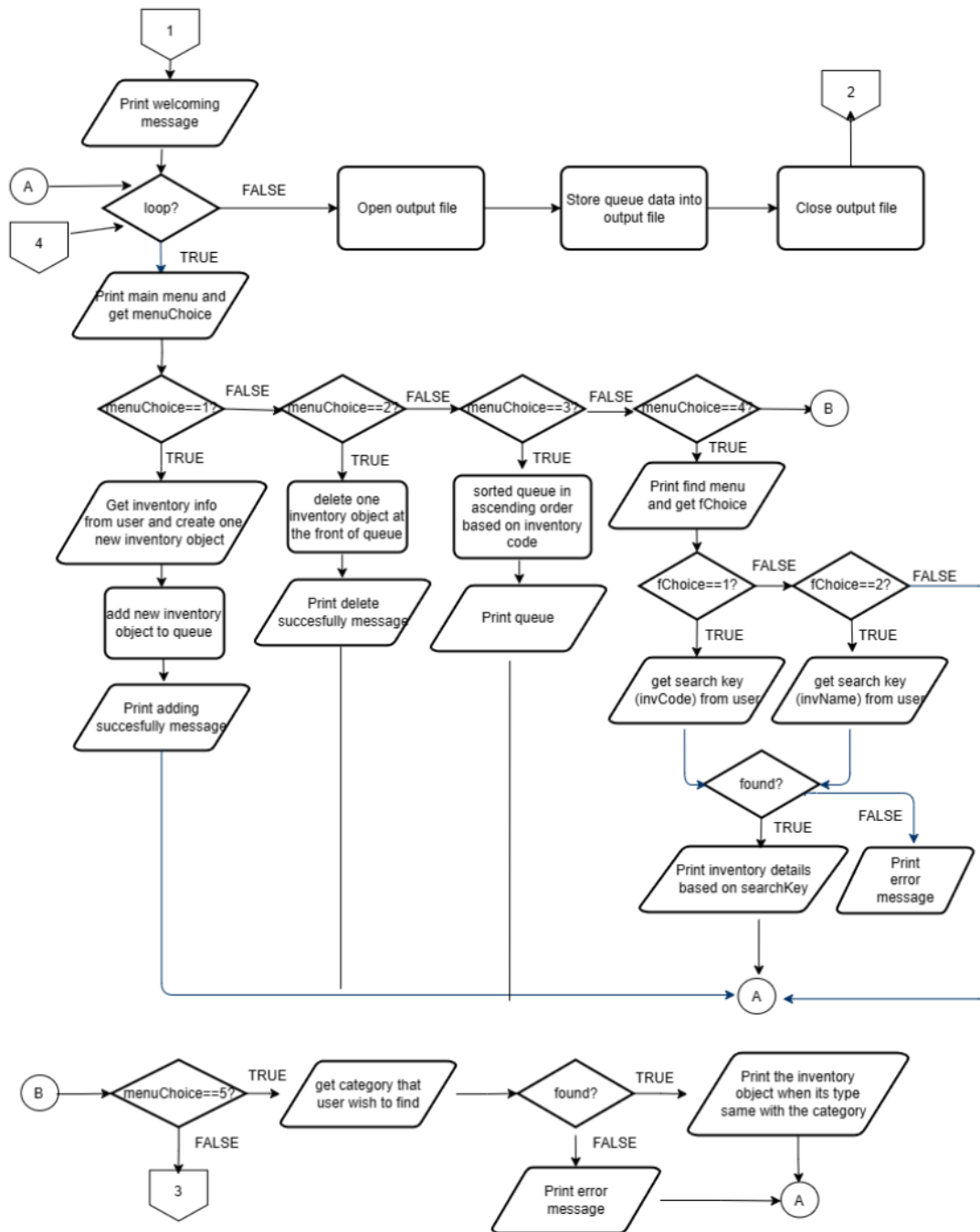
10.4.    display "The inventory with code [" + iCode + "] is
         updated!"
10.5.    printDetail(temp)
10.6.    break
11.     Else if (choice == 3)
11.1.    Get iQuantity
11.2.    If (chekcQuantity(iQuantity)
11.2.1.    temp->putQuantity(iQuantity)
11.2.2.    Show "The inventory with code [" + iCode + "] is
         updated!"
11.2.3.    printDetail(temp)
11.3.    Else
11.3.1.    Show "The entered quantity is negative value!"
         "Unsuccessfully update!"
11.4.    break
12.     Else if (choice == 4)
12.1.    Get iPrice
12.2.    temp->putPrice(iPrice)
12.3.    Show "The inventory with code [" + iCode + "] is
         updated!"
12.4.    printDetail(temp)
12.5.    Break
13.     Else
13.1.    return
14.     End

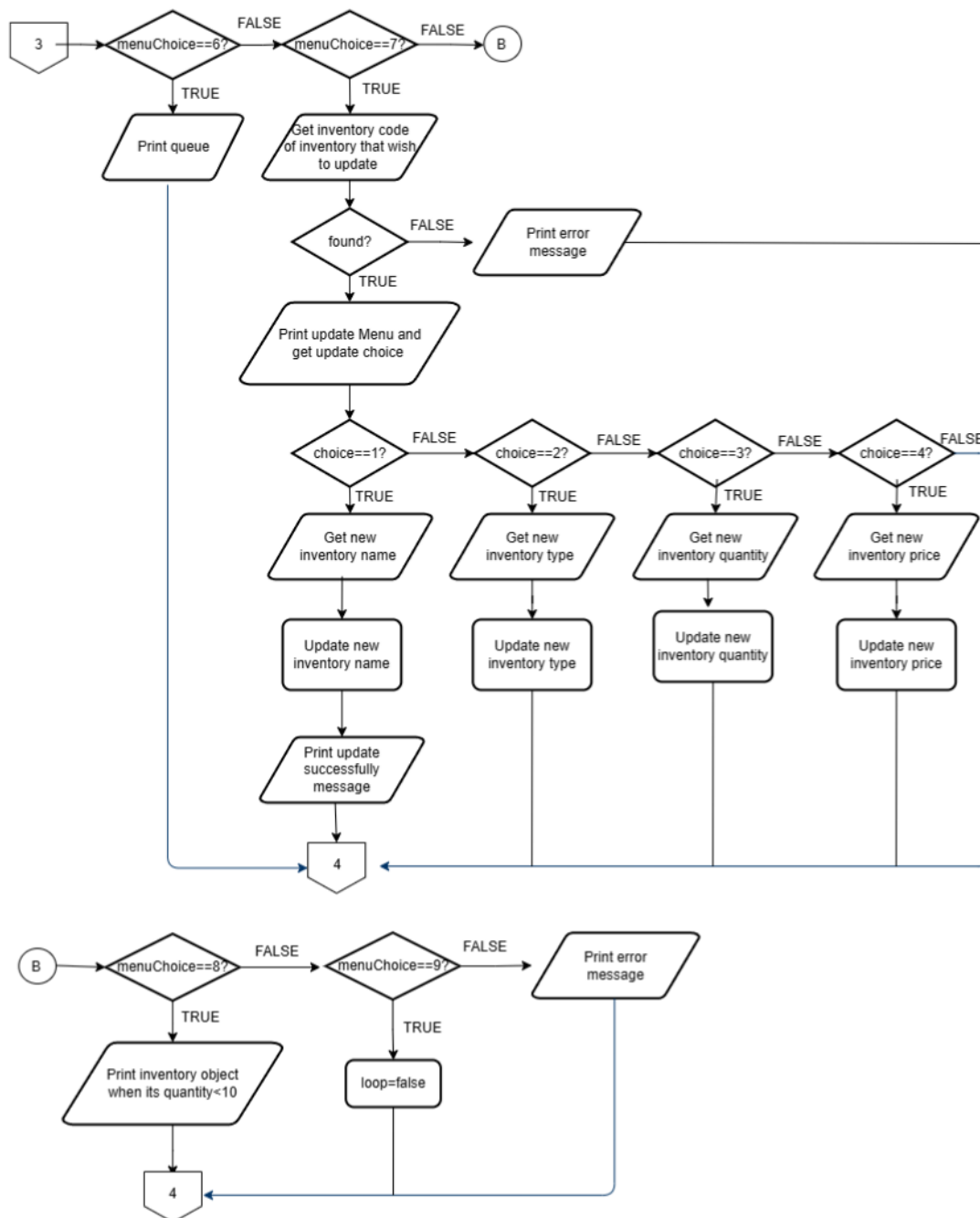
```

- 4.1.8. Check Low Stock Inventory
1. Start
 2. Set min = 10
 3. Set temp = back->next
 4. Show Low Stock Inventory interface
 5. Do
 - 5.1. If (temp->getQuantity < min)
 - 5.1.1. Show temp->getCode, temp->getName,
temp->getType, temp->getQuantity,
temp->getPrice
 - 5.2. temp = temp->next
 6. While (temp != back->next)
 7. End

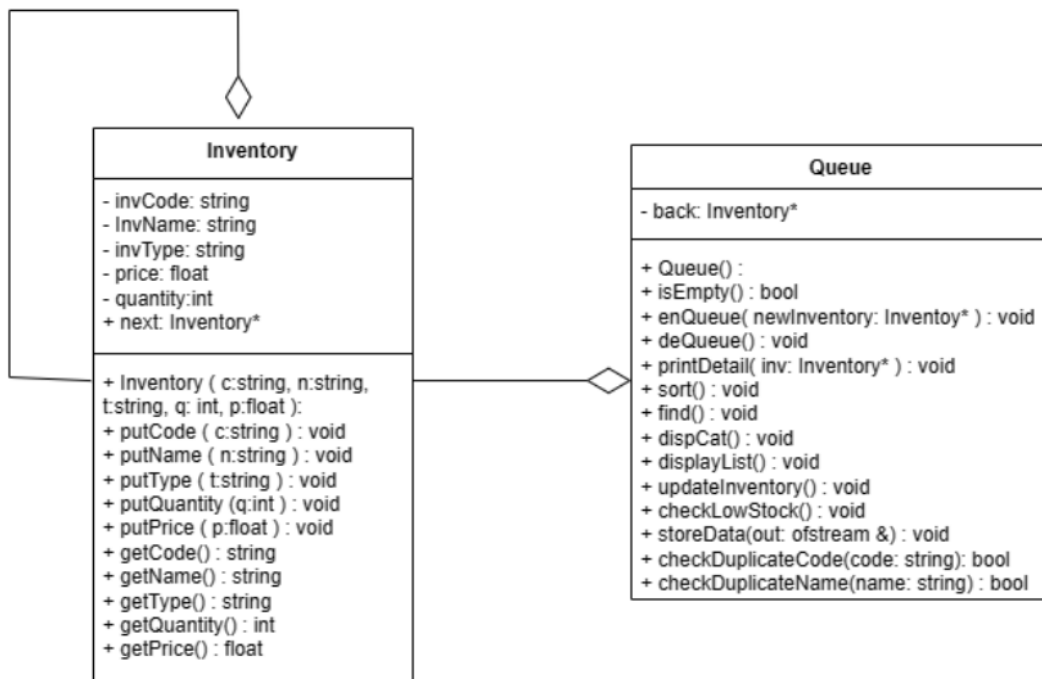
4.2. Flow Chart







4.3. Class Diagram



5. Data Structure Concept Implementation

The data structure that is implemented in this program is Queue. Queue has ¹First-in, first-out (FIFO) property where the first item inserted into a queue will be the first item to leave the queue and at the same time the middle elements are logically inaccessible. Here, we apply the circular linked list to represent the queue. This type of linked list can provide flexible size and do not have rightward-drifting problems.

- Initial the queue by setting back = NULL

```
Queue() { back = NULL; }
```

- bool isEmpty()

Return true when back is equal to NULL

```
bool isEmpty() { return back == NULL; }
```

- void enqueue(Inventory *newInventory)

Add an new inventory to the back of the queue

```
// Add Inventory
void enqueue(Inventory *newInventory)
{
    // case 1: queue is empty
    if (isEmpty())
    {
        back = newInventory;
        newInventory->next = back;
    }

    // case 2 : queue have thing inside
    else
    {
        newInventory->next = back->next;
        back->next = newInventory;
        back = newInventory;
    }
}
```

- If the queue is empty, set back = newInventory, newInventory->next = back;
- Else, newInventory->next = back->next (first element in queue), add a new element at the back of queue and set back = newInventory

- void deQueue()

Delete the first element in the queue

```
// Delete Inventory
void deQueue()
{
    // is empty
    if (isEmpty())
    {
        cout << "No nodes left!!" << endl;
    }
    else
    {
        Inventory *temp = back->next;

        if (back->next != back)
        { // more than 1 nodes
            back->next = temp->next;
        }

        else
        { // left only 1 nodes
            back = NULL;
        }
        temp->next = NULL;
        delete temp;
    }
}
```

- If the queue is empty, then pop out the "No nodes left" message.
- Else, set temp = back->next (first element in the queue)
 - If there is more than 1 node in the queue, link the last node the the second node in the queue
 - Else, there is only 1 node left in the queue, directly set back as NULL
- Set temp->next as NULL and delete temp

- void displayList()

```
Inventory *temp = back->next;

do
{
    cout << left << setw(20) << temp->getCode()
        << setw(20) << temp->getName()
        << setw(20) << temp->getType()
        << setw(15) << temp->getQuantity()
        << setw(10) << fixed << setprecision(2) << temp->getPrice() << endl
        << endl;

    temp = temp->next;
} while (temp != back->next);
```

- Set temp = back->next (first element in the queue)
- Start looping to display the detail of temp by calling the accessors
- Set temp = temp->next
- End the loopint when temp == back->next (first element in the queue)

- void sort()

If the queue is empty, display "The queue is empty" message

```
if (isEmpty())
    cout << "The queue is empty\n";
```

Else, set node *first* and *curr* which point to the first node of current linked list

Also create a new circular linked list named *sBack* which the first node is a empty node

```
Inventory *first = back->next;
Inventory *curr = back->next;
Inventory *sBack = new Inventory("", "", "", 0, 0);
sBack->next = sBack;
```

Start looping and stop until *curr == first*

In the looping, set a new pointer *Next* which is the second node of the current linked list.

- If the *curr* node code value is smaller than the first node in the *sBack*, add the *curr* node before the *sBack* first node
- Else, do another looping to find the position where to put the current node of current linked list to *sBack* and add it into the *sBack*.

```
while (temp != sBack->next && curr->getCode() > temp->getCode())
{
    prev = temp;
    temp = temp->next;
}
```

- At the meantime, if already achieved the last node of the *sBack*, directly insert the node at the end of the *sBack*.

```
if (temp == sBack->next)
{
    curr->next = sBack->next;
    sBack->next = curr;
    sBack = curr;
}
```

- Delete the first empty node of the *sBack*.

```
// delete first node - the zero data
sBack->next = sBack->next->next;
```

- Replace the original list with sorted list

```
// to link sortedQueue back to original queue
back = sBack;
```

- void find()
 - Set a *temp* node where point to the first node of the list

```
Inventory *temp = back->next;
```

[Example: Find an inventory by using inventory code]

- Do the looping to compare the user input code with the code of *temp*, stop the loop when it is found or temp is the first node of the list.

```
do
{
    if (temp->getCode() == sKey)
    {
        found = true;
        printDetail(temp);
    }
    temp = temp->next;
} while (temp != back->next && found == false);
```

- If the code is the same with the entered code, set found = true and print the details of inventory.
 - Else, continue to compare the code of the next node with the entered code.
- If it already break the loop, still cannot find the node that has the same value as the entered code, pop out a message that shows unsuccessfully finding the inventory.

```
if (found == false)
{
    cout << "\nThe entered value is invalid!\n\n";
}
```

6. User Manual/Guide

Main Menu

```

##### WELCOME TO INVENTORY MANAGEMENT SYSTEM #####
What do you need?
1. Add Inventory
2. Delete Inventory
3. Sort the Inventory
4. Find an Inventory
5. Display Inventory List based on Category
6. Display Full Inventory List
7. Update an Inventory
8. Check Low Stock Inventory
9. Exit
Enter your choice: |

```

1. Add Inventory

Enter the information of an inventory which includes inventory code, inventory name, inventory type, quantity and price.

```

-----New inventory Info-----
Enter Inventory Code: I013
Enter Inventory Name: SPOON
Enter Inventory Type: CUTLERY
Enter Quantity: 23
Enter Price: 2.30

```

The new inventory will automatically insert at the back of the list and a success message will pop out.

```

Congrats! You inventory has been added!

```

Constraints

**The inventory code is restricted to format "IXXX" where X should be a digit.

```

Invalid code! Re-enter your code (IXXX) where X should be a digit.

```

**The inventory code cannot be duplicated

```

Invalid code! The code already exists.

```

**The inventory name cannot be duplicated

```

Invalid name! The name already exists. Re-enter.

```

**The quantity cannot in negative value

```

Invalid quantity! Reenter!

```


2. Delete Inventory

The first inventory of the list will be deleted.

Congrats! Your first inventory has been deleted!

3. Sort the Inventory

Sort the inventory by inventory code in ascending order.

Inventory Code	Inventory Name	Inventory Type	Quantity	Price
I001	LAPTOP	HARDWARE	20	23.00
I002	STORY BOOK	BOOK	9	100.00
I003	NEWSPAPER	BOOK	130	1.40
I004	MOUSE PAD	HARDWARE	2	77.70
I005	FORK	CUTLERY	7	10.20
I006	MAGAZINE	BOOK	25	30.10
I007	SALAB FORK	CUTLERY	12	30.00
I008	KEYBOARD	HARDWARE	12	120.60
I009	NOVEL	BOOK	20	120.30
I010	MOUSE	HARDWARE	30	19.80
I011	BUTTER KNIFE	CUTLERY	24	13.00

4. Find an Inventory

Find the inventory details based on inventory code or inventory name.

First, select either want to find an inventory based on inventory code or inventory name

```
Do you want to find the inventory details based on
[1] Inventory Code
[2] Inventory Name
[3] Exit
Option:
```

If choose to find an inventory based on inventory code [example: I001]

```
--Find the Inventory--

Please enter the value of Inventory Code: I001

---Inventory Detail---
Inventory Code: I001
Inventory Name: LAPTOP
Inventory Type: HARDWARE
Quantity:      20
Price:         23.00
```

If choose to find an inventory based on inventory name [example: novel]

```
--Find the Inventory--

Please enter the value of Inventory Name: novel

---Inventory Detail---
Inventory Code: I009
Inventory Name: NOVEL
Inventory Type: BOOK
Quantity:      20
Price:         120.30
```

5. Display Inventory List based on Category
Find the inventories that have the same category.

Enter the inventory type and the list of that type will be displayed.

Example value entered: BOOK

```
Pls enter the category you wish to find: BOOK
```

```
::::::::::INVENTORY LIST IN CATEGORY BOOK::::::::::
```

Inventory Code	Inventory Name	Inventory Type	Quantity	Price
I002	STORY BOOK	BOOK	9	100.00
I003	NEWSPAPER	BOOK	130	1.40
I006	MAGAZINE	BOOK	25	30.10
I009	NOVEL	BOOK	20	120.30

Error message if the category entered is invalid:

```
There is no such category in inventory list. Pls try again
```

6. Display Full Inventory List
Display of the information in the Inventory Management System

Inventory Code	Inventory Name	Inventory Type	Quantity	Price
I002	STORY BOOK	BOOK	9	100.00
I009	NOVEL	BOOK	20	120.30
I010	MOUSE	HARDWARE	30	19.80
I004	MOUSE PAD	HARDWARE	2	77.70
I008	KEYBOARD	HARDWARE	12	120.60
I003	NEWSPAPER	BOOK	130	1.40
I005	FORK	CUTLERY	7	10.20
I011	BUTTER KNIFE	CUTLERY	24	13.00
I006	MAGAZINE	BOOK	25	30.10
I007	SALAB FORK	CUTLERY	12	30.00
I013	SPOON	CUTLERY	23	2.30
I023	SDS	E D	5	7.00

7. Update an Inventory

Update the part of inventory either inventory name, inventory type, quantity or price by using their inventory code to define them.

Enter the inventory code of the inventory that desired to update

```
-----Update Inventory Info-----  
Enter the Inventory Code: I002
```

The details of the inventory will be displayed. Check it.

```
---Inventory Detail---  
Inventory Code: I002  
Inventory Name: STORY BOOK  
Inventory Type: BOOK  
Quantity:      9  
Price:         100
```

Select the part that would like to update

```
Choose the part you would like to update  
[1] Inventory Name  
[2] Inventory Type  
[3] Quantity  
[4] Price  
[5] Exit
```

Enter a new value for the part (example: [1] Inventory Name)

```
Enter a new inventory name: NOVEL|
```

8. Check Low Stock Inventory

Display the information of the inventory with quantity less than 10.

```

          ::::::::::Low Stock Inventory::::::::
***Please remember to buy the new inventory!!!***
-----
Inventory Code   Inventory Name   Inventory Type   Quantity   Price
-----
I002             STORY BOOK       BOOK             9          100.00
I004             MOUSE PAD        HARDWARE         2          77.70
I005             FORK             CUTLERY          7          10.20
I023             SDS              E D              5          7.00

```

9. Exit

Exit and terminate the program.

7. Conclusion

To sum up, a queue which implements a circular linked list is a robust and efficient solution for inventory management systems as there are no size restrictions for pointer-based implementations. Also, the queue has restricted the position to the front and back, only the end position can be accessed. This makes the insertion, deletion operation more efficient.

Compared to our assignment 2 system, we have added more features for our system. Firstly, this system enables users to get the list of inventory by category. Secondly, we also provide a function to update the inventory details so that the workers can update the inventory details such as price and quantity when they change. Thirdly, this system allows workers to check the low stock too. If the stock of inventory is less than 10, it will be displayed. Lastly, we also add more constraints to ensure our data integrity and consistency such as restricting duplication of name, code, quantity cannot be less than 0 and code format must be in IXXX where X should be digit.

It is possible to conclude that this program is more complete, integrity and efficient than the previous program in Assignment 2.

DSA Project

ORIGINALITY REPORT

1 %

SIMILARITY INDEX

1 %

INTERNET SOURCES

0 %

PUBLICATIONS

0 %

STUDENT PAPERS

PRIMARY SOURCES

1

www.coursehero.com

Internet Source

1 %

Exclude quotes Off

Exclude bibliography Off

Exclude matches Off