FACULTY OF COMPUTING
UTM Johor Bahru

SESSION 2023/2024 SEMESTER 1

SECJ - DATA STRUCTURE AND ALGORITHMS

ASSIGNMENT 2

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

1. Objective

The assignment 2 is designed to implement the linked list concept into the inventory management system. In this system, we get initial inventory information from a file, and keep it in the linked list data structure, and finally store the modified linked list into file.

To use this system, user can modify the inventory information by

-adding new item into inventory

-delete item from inventory list

-search items in inventory list

-sort the inventory list

-display the entire inventory list

2. Synopsis

3. System Design

3.1. Pseudocode

3.1.1. Main function

3.1.2. Adding new inventory function

3.1.3. Delete inventory function

1. Start
2. Print main menu
3. Get mainMenuChoice
4. If (mainMenuChoice == 2)
    - 4.1. If list is empty
        - 4.1.1. Print “Error: The inventory list is empty. Cannot delete any inventory anymore”
        - 4.1.2. Go to step 2
    - 4.2. Print delete menu
    - 4.3. Get delete option
    - 4.4. If delete option == 1 //delete front
        - 4.4.1. Set temp=head;
        - 4.4.2. head = head->next;

4.4.3. delete temp;

4.5. Else if delete option ==2 //delete middle

4.5.1. Get the position of inventory node user wish to delete (delPos)

4.5.2. If (delPos==1)

4.5.2.1. Go to step 4.4.1

4.5.3. Else

4.5.3.1. temp = head

4.5.3.2. for(i=0; i<(pos-1)&&temp;i++)

4.5.3.2.1. temp=temp->next;

4.5.3.3. if(!temp)

4.5.3.3.1. Print “Error: Your entered position of node that to be deleted is out of the range of current list.”

4.5.3.3.2. Go to step 4.8

4.5.3.4. Else if (temp->next->next==NULL) //delete back

4.5.3.4.1. delete temp->next;

4.5.3.4.2. temp->next=NULL;

4.5.3.5. Else

4.5.3.5.1. prev=temp;

4.5.3.5.2. temp=temp->next

4.5.3.5.3. prev-next=temp->next

4.5.3.5.4. delete temp

4.6. Else if delete option ==3 //delete end

4.6.1. temp=head;

4.6.2. while(temp->next->next != NULL)

4.6.2.1. temp=temp->next

4.6.3. delete temp->next

4.6.4. temp->next=NULL

4.7. Else

4.7.1. Go to step 2

4.8. Print the inventory list after delete operation

4.9. Go to step 2 //End

3.1.4. Sort the inventory list function

5. Start
6. Print main menu
7. Get mainMenuChoice
8. If (mainMenuChoice == 3)

8.1. If (head == NULL || head->next == NULL)
    8.1.1. Go to step 2
8.2. Set sortedList = NULL
8.3. Set curr = head
8.4. While (curr)
    8.4.1. Set next = curr->next
    8.4.2. If (sortedList == NULL || curr->getCode() < sortedList->getCode())
        8.4.2.1. curr->next = sortedList
        8.4.2.2. sortedList = curr
    8.4.3. Else
        8.4.3.1. Set temp = sortedList
        8.4.3.2. While (temp->next != NULL && temp->next->getCode < curr->getCode())
            8.4.3.2.1. temp = temp->next
        8.4.3.3. curr->next = temp->next
        8.4.3.4. temp->next = curr
    8.4.4. curr = next
8.5. head = sortedList
8.6. Print "Inventory List is sorted by Inventory Code in Ascending"
8.7. Display sorted list
8.8. End

3.1.5. Find an inventory based on search key function

1. Start
2. Print main menu
3. Get mainMenuChoice
4. If (mainMenuChoice == 4)
    4.1. Set temp = head
    4.2. Set found = false
    4.3. Print finding user menu
    4.4. Get menuChoice
    4.5. If (menuChoice == 1)
        4.5.1. Print finding based on inventory code page
        4.5.2. Get user input inventoryCode
        4.5.3. Convert the input inventoryCode to uppercase
        4.5.4. While (!found && temp->next != NULL)
            4.5.4.1. If (temp->getCode() == inventoryCode)
                4.5.4.1.1. found = true

4.5.4.1.2. Print the inventory detail

4.5.4.2. temp = temp->next

4.5.5. Go to step 2

4.6. Else if(menuChoice == 2)

4.6.1. Print finding based on inventory name page

4.6.2. Get user input inventoryName

4.6.3. Convert the input inventoryName to uppercase

4.6.4. While (!found && temp->next != NULL)

4.6.4.1. If (temp->getName() == inventoryName)

4.6.4.1.1. found = true

4.6.4.1.2. Print the inventory detail

4.6.4.2. temp = temp->next

4.6.5. Go to step 2

4.7. Else

4.7.1. Go to step 2

4.8. If(!found)

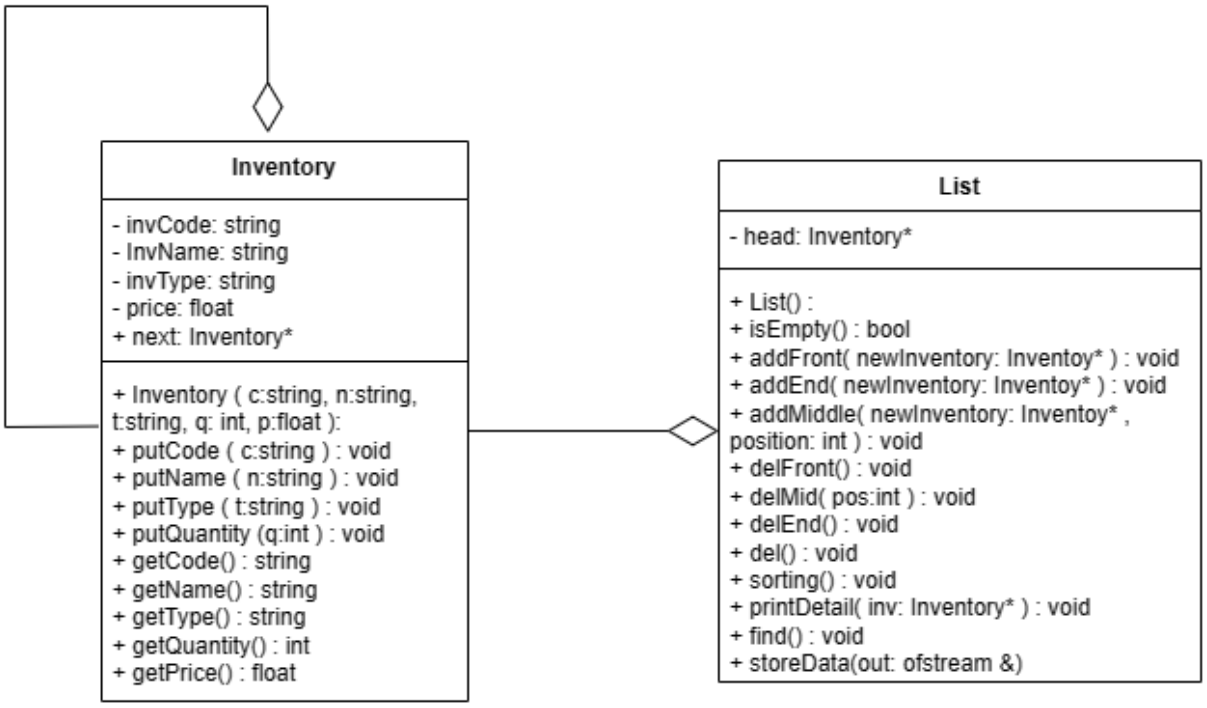
4.8.1. Print “The entered value is invalid!”

5. End

3.1.6. Display inventory list function

3.2. Flow Chart

3.3. Class Diagram



4. Data Structure Operation Description

4.1. Linked List Implementation

In this case, we declare a class named “Inventory” which acts as a node and a class named “List” to store nodes. Both of these classes will function as a linked list.

In tha main function, we declare an List object named InvList which purpose is to store all the inventory nodes. At first, we retrieve data from “input.txt” file and store it into the invList by using addEnd() function where it will link the new node (inventory) at the end of the linked list (InvList).

In this program, we have 5 functions and the first function is adding new inventory where users can choose to add inventory in front, in the specific position or at the end of the InvList.

Adding Function

addFront()  	• If the list is empty, set head = newInventory • Else, newInventory->next = head->next and head = newInventory
addEnd()  	• If the list is empty, set head = newInventory • Else, loop until find the last node in the linked list, set newInventory->next = NULL and insert the newInventory after the last node
addMiddle()  	• If the list is empty, set head = newInventory • If (position == 1), addFront() • while(NO achieve the last node in the list AND count < position-1 AND NO found), temp will become its next value and increase count by 1 • Then, the newInventory will be inserted at the position • If the entered position value is invalid, it will auto inserted at the end of list

The second function is deleting inventory function where users can choose to delete inventory in front, in the specific position or at the end of the InvList.

Deleting Function

()	
()	•
()	•

The third function is sorting where it will sort the invList based on Inventory Code in ascending order.

If the list is empty, it will quit the sorting

```
    if (head == NULL || head->next == NULL)
    {
        return;
    }
```

Else, it will compare the inventory code value from sortedList and curr

- If the inventory code from curr is smaller than inventory code from sorted list, inventory code from curr will be stored in front of inventory code from sorted list
- Else, set a temp = sortedList and start looping with condition (NO achieve the end of list AND temp->next inventory Code value is smaller than the inventory code from curr), temp will go to next value. Until out of looping, store the inventory code from curr at the next of the inventory code of temp
- Then, move the curr to be curr->next value
- And lastly update the head with sortedList

```
    Inventory *sortedList = NULL;
    Inventory *curr = head;

    while (curr)
    {
        Inventory *next = curr->next;

        if (sortedList == NULL || curr->getCode() < sortedList->getCode())
        {
            curr->next = sortedList;
            sortedList = curr;
        }
        else
        {
            Inventory *temp = sortedList;
            while (temp->next != NULL && temp->next->getCode() < curr->getCode())
            {
                temp = temp->next;
            }
            curr->next = temp->next;
            temp->next = curr;
        }
        curr = next;
    }

    head = sortedList;
```

Finally, display the sorted InvList

```
    cout << "Inventory List is sorted by Inventory Code in Ascending\n";
    displayList();
}
```

The fourth function is a finding function where users can find the inventory based on inventory code or inventory name.

It will loop the list since it has not achieved the last node in the list and show the details of the inventory if the value of temp->getCode() is the same as the sKey that is input by users.

```
if (fChoice == 1)
{
    cout << "Inventory Code: ";
    cin >> sKey;
    sKey = changeToUpper(sKey);
    while (temp->next != NULL && found == false)
    {
        if (temp->getCode() == sKey)
        {
            found = true;
            printDetail(temp);
        }
        temp = temp->next;
    }
}
```

The fifth function is displayList function which will display all the inventory node in the InvList linked list.

It will loop the list since it has not achieved the last node in the list and display the data by calling the accessor of the inventory node.

```
Inventory *temp = head;
while (temp != NULL)
{
    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;
}
```

5. Conclusion

In conclusion, a pointer linked list is a useful method to store data. It is flexible without limited size as an array so that we don't waste the memory space. When we apply a pointer link list concept in our system, it provides more freedom to users in managing the data in the Inventory Management System.

In our system, users can choose to add new inventory in front, at the specific position or at the end of the inventory list. It is going the same as deleting the inventory. Besides that, it also helps to sort the list based on the inventory code in ascending order to provide an easier view to users. At the same time, it also allows users to search certain inventory based on their inventory code or inventory name because both of these values are unique. Lastly, it also can print the existing inventory in the inventory list.

This program can be concluded that is more integrity, efficiency and usability than the previous program in Assignment 1.