UNIVERSITI TEKNOLOGI MALAYSIA

Department of Computer Science
Faculty of Computing

DATA STRUCTURE AND ALGORITHMS

ASSIGNMENT 22

Programme	:	Bachelor of Computer Science (*Data Engineering*)
Subject Code	:	SECJ2013
Session-Sem	:	2023/2024-1

Prepared by :
1) LOO JIA CHANG (A22EC0074)
2) GOH JING YANG (A22EC0052)
3) LOW JIE SHENG (A22EC0075)

Section : 02

Group : Nothing

Topic : Inventory Management System

Lecturer : Dr. LIZAWATI BINTI MI YUSUF

Date : 31/12/2023

TABLE OF CONTENTS

1.0 Objective

The primary goal of the provided C++ code is establishing an interactive inventory management system through the utilization of linked lists. The system's engineers made it so that users may do a cluster of capacities, such as including and evacuating item, looking, sorting and showing the inventory. This class serves as a template for every inventory item and contains basic details like the object's number, name, description, quantity, cost, and location. The basic usage makes utilize of the essential connected list information structure, in which every node speaks to a thing within the stock and can be alluded to by the following hub. The system's breadth of its operations; users can include, remove, search for and arrange objects according to a few criteria. The three sorting strategies sorting by thing number, by item name, and by item area gives users with adaptability in organizing stock information custom fitted to their inclinations.

Representation of Inventory Items:

The code for Inventory class represents each item in the inventory. This class encapsulates essential attributes such as item number, item name, description, quantity, cost, and item location.

Linked List Implementation:

The inventory using a linked list, a basic data structure. A pointer to the next node in the sequence be stored in each node in the list, which represents each item in the inventory.

Comprehensive Functionalities:

The system is a comprehensive set of basic, including insertion, deletion, searching sorting and display inventory items. Users can add items at different positions, delete and search for specific items.

Sorting:

Three sorting functions implement in code sorting by item number, item name and item location. These function provide users can options organize the inventory data according to their preferences.

2.0 Synopsis

The Inventory Management System is a complete solution designed to make the organizing and processing of inventory data faster. It evolves using linked lists and C++. Linked lists, a dynamic data structure for arranging and obtaining inventory items, are the basis of this system.

Inventory Management:

1. Item Number

- A unique identifier is assigned to each item store in the inventory.

2.Item Name

- The item's name offers a legible and easily recognizable identity.

3.Description of item

- Description of the item provides additional details which can help in understanding the product.

4.Quantity of item

- The quantity of each item stored in the inventory, indicating the stock level and help in inventory control.

5.Cost

- The value of each item in terms of money enables effective financial management and facilitates the analysis of costs.

6. Location at inventory

- The location or storage position of the item within the store

Linked List Implementation:

The basic data structure used to manage the inventory is a linked list. Each node in this structure represents an inventory item, holding a pointer to the following node in the sequence as well as an instance of the Inventory class. Efficient addition, deletion, and traversal of inventory items are made possible by this dynamic link.

Comprehensive Functionalities:

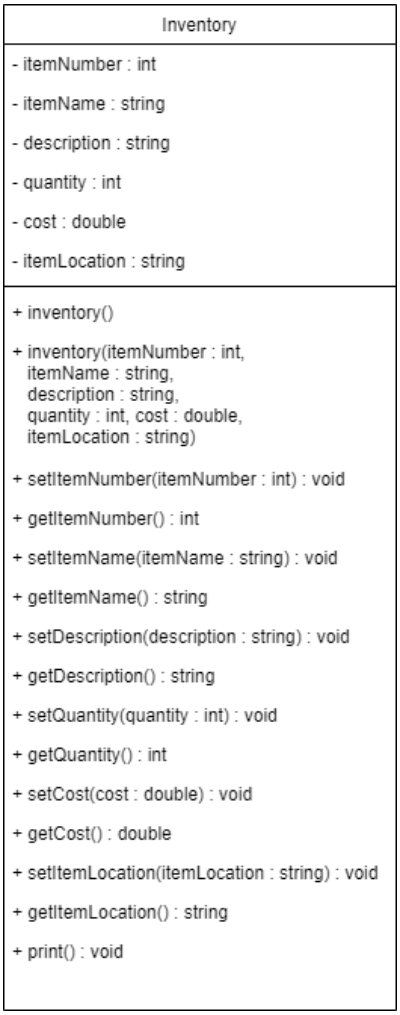
The system provides an array of functions to meet various inventory management requirements. Users may easily to add items in the inventory at different locations, remove items, search for

specific items using different parameters (such as number, name, or location), and can view the inventory.

Sorting Mechanisms:

Three type sorting mechanisms have been incorporated with the system: item number, item name, and item location. With the use of these sorting options, users can structure the inventory data in an arrangement that best matches the user's need.

3.0 Class Diagram


Inventory
- itemNumber : int
- itemName : string
- description : string
- quantity : int
- cost : double
- itemLocation : string
+ inventory()
+ inventory(itemNumber : int,
itemName : string,
description : string,
quantity : int, cost : double,
itemLocation : string)
+ setItemNumber(itemNumber : int) : void
+ getItemNumber() : int
+ setItemName(itemName : string) : void
+ getItemName() : string
+ setDescription(description : string) : void
+ getDescription() : string
+ setQuantity(quantity : int) : void
+ getQuantity() : int
+ setCost(cost : double) : void
+ getCost() : double
+ setItemLocation(itemLocation : string) : void
+ getItemLocation() : string
+ print() : void

4.0 Pseudo Code

```
Main and other function:

Function main()
1. Initialize `List list`, `string inp`, `int choice`, `inventory* item`
2. Clear the console
3. Print "Welcome to the inventory management system!"
4. Delay for 500 milliseconds
5. Clear the console
6. Start an infinite loop
   1. Start an inner infinite loop
      1. Display menu options
      2. Take user input
      3. If input is a number
         1. Convert input to integer
         2. If input length is 1 and choice is between 1 and 6
            1. Clear the console
            2. Break the inner loop
         3. If not, print error message, pause and clear the console
   2. If choice is 1 (Add item)
      1. Start an inner loop for adding item
         1. Display add item options
         2. Take user input
         3. If input is a number
            1. Convert input to integer
            2. If input length is 1 and choice is between 1 and 4
               1. Clear the console
               2. Break the inner loop
```

3. If not, print error message, pause and clear the console

2. If choice is 4, continue to the next iteration of the outer loop

3. Take inputs for item details

4. Create a new inventory item

5. If choice is 1, 2, or 3, insert the item at the appropriate position

6. Print "Item added successfully!" and clear the console

3. If choice is 2 (Delete item)

1. If list is empty, print message, pause, clear console, and continue to next iteration

2. If not, start an inner loop for deleting item

1. Display delete item options

2. Take user input

3. If input is a number

1. Convert input to integer

2. If input length is 1 and choice is between 1 and 4

1. Clear the console

2. Break the inner loop

3. If not, print error message, pause and clear the console

3. If choice is 4, continue to the next iteration of the outer loop

4. If choice is 1, 2, or 3, delete the item at the appropriate position and print success message

4. If choice is 3 (Search item)

1. If list is empty, print message, pause, clear console, and continue to next iteration

2. If not, start an inner loop for searching item

1. Display search item options

2. Take user input

3. If input is a number

1. Convert input to integer

2. If input length is 1 and choice is between 1 and 4

1. Clear the console

2. Break the inner loop

3. If not, print error message, pause and clear the console

3. If choice is 4, continue to the next iteration of the outer loop

4. If choice is 1, 2, or 3, search the item by the appropriate field and print the item details

5. If choice is 4 (Sort items)

1. If list is empty, print message, pause, clear console, and continue to next iteration

2. If not, start an inner loop for sorting items

1. Display sort item options

2. Take user input

3. If input is a number

1. Convert input to integer

2. If input length is 1 and choice is between 1 and 4

1. Clear the console

2. Break the inner loop

3. If not, print error message, pause and clear the console

3. If choice is 4, continue to the next iteration of the outer loop

4. If choice is 1, 2, or 3, sort the list by the appropriate field and print success message

6. If choice is 5 (Display all items)

1. If list is empty, print message, pause, clear console, and continue to next iteration

2. If not, print all items in the list

   7. If choice is 6, break the outer loop
7. Print "Thank you for using the inventory management system!"
8. Sleep for 1 second
9. Return 0

Function `isEmpty`

1. Return whether `head` is `NULL`

Function insertNode(inventory item)

1. Create a new Node with the passed item.
2. If the list is empty:
   2.1 Set the new node as the head of the list.
3. Else:
   3.1 Find the appropriate position to insert the new node.
   3.2 Insert the new node at the identified position.
4. Update the list to reflect the addition of the new node.
5. End If
6. End function

Function findNodeNum(int itemNumber)

1. Start with the head of the list.

2. Initialize a variable currIndex to keep track of the current index in the list.

3. While the current node is not null and the itemNumber of the current node's item does not match the given itemNumber:

    3.1 Move to the next node.

    3.2 Increment currIndex.

4. If a node with the matching itemNumber is found, return the current node.

5. If no matching node is found, return an indication of "not found".

6. End If

7. End function

Function (findNodeName and findNodeLocation) have the similar algorithm like the findNodenum

Function deleteNode(int index)

1. If index is less than 0, return immediately (invalid index).

2. Start with the head of the list and initialize a variable currIndex to 1.

3. While the current node is not null and index is greater than currIndex:

    3.1 Keep track of the current node and its predecessor.

    3.2 Move to the next node.

    3.3 Increment currIndex.

4. If the node at the given index is found:

   4.1 Remove the node from the list.

   4.2 Adjust the links of the predecessor and the following node.

5. If the node at the given index is not found, indicate an unsuccessful operation.

6. End If

7. End function

Function `displayList`:

1. Initialize `currNode` to `head`

2. While `currNode` is not `NULL`, print the item of `currNode` and move `currNode` to the next node

Function SortbyNumber()

1. If the head of the list is null, return immediately (empty list).

2. Initialize a boolean swapped to keep track of whether any swaps have been made.

3. Perform a bubble sort: for each node in the list (except the last one):

3.1 Set swapped to false.

3.2 Traverse the list up to the unsorted part.

3.3 If the itemNumber of the current node is greater than that of the next node:

3.3.1 Swap the items of the current node and the next node.

3.3.2 Set swapped to true.

4. If no swaps are made in an entire pass, the list is sorted and the function can return.

5. End If

6. End function

Function SortbyLocation()

1. If the head of the list is null, return immediately (empty list).

2. Initialize a boolean swapped to keep track of whether any swaps have been made.

3. Use a bubble sort algorithm:

  3.1 For each node in the list, compare the itemLocation of the current node with the next node.

   3.2 If the current node's location is greater, swap the items.

  3.3 Continue until no swaps are needed, indicating the list is sorted.

4. End If

5. End function

Function isfloat(string s)

1. If s is empty, return false (an empty string is not a valid float).

2. If s contains any character that is not a digit, a decimal point, or a sign indicator (plus or minus sign) at the beginning, return false.

3. If s contains more than one decimal point, return false.

4. Otherwise, return true (indicating the string is a valid representation of a float).

5. End If

6. End function

Function isNumber(string s)

1. If s is empty, return false (an empty string is not a valid number).

2. For each character in s:

   2.1 If the character is not a digit, return false.

3. If all characters are digits, return true (indicating the string is a valid representation of a number).

4. End If

5. End function

5.0 Description of how data structure operations

Insertion Operation:

The insertion of nodes into the linked list is handled by the List class's insertNode function. Items can be added to the inventory at any point, starting, stopping, or at a specified location. To keep the linked list connected, a new node must be created, its item data must be set, and pointers must be appropriately adjusted.

Deletion Operation:

To handle removing nodes from the linked list, use the deleteNode function in the List class. Items can be removed from the start, the finish, or a particular location by users. In order to avoid deleting the node, pointers must be updated. After that, the memory that the node was using is released.

Search Operation:

The List class implements three distinct search functions (findNodeNum, findNodeName, and findNodeLocation) to search for items by name, number, or location. Until a match is found, these operations iterate through the linked list, comparing the search criteria with the characteristics of each node's inventory item.

Sorting Operations:

Sorting algorithms based on number, name, and location are implemented by the SortbyNumber, SortbyName, and SortbyLocation functions in the List class, respectively. These sorting operations reorganize the linked list's nodes according to the given criteria by using algorithms.

Display Operation:

Every inventory item's details are displayed by the displayList method of the List class, which iterates through the linked list. A comprehensive view of the entire inventory is given to users by this operation.