Q1: Application of Stack

1. Reversing a String using Stack

Code:

```
def reverse_string(input_string):
    stack = []
    for char in input_string:
        stack.append(char)
    reversed_string = ''
    while stack:
        reversed_string += stack.pop()
    return reversed_string
```

2. Balancing the brackets

Code:

```
def is_balanced(expression):
    stack = []
    opening = "({["
    closing = ")}]"
    match = {')': '(', '}': '{', ']': '['}

    for char in expression:
        if char in opening:
            stack.append(char)
        elif char in closing:
            if not stack or stack[-1] != match[char]:
                return False
            stack.pop()
    return len(stack) == 0
```

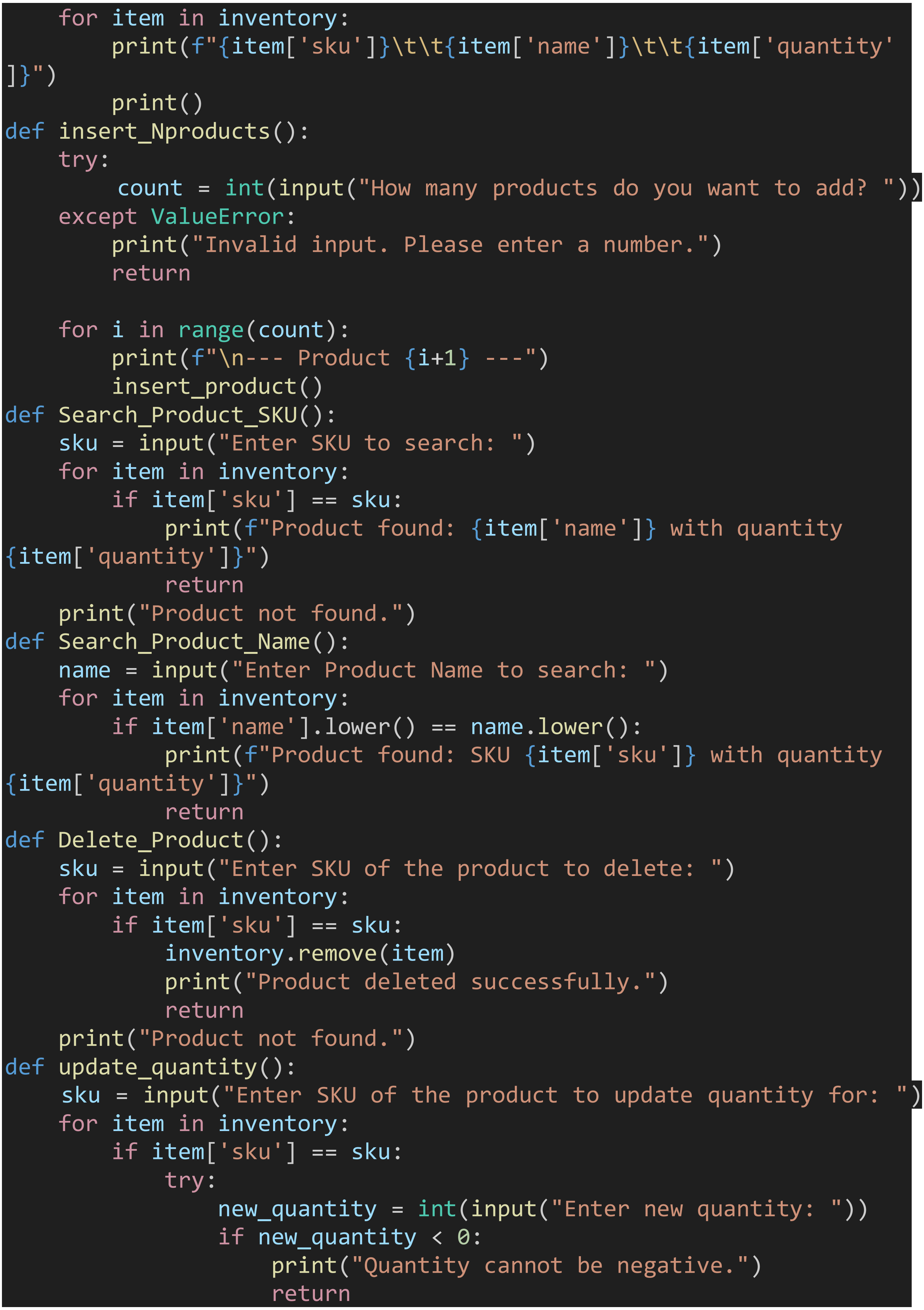
3. Undo Operation

```
def undo_operations(stack):
    stack.append(1)
    stack.append(2)
    stack.append(3)
    stack.append(4)
    print("initial stack:", stack)
    stack.pop()  # Undo last operation (4)
    print("After undoing last operation:", stack)
```

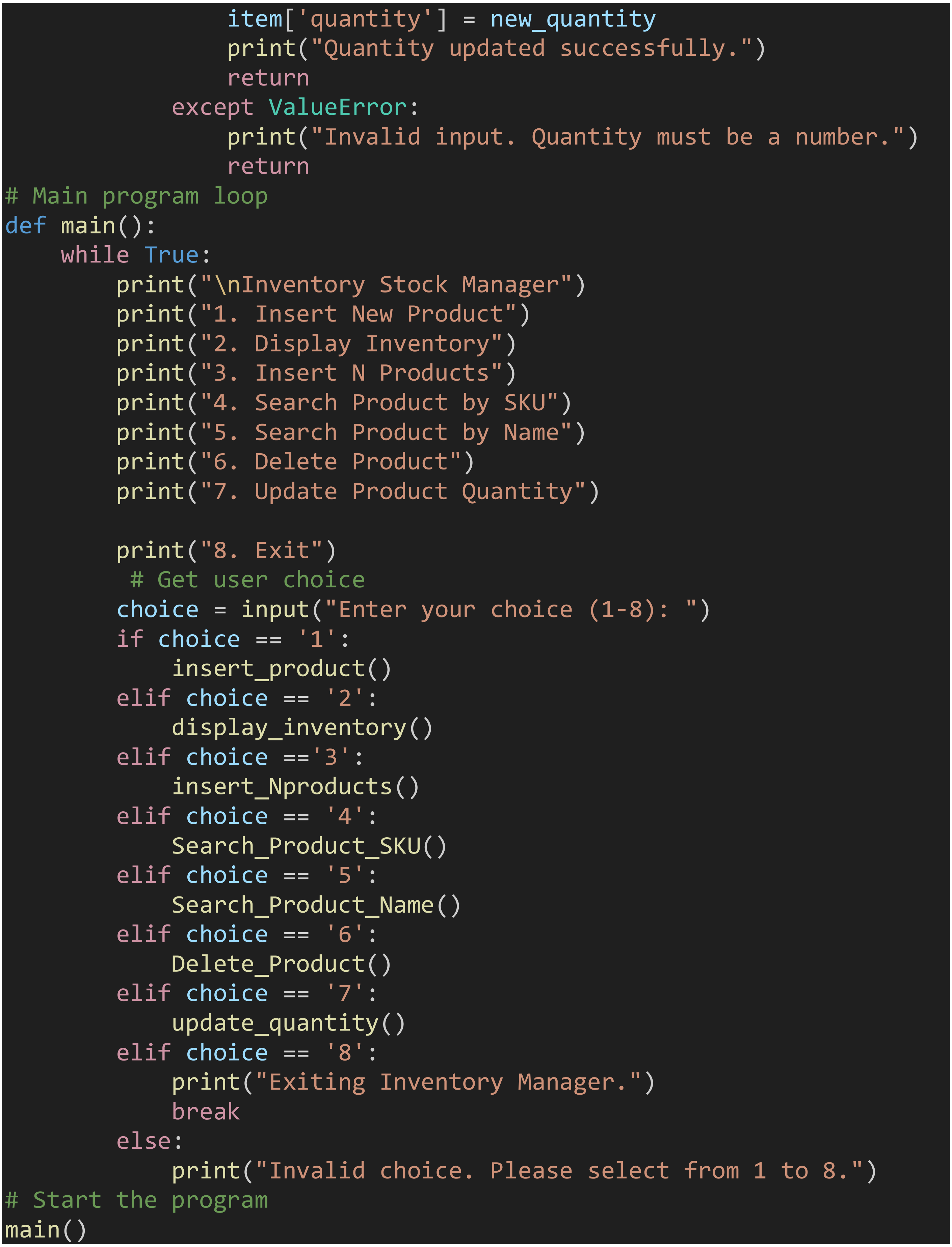
Q2 Inventory (Inserting products, Searching, Display, Update, Delete)

Code:

```
inventory = []
MAX_PRODUCTS = 100  # Maximum number of products allowed in
inventory
# Function to insert a new product
def insert_product():
    if len(inventory) >= MAX_PRODUCTS:
        print("Inventory is full. Cannot insert more products.
current products:", len(inventory))
        return
    sku = input("Enter SKU: ")

    # Check for duplicate SKU
    for item in inventory:
        if item['sku'] == sku:
            print("Product with this SKU already exists!")
            return
    name = input("Enter Product Name: ")
    if not name.strip():
        print("Invalid input. Product name cannot be empty.")
        return

    try:
        quantity = int(input("Enter Quantity: "))
    except ValueError:
        print("Invalid input. Quantity must be a number.")
        return
    if quantity < 0:
        print("Quantity cannot be negative.")
        return

 # Create product dictionary and add to inventory
    product = {'sku': sku, 'name': name, 'quantity': quantity}
    inventory.append(product)
    print("Product inserted successfully.")
# Function to display inventory
def display_inventory():
    if not inventory:
        print("Inventory is empty.")
        return
    print("\nCurrent Inventory:")
    print("SKU\t\tProduct Name\t\tQuantity")
    print("-----------------------------------------------")
```



```
    for item in inventory:
        print(f"{item['sku']}\t\t{item['name']}\t\t{item['quantity'
]}")
        print()
def insert_Nproducts():
    try:
        count = int(input("How many products do you want to add? "))
    except ValueError:
        print("Invalid input. Please enter a number.")
        return

    for i in range(count):
        print(f"\n--- Product {i+1} ---")
        insert_product()
def Search_Product_SKU():
    sku = input("Enter SKU to search: ")
    for item in inventory:
        if item['sku'] == sku:
            print(f"Product found: {item['name']} with quantity
{item['quantity']}")
            return
    print("Product not found.")
def Search_Product_Name():
    name = input("Enter Product Name to search: ")
    for item in inventory:
        if item['name'].lower() == name.lower():
            print(f"Product found: SKU {item['sku']} with quantity
{item['quantity']}")
            return
def Delete_Product():
    sku = input("Enter SKU of the product to delete: ")
    for item in inventory:
        if item['sku'] == sku:
            inventory.remove(item)
            print("Product deleted successfully.")
            return
    print("Product not found.")
def update_quantity():
    sku = input("Enter SKU of the product to update quantity for: ")
    for item in inventory:
        if item['sku'] == sku:
            try:
                new_quantity = int(input("Enter new quantity: "))
                if new_quantity < 0:
                    print("Quantity cannot be negative.")
                    return
```

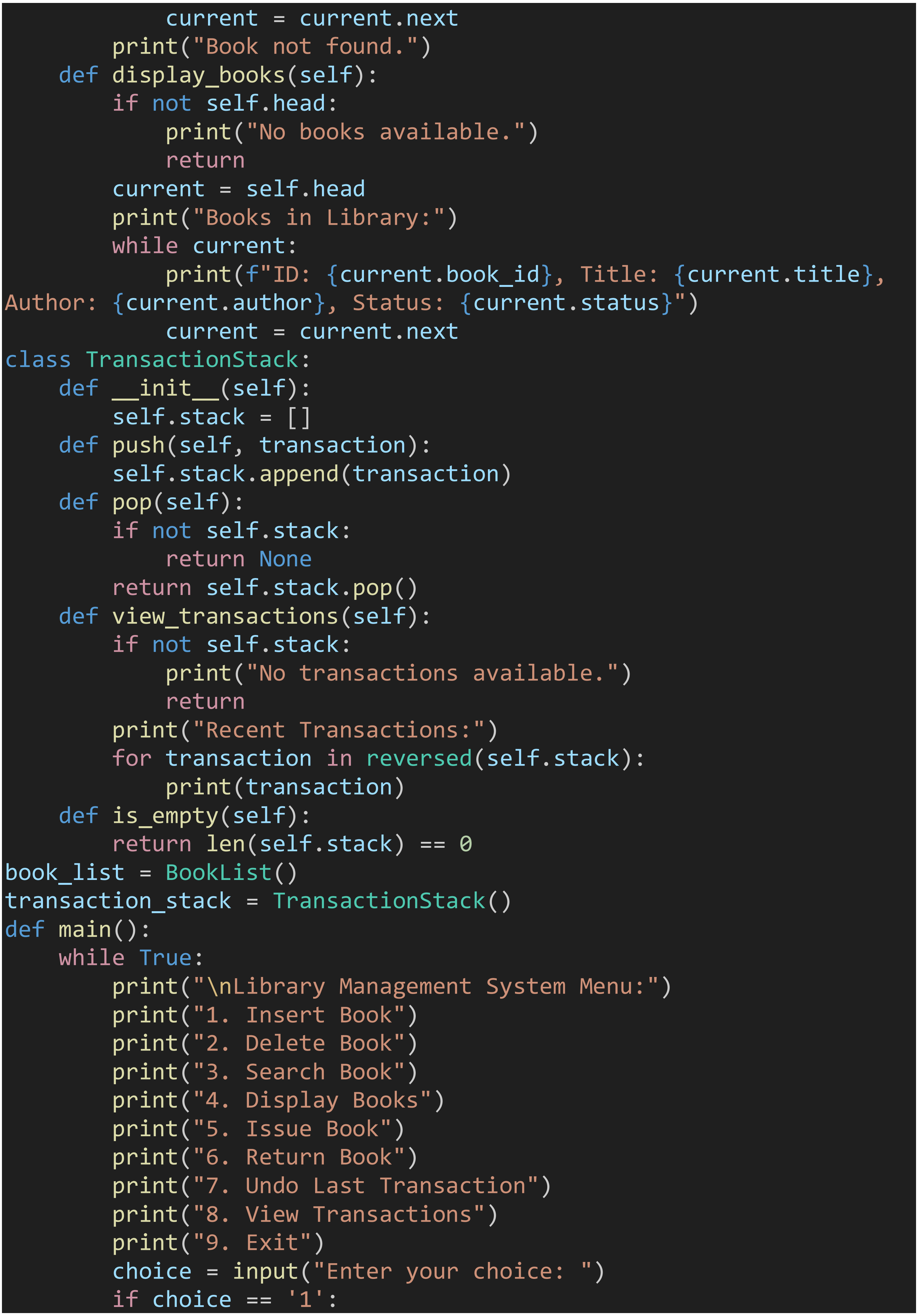


```
                item['quantity'] = new_quantity
                print("Quantity updated successfully.")
                return
            except ValueError:
                print("Invalid input. Quantity must be a number.")
                return
# Main program loop
def main():
    while True:
        print("\nInventory Stock Manager")
        print("1. Insert New Product")
        print("2. Display Inventory")
        print("3. Insert N Products")
        print("4. Search Product by SKU")
        print("5. Search Product by Name")
        print("6. Delete Product")
        print("7. Update Product Quantity")

        print("8. Exit")
         # Get user choice
        choice = input("Enter your choice (1-8): ")
        if choice == '1':
            insert_product()
        elif choice == '2':
            display_inventory()
        elif choice =='3':
            insert_Nproducts()
        elif choice == '4':
            Search_Product_SKU()
        elif choice == '5':
            Search_Product_Name()
        elif choice == '6':
            Delete_Product()
        elif choice == '7':
            update_quantity()
        elif choice == '8':
            print("Exiting Inventory Manager.")
            break
        else:
            print("Invalid choice. Please select from 1 to 8.")
# Start the program
main()
```

Q3 Library

Code:

```
import sys
class BookNode:
    def __init__(self, book_id, title, author, status="Available"):
        self.book_id = book_id
        self.title = title
        self.author = author
        self.status = status
        self.next = None
class BookList:
    def __init__(self):
        self.head = None
    def insert_book(self, book_id, title, author):
        new_book = BookNode(book_id, title, author)
        if not self.head:
            self.head = new_book
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = new_book
        print(f"Book '{title}' inserted.")
    def delete_book(self, book_id):
        current = self.head
        prev = None
        while current and current.book_id != book_id:
            prev = current
            current = current.next
        if not current:
            print("Book not found.")
            return
        if prev:
            prev.next = current.next
        else:
            self.head = current.next
        print(f"Book ID {book_id} deleted.")
    def search_book(self, book_id):
        current = self.head
        while current:
            if current.book_id == book_id:
                print(f"Book Found: ID: {current.book_id}, Title: {current.title}, Author: {current.author}, Status: {current.status}")
                return
```



```
            current = current.next
        print("Book not found.")
    def display_books(self):
        if not self.head:
            print("No books available.")
            return
        current = self.head
        print("Books in Library:")
        while current:
            print(f"ID: {current.book_id}, Title: {current.title}, Author: {current.author}, Status: {current.status}")
            current = current.next
class TransactionStack:
    def __init__(self):
        self.stack = []
    def push(self, transaction):
        self.stack.append(transaction)
    def pop(self):
        if not self.stack:
            return None
        return self.stack.pop()
    def view_transactions(self):
        if not self.stack:
            print("No transactions available.")
            return
        print("Recent Transactions:")
        for transaction in reversed(self.stack):
            print(transaction)
    def is_empty(self):
        return len(self.stack) == 0
book_list = BookList()
transaction_stack = TransactionStack()
def main():
    while True:
        print("\nLibrary Management System Menu:")
        print("1. Insert Book")
        print("2. Delete Book")
        print("3. Search Book")
        print("4. Display Books")
        print("5. Issue Book")
        print("6. Return Book")
        print("7. Undo Last Transaction")
        print("8. View Transactions")
        print("9. Exit")
        choice = input("Enter your choice: ")
        if choice == '1':
            book_id = int(input("Enter Book ID: "))
            title = input("Enter Book Title: ")
            author = input("Enter Author Name: ")
            book_list.insert_book(book_id, title, author)
        elif choice == '2':
            book_id = int(input("Enter Book ID to delete: "))
            book_list.delete_book(book_id)
        elif choice == '3':
            book_id = int(input("Enter Book ID to search: "))
            book_list.search_book(book_id)
        elif choice == '4':
            book_list.display_books()
        elif choice == '5':
            book_id = int(input("Enter Book ID to issue: "))
            current = book_list.head
            while current:
                if current.book_id == book_id:
                    if current.status == "Available":
                        current.status = "Issued"
                        transaction_stack.push(f"Issued Book ID
{book_id}")
                        print(f"Book ID {book_id} issued.")
                    else:
                        print("Book is already issued.")
                    break
                current = current.next
            else:
                print("Book not found.")
        elif choice == '6':
            book_id = int(input("Enter Book ID to return: "))
            current = book_list.head
            while current:
                if current.book_id == book_id:
                    if current.status == "Issued":
                        current.status = "Available"
                        transaction_stack.push(f"Returned Book ID
{book_id}")
                        print(f"Book ID {book_id} returned.")
                    else:
                        print("Book is not issued.")
                    break
                current = current.next
            else:
                print("Book not found.")
        elif choice == '7':
            last_transaction = transaction_stack.pop()
            if not last_transaction:
                print("No transactions to undo.")
            else:
                action, _, book_id_str =
last_transaction.partition(' ')
                book_id = int(book_id_str.split()[-1])
                current = book_list.head
                while current:
                    if current.book_id == book_id:
                        if action == "Issued":
                            current.status = "Available"
                        elif action == "Returned":
                            current.status = "Issued"
                        break
                    current = current.next
                print(f"Undo: {last_transaction}")
        elif choice == '8':
            transaction_stack.view_transactions()
        elif choice == '9':
            sys.exit()
        else:
            print("Invalid choice. Please try again.")
if __name__ == "__main__":
    main()
```

Q4 Insertion in QUEUE

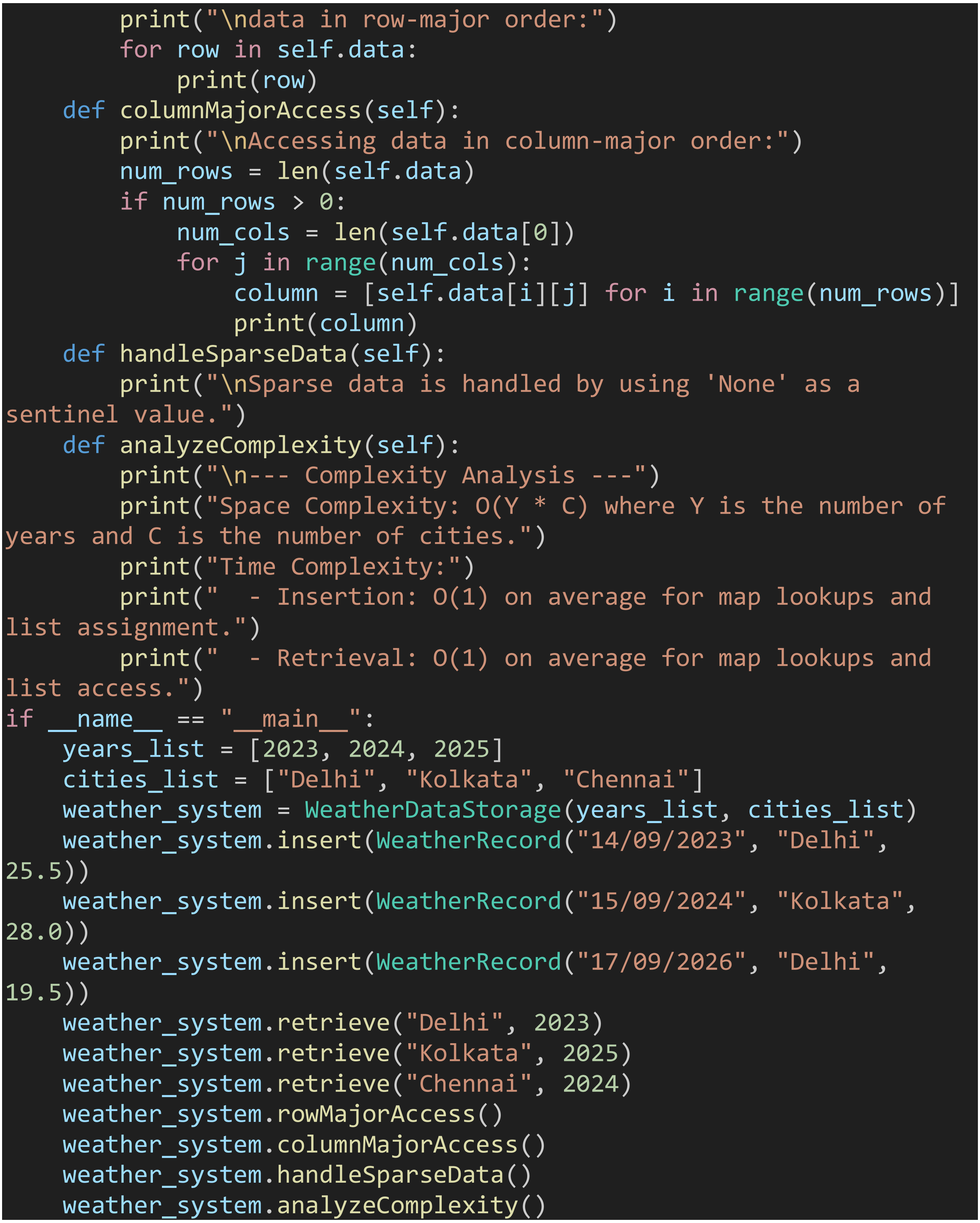
Code:

```
queue = []
front = -1
rear = -1
def enqueue(front,rear):
    element = input("Enter the element to enqueue: ")
    if front == -1 and rear == -1:
        front = 0
        rear = 0
    elif front ==0:
        rear =rear + 1
    queue.append(element)
    print(f"{element} enqueued to queue")
    print(f"Current queue: {queue}")
    print(f"Front index: {front}, Rear index: {rear}")
    return front,rear
n = int(input("Enter number of elements to enqueue: "))
for i in range(0,n):
    enqueue(front,rear)
```

Q5 Whether Record Storage using Abstract Data Type

Code:

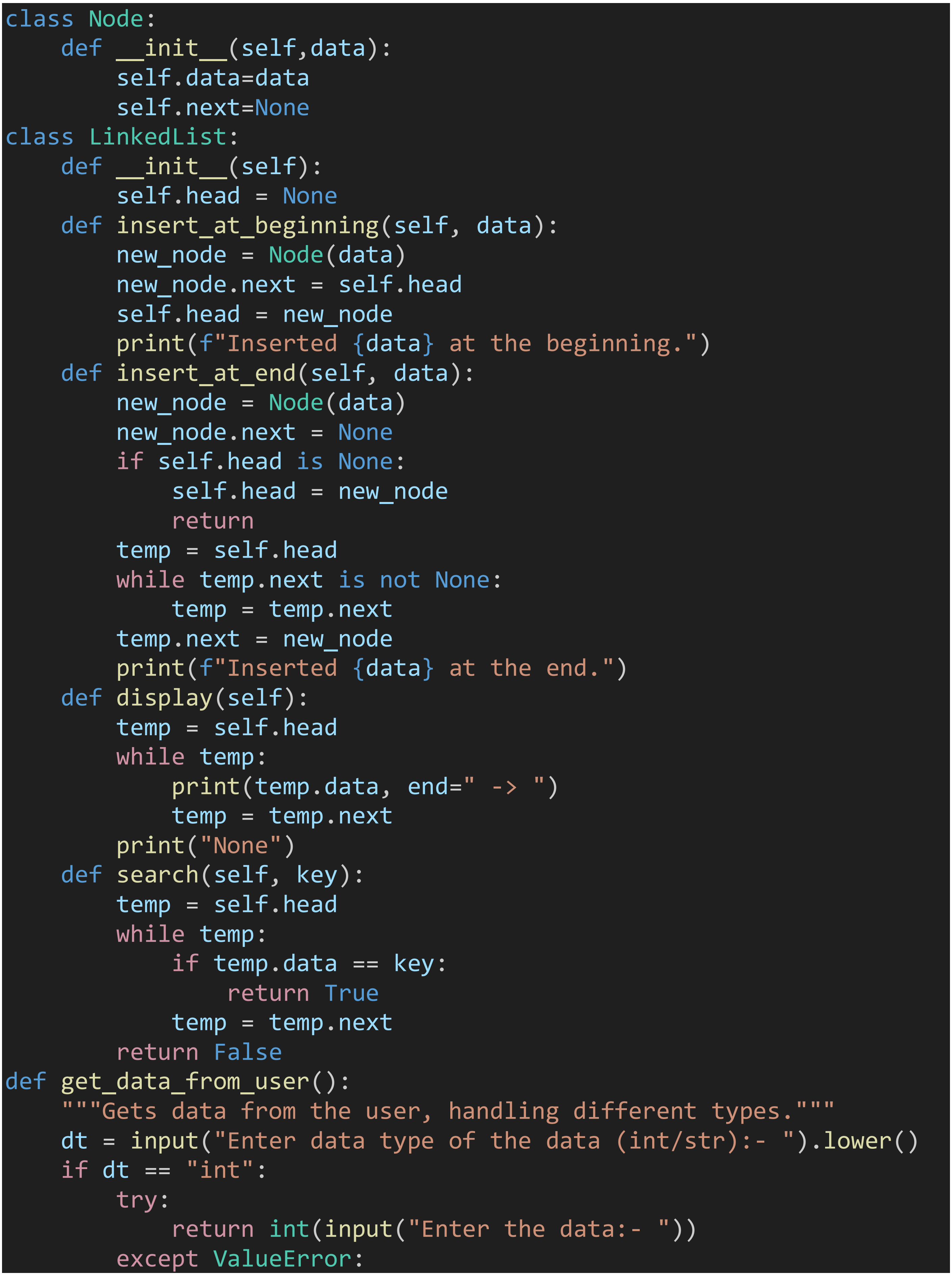
```
class WeatherRecord:
    def _init_(self, date, city, temperature):
        self.date = date
        self.city = city
        self.temperature = temperature
class WeatherDataStorage:
    def _init_(self, years, cities):
        self.data = [[None for _ in cities] for _ in years]
        self.years = {year: i for i, year in enumerate(years)}
        self.cities = {city: i for i, city in enumerate(cities)}
    def insert(self, record):
        try:
            year = int(record.date.split('/')[-1])
            year_index = self.years.get(year)
            city_index = self.cities.get(record.city)
            if year_index is not None and city_index is not None:
                self.data[year_index][city_index] = record.temperature
                print(f"Record for {record.city} on {record.date} inserted successfully.")
            else:
                print("Error: Year or City not found in the storage system.")
        except (ValueError, IndexError):
            print("Error: Invalid date format. Please use day/month/year.")
    def retrieve(self, city, year):
        year_index = self.years.get(year)
        city_index = self.cities.get(city)
        if year_index is not None and city_index is not None:
            temperature = self.data[year_index][city_index]
            if temperature is not None:
                print(f"Temperature for {city} in {year}: {temperature}°C")
                return temperature
            else:
                print(f"No data available for {city} in {year}.")
                return None
        else:
            print("Error: Year or City not found in the storage system.")
            return None
    def rowMajorAccess(self):
```



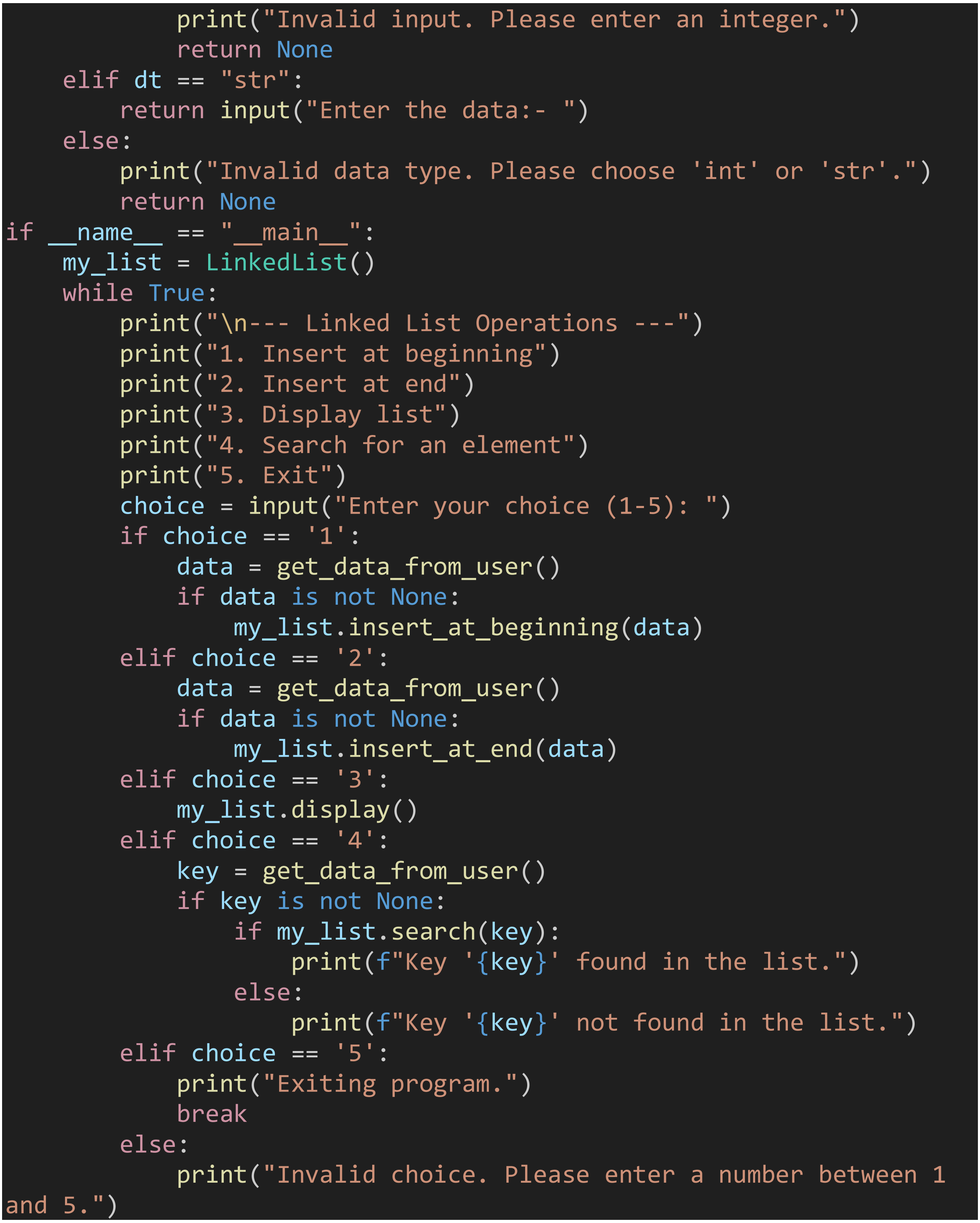
```
        print("\ndata in row-major order:")
        for row in self.data:
            print(row)
    def columnMajorAccess(self):
        print("\nAccessing data in column-major order:")
        num_rows = len(self.data)
        if num_rows > 0:
            num_cols = len(self.data[0])
            for j in range(num_cols):
                column = [self.data[i][j] for i in range(num_rows)]
                print(column)
    def handleSparseData(self):
        print("\nSparse data is handled by using 'None' as a
sentinel value.")
    def analyzeComplexity(self):
        print("\n--- Complexity Analysis ---")
        print("Space Complexity: O(Y * C) where Y is the number of
years and C is the number of cities.")
        print("Time Complexity:")
        print("  - Insertion: O(1) on average for map lookups and
list assignment.")
        print("  - Retrieval: O(1) on average for map lookups and
list access.")
if __name__ == "__main__":
    years_list = [2023, 2024, 2025]
    cities_list = ["Delhi", "Kolkata", "Chennai"]
    weather_system = WeatherDataStorage(years_list, cities_list)
    weather_system.insert(WeatherRecord("14/09/2023", "Delhi",
25.5))
    weather_system.insert(WeatherRecord("15/09/2024", "Kolkata",
28.0))
    weather_system.insert(WeatherRecord("17/09/2026", "Delhi",
19.5))
    weather_system.retrieve("Delhi", 2023)
    weather_system.retrieve("Kolkata", 2025)
    weather_system.retrieve("Chennai", 2024)
    weather_system.rowMajorAccess()
    weather_system.columnMajorAccess()
    weather_system.handleSparseData()
    weather_system.analyzeComplexity()
```

Q6. Node operations

Code:



```
class Node:
    def __init__(self,data):
        self.data=data
        self.next=None
class LinkedList:
    def __init__(self):
        self.head = None
    def insert_at_beginning(self, data):
        new_node = Node(data)
        new_node.next = self.head
        self.head = new_node
        print(f"Inserted {data} at the beginning.")
    def insert_at_end(self, data):
        new_node = Node(data)
        new_node.next = None
        if self.head is None:
            self.head = new_node
            return
        temp = self.head
        while temp.next is not None:
            temp = temp.next
        temp.next = new_node
        print(f"Inserted {data} at the end.")
    def display(self):
        temp = self.head
        while temp:
            print(temp.data, end=" -> ")
            temp = temp.next
        print("None")
    def search(self, key):
        temp = self.head
        while temp:
            if temp.data == key:
                return True
            temp = temp.next
        return False
def get_data_from_user():
    """Gets data from the user, handling different types."""
    dt = input("Enter data type of the data (int/str):- ").lower()
    if dt == "int":
        try:
            return int(input("Enter the data:- "))
        except ValueError:
```

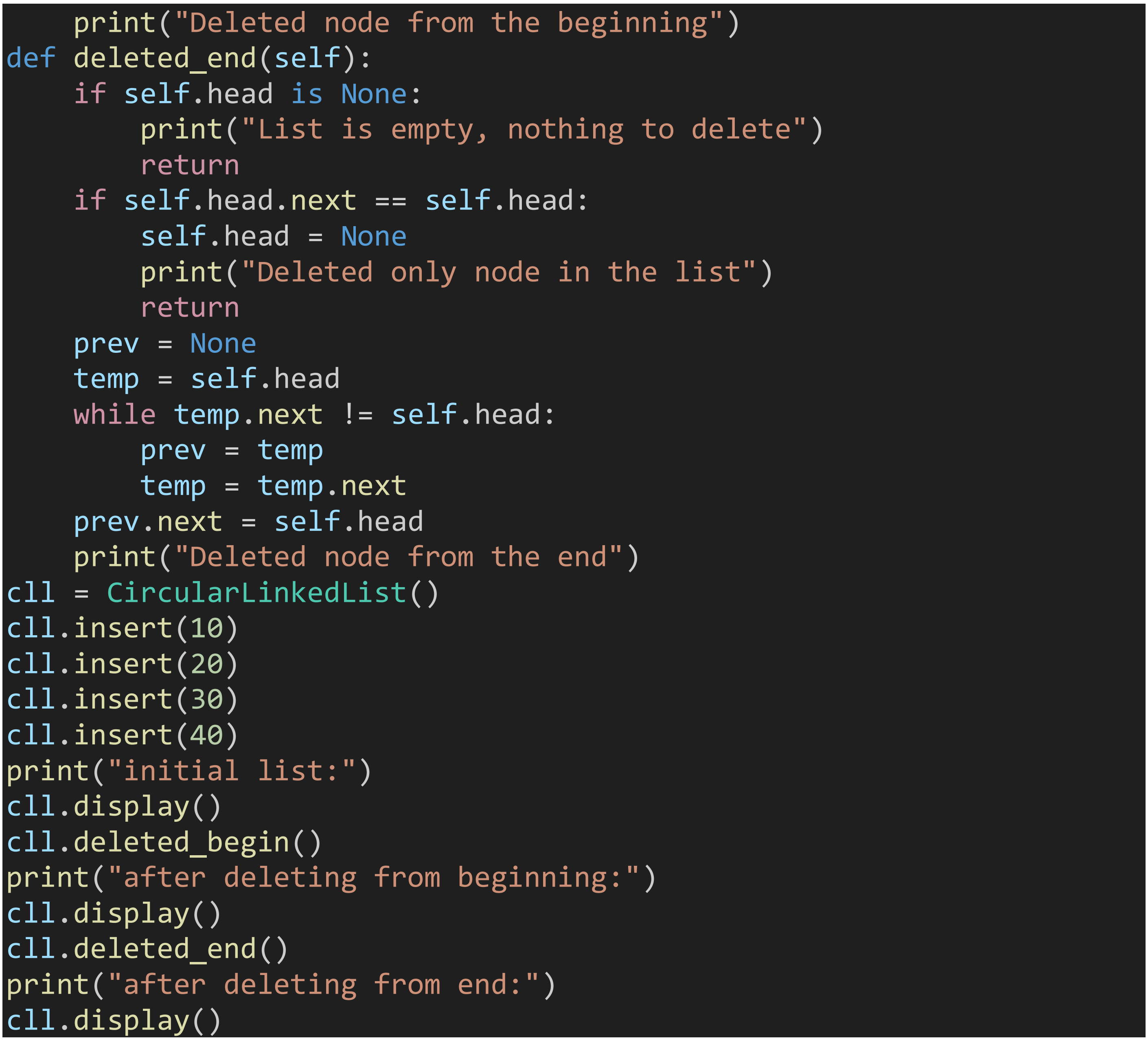


```
            print("Invalid input. Please enter an integer.")
            return None
    elif dt == "str":
        return input("Enter the data:- ")
    else:
        print("Invalid data type. Please choose 'int' or 'str'.")
        return None
if __name__ == "__main__":
    my_list = LinkedList()
    while True:
        print("\n--- Linked List Operations ---")
        print("1. Insert at beginning")
        print("2. Insert at end")
        print("3. Display list")
        print("4. Search for an element")
        print("5. Exit")
        choice = input("Enter your choice (1-5): ")
        if choice == '1':
            data = get_data_from_user()
            if data is not None:
                my_list.insert_at_beginning(data)
        elif choice == '2':
            data = get_data_from_user()
            if data is not None:
                my_list.insert_at_end(data)
        elif choice == '3':
            my_list.display()
        elif choice == '4':
            key = get_data_from_user()
            if key is not None:
                if my_list.search(key):
                    print(f"Key '{key}' found in the list.")
                else:
                    print(f"Key '{key}' not found in the list.")
        elif choice == '5':
            print("Exiting program.")
            break
        else:
            print("Invalid choice. Please enter a number between 1
and 5.")
```

Q7 Insertion in Circular Linked List

Code:

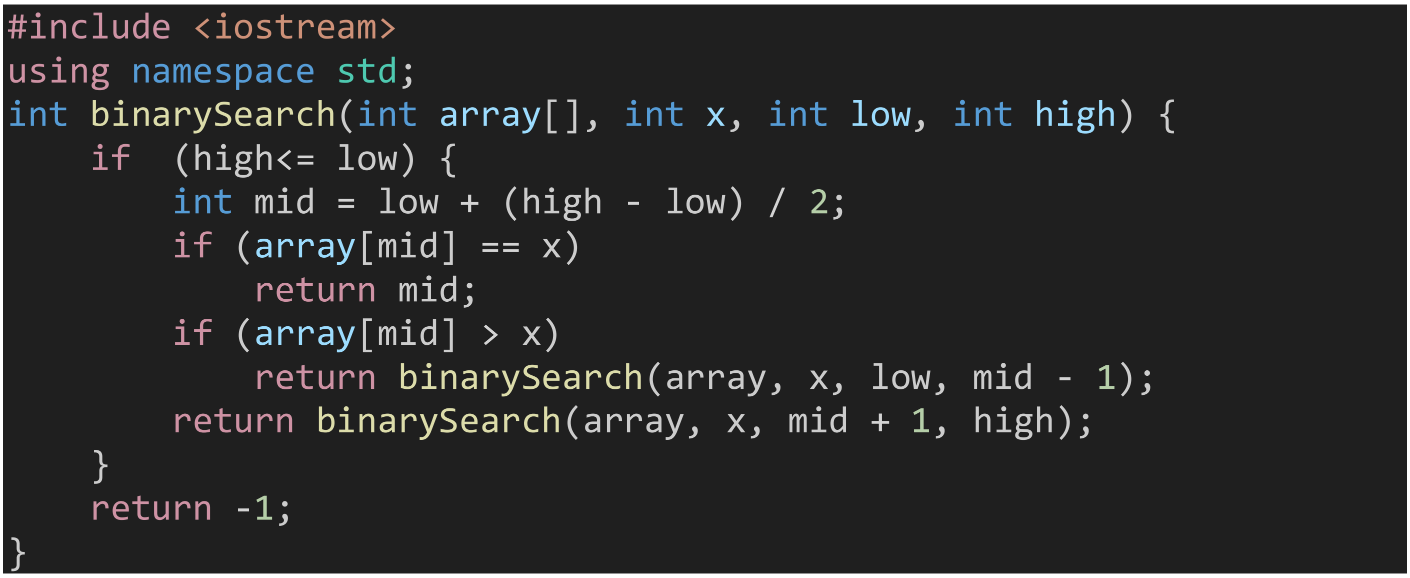
```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
class CircularLinkedList:
    def __init__(self):
        self.head = None
    def insert(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
            self.head.next = self.head
        else:
            temp = self.head
            while temp.next != self.head:
                temp = temp.next
            temp.next = new_node
            new_node.next = self.head
    def display(self):
        if self.head is None:
            print("List is empty")
            return
        temp = self.head
        print("Circular Linked List:"), end=""
        while True:
            print(temp.data, end=" -> ")
            temp = temp.next
            if temp == self.head:
                break
        print("(head)")
def deleted_begin(self):
    if self.head is None:
        print("List is empty, nothing to delete")
        return
    if self.head.next == self.head:
        self.head = None
        print("Deleted only node in the list")
        return
    last = self.head
    while last.next != self.head:
        last = last.next
    last.next = self.head.next
    self.head = self.head.next
```



```
    print("Deleted node from the beginning")
def deleted_end(self):
    if self.head is None:
        print("List is empty, nothing to delete")
        return
    if self.head.next == self.head:
        self.head = None
        print("Deleted only node in the list")
        return
    prev = None
    temp = self.head
    while temp.next != self.head:
        prev = temp
        temp = temp.next
    prev.next = self.head
    print("Deleted node from the end")
cll = CircularLinkedList()
cll.insert(10)
cll.insert(20)
cll.insert(30)
cll.insert(40)
print("initial list:")
cll.display()
cll.deleted_begin()
print("after deleting from beginning:")
cll.display()
cll.deleted_end()
print("after deleting from end:")
cll.display()
```

Q8 Binary Search

Code:



```
#include <iostream>
using namespace std;
int binarySearch(int array[], int x, int low, int high) {
    if  (high<= low) {
        int mid = low + (high - low) / 2;
        if (array[mid] == x)
            return mid;
        if (array[mid] > x)
            return binarySearch(array, x, low, mid - 1);
        return binarySearch(array, x, mid + 1, high);
    }
    return -1;
}
```

```
int main() {
    int array[] = {3,4,5,6,7,8,9};
    int n = sizeof(array) / sizeof(array[0]);
    int x = 4;
    int result = binarySearch(array, x, 0, n - 1);
    if (result == -1)
        cout << "not found";
    else
        cout << "Element is present at index " << result;
    return 0;
}
```

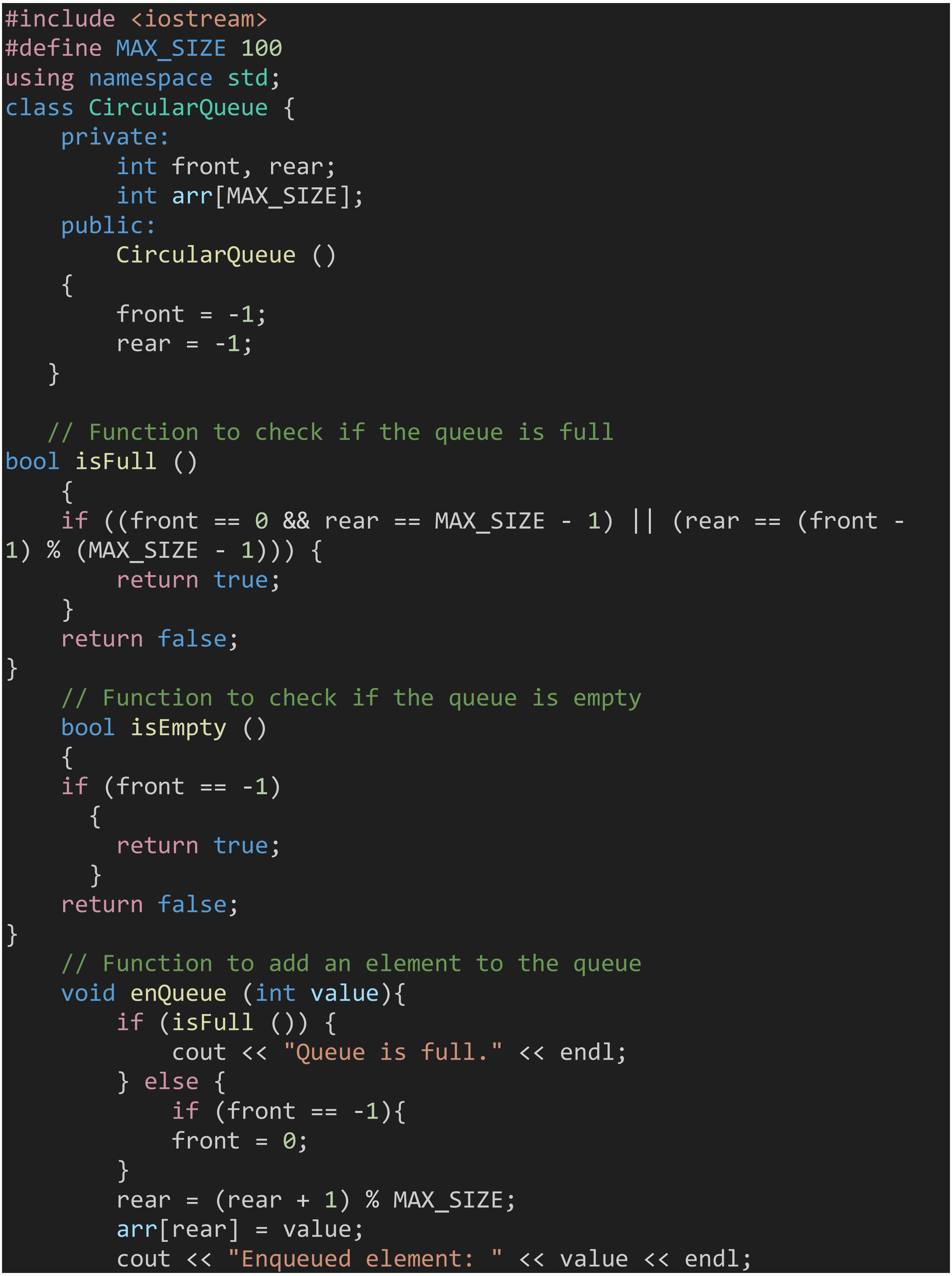
Q9 Bubble Sort

Code:

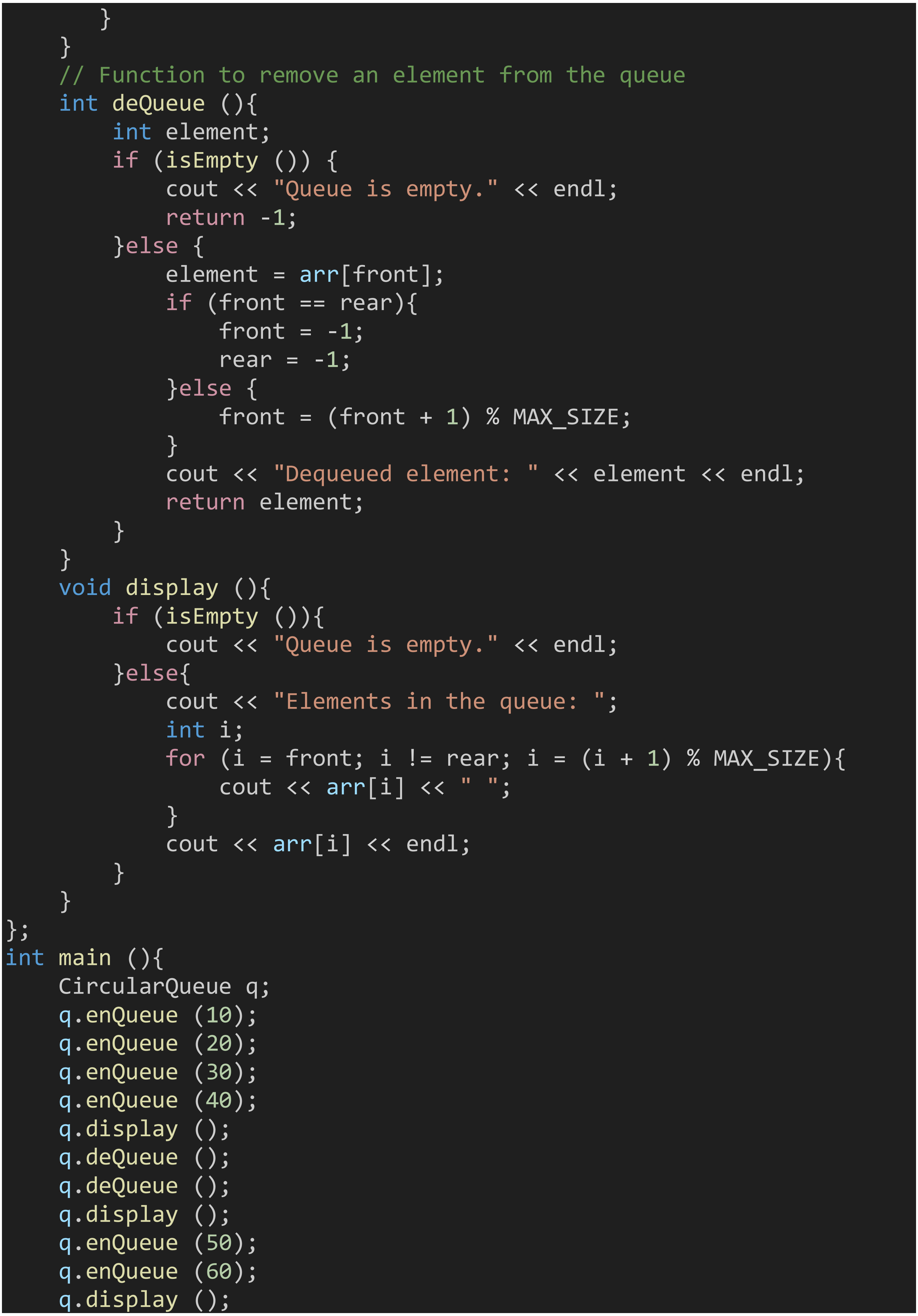
```
include <bits/stdc++.h>
using namespace std;
void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                swap(arr[j], arr[j + 1]);
            }
        }
    }
}
void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}
int main() {
    int arr[] = {5,1,4,2,8};
    int n = sizeof(arr) / sizeof(arr[0]);
    bubbleSort(arr, n);
    cout << "Sorted array: \n";
    printArray(arr, n);
    return 0;
}
```

Q10 Circular Queue : Insertion and Deletion of elements

Code:



```
#include <iostream>
#define MAX_SIZE 100
using namespace std;
class CircularQueue {
    private:
        int front, rear;
        int arr[MAX_SIZE];
    public:
        CircularQueue ()
    {
        front = -1;
        rear = -1;
   }

   // Function to check if the queue is full
bool isFull ()
    {
    if ((front == 0 && rear == MAX_SIZE - 1) || (rear == (front -
1) % (MAX_SIZE - 1))) {
        return true;
    }
    return false;
}
    // Function to check if the queue is empty
    bool isEmpty ()
    {
    if (front == -1)
      {
        return true;
      }
    return false;
}
    // Function to add an element to the queue
    void enQueue (int value){
        if (isFull ()) {
            cout << "Queue is full." << endl;
        } else {
            if (front == -1){
            front = 0;
        }
        rear = (rear + 1) % MAX_SIZE;
        arr[rear] = value;
        cout << "Enqueued element: " << value << endl;
```



```
       }
    }
    // Function to remove an element from the queue
    int deQueue (){
        int element;
        if (isEmpty ()) {
            cout << "Queue is empty." << endl;
            return -1;
        }else {
            element = arr[front];
            if (front == rear){
                front = -1;
                rear = -1;
            }else {
                front = (front + 1) % MAX_SIZE;
            }
            cout << "Dequeued element: " << element << endl;
            return element;
        }
    }
    void display (){
        if (isEmpty ()){
            cout << "Queue is empty." << endl;
        }else{
            cout << "Elements in the queue: ";
            int i;
            for (i = front; i != rear; i = (i + 1) % MAX_SIZE){
                cout << arr[i] << " ";
            }
            cout << arr[i] << endl;
        }
    }
};
int main (){
    CircularQueue q;
    q.enQueue (10);
    q.enQueue (20);
    q.enQueue (30);
    q.enQueue (40);
    q.display ();
    q.deQueue ();
    q.deQueue ();
    q.display ();
    q.enQueue (50);
    q.enQueue (60);
    q.display ();
    return 0;
}
```

Q11 Linear Search

Code:

```
#include <iostream>
using namespace std;
int main() {
    int arr[10], i, num, index;
    cout<<"enter 10 numbers: ";
    for(i=0;i<10;i++) {
        cin>>arr[i];
    }
    cout<<"enter number to be searched: ";
    cin>>num;
    for(i=0;i<10;i++) {
        if(arr[i]==num) {
            index=i;
            break;
        }
    }
    if(i<10)
        cout<<"number found at index: "<<index;
    else
        cout<<"number not found";
}
```