DS September Assignments

Natya Vidhan Biswas

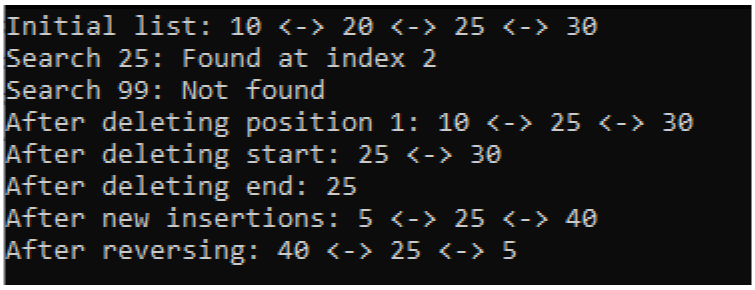
B.Sc. (H) C.S. 25771

Write a program to implement doubly linked list as an ADT that supports the following operations:

- Insert an element x at the beginning of the doubly linked list
- Insert an element x at ith position in the doubly linked list
- Insert an element x at the end of the doubly linked list
- Remove an element from the beginning of the doubly linked list
- Remove an element from ith position in the doubly linked list
- Remove an element from the end of the doubly linked list
- Search for an element x in the doubly linked list and return its pointer
- Reverse a doubly linked list
- Display

```cpp
#include <iostream>

using namespace std;

class Node {
    Node *prev;
    Node *next;
    int val;
public:
    Node(int v = 0, Node *p = 0, Node *n = 0) {
        prev = p;
        next = n;
        val = v;
    };
    friend class DoublyLL;
};

class DoublyLL {
    Node *head;
    Node *tail;
public:
    DoublyLL(Node *h = 0, Node *t = 0) {
        head = h;
        tail = t;
    }


    bool isempty() {
        return head == 0;
    }
```

```
void print() {
    Node *p;
    p = head;
    while (p)
    {
        cout << p->val;
        if (p->next) {
            cout << " <-> ";
        }
        p = p->next;
    }
    cout << "\n";
    return;
}

void add_start(int val) {
    Node *p;
    p = new Node(val);
    if (!head) {
        head = tail = p;
    } else {
        Node *q;
        q = head;
        head = p;
        head->next = q;
        q->prev = head;
    }
    return;
}

void add_end(int val) {
    Node *p;
    p = new Node(val);
    if (!tail) {
        tail = head = p;
    } else {
        Node *q;
        q = tail;
        tail = p;
        tail->prev = q;
        q->next = tail;
    }
    return;
}

void add_ith(int val, int i) {
    Node *p;
    p = new Node(val);
    int n = 0;
    Node *q;
    q = head;
    while (n < i-1) {
        q = q->next;
        n++;
    }
    p->next = q->next;
    p->prev = q;
    q->next = p;
    p->next->prev = p;
    return;
}

void delete_start() {
    Node *p;
    p = head;
    head = head->next;
    if (!head) {
        tail = 0;
    }
    delete(p);
    return;
}

void delete_end() {
    Node *p;
    p = tail;
    tail = tail->prev;
    if (!tail) {
        head = 0;
    }
    delete(p);
    return;
}

void delete_ith(int i) {
    int n = 0;
    Node *q;
    q = head;
    while (n < i-1) {
        q = q->next;
        n++;
    }
    q->next = q->next->next;
    q->next->prev = q;
    return;
}

bool search(int val) {
    Node *curr = head;
    while (curr) {
        if (curr->val == val) {
            return true;
        }
        curr = curr->next;
    }
    return false;
}
void reverse() {
    if (!head || !head->next) {
        return;
    }
    Node *curr = head;
    while (curr) {
        Node *nextNode = curr->next;
        curr->next = curr->prev;
        curr->prev = nextNode;
        curr = nextNode;
    }
    Node *oldHead = head;
    head = tail;
    tail = oldHead;
}
};



int main() {
DoublyLL list;
list.add_start(10);
list.add_end(20);
list.add_end(30);
list.add_ith(25, 2);

cout << "Initial list: ";
list.print();

cout << (list.search(25) ? "Search 25: Found\n" : "Search 25: Not
found\n");
cout << (list.search(99) ? "Search 99: Found\n" : "Search 99: Not
found\n");

list.delete_ith(1);
cout << "After deleting position 1: ";
list.print();

list.delete_start();
cout << "After deleting start: ";
list.print();

list.delete_end();
cout << "After deleting end: ";
list.print();

list.add_start(5);
list.add_end(40);
cout << "After new insertions: ";
list.print();

list.reverse();
cout << "After reversing: ";
list.print();
return 0;
}
```

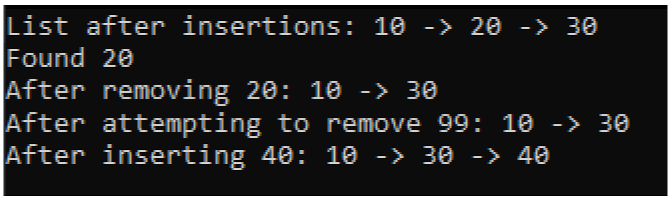


```
Initial list: 10 <-> 20 <-> 25 <-> 30
Search 25: Found at index 2
Search 99: Not found
After deleting position 1: 10 <-> 25 <-> 30
After deleting start: 25 <-> 30
After deleting end: 25
After new insertions: 5 <-> 25 <-> 40
After reversing: 40 <-> 25 <-> 5
```

Write a program to implement circular linked list as an ADT which supports the following operations:

- Insert an element x in the list (after cursor)
- Remove an element from the list (first search for the element anddelete)
- Search for an element x in the list and return its pointer
- Display

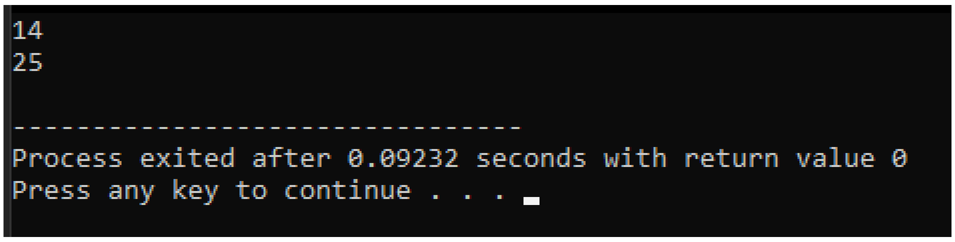
```cpp
#include <iostream>

using namespace std;

class Node {
    int val;
    Node *next;
public:
    Node(int v = 0, Node *n = 0) : val(v), next(n) {}
    friend class CircularLL;
};

class CircularLL {
    Node *cursor;
public:
    CircularLL(Node *c = 0) : cursor(c) {}

    bool isempty() {
        return cursor == 0;
    }

    void insert_after_cursor(int val) {
        Node *p = new Node(val);
        if (!cursor) {
            p->next = p;
            cursor = p;
        } else {
            p->next = cursor->next;
            cursor->next = p;
            cursor = p;
        }
    }

    Node* search(int val) {
        if (!cursor) {
            return 0;
        }
        Node *start = cursor->next;
        Node *curr = start;
        do {
            if (curr->val == val) {
                return curr;
            }
            curr = curr->next;
        } while (curr != start);
        return 0;
    }

    void remove(int val) {
        if (!cursor) {
            return;
        }
        Node *prev = cursor;
        Node *curr = cursor->next;
        do {
            if (curr->val == val) {
                prev->next = curr->next;
                if (curr == cursor) {
                    cursor = (curr == curr->next) ? 0 : prev;
                }
                delete curr;
                return;
            }
            prev = curr;
            curr = curr->next;
        } while (curr != cursor->next);
    }

    void display() {
        if (!cursor) {
            cout << "(empty)\n";
            return;
        }
        Node *start = cursor->next;
        Node *curr = start;
        do {
            cout << curr->val;
            curr = curr->next;
            if (curr != start) {
                cout << " -> ";
            }
        } while (curr != start);
        cout << "\n";
    }
};

int main() {
    CircularLL list;
    list.insert_after_cursor(10);
    list.insert_after_cursor(20);
    list.insert_after_cursor(30);

    cout << "List after insertions: ";
    list.display();
    Node *found = list.search(20);
    cout << (found ? "Found 20\n" : "20 not found\n");

    list.remove(20);
    cout << "After removing 20: ";
    list.display();

    list.remove(99);
    cout << "After attempting to remove 99: ";
    list.display();

    list.insert_after_cursor(40);
    cout << "After inserting 40: ";
    list.display();

    return 0;
}
```



```
List after insertions: 10 -> 20 -> 30
Found 20
After removing 20: 10 -> 30
After attempting to remove 99: 10 -> 30
After inserting 40: 10 -> 30 -> 40
```

Implement Stack as an ADT (using array and linked list) and use it to evaluate a prefix/postfix expression.

```
#include <iostream>
#include <sstream>
#include <string>
#include <vector>

using namespace std;

class Node {
    int val;
    Node *next;
public:
    Node(int v = 0, Node *n = 0) {
        val = v;
        next = n;
    };
    friend class Stack;
};

class Stack {
    Node *head;
    int cap;
public:
    Stack() {
        head = 0;
        cap = 0;
    };
    void push(int val) {
        Node *p;
        p = new Node(val);
        if (!head) {
            head = p;
        } else {
            Node *q;
            q = head;
            head = p;
            head->next = q;
        };
        cap++;
        return;
    };

    void pop() {
        if (head) {
            Node *q;
            q = head;
            head = head->next;
            delete(q);
            cap--;
        }
        return;
    }
    int peek() {
        return head->val;
    }

    bool isEmpty() {
        return head == 0;
    }

    int size() {
        return cap;
    }

    void clear() {
        head = 0;
        return;
    }

    int search(int v) {
        Node *q;
        q = head;
        int n;
        while (q->next) {
            if (q->val == v) {
                return n;
            }
            q =  q->next;
            n++;
        }
        return -1;
    }
};

int applyOp(int lhs, int rhs, char op) {
    switch (op) {
        case '+': return lhs + rhs;
        case '-': return lhs - rhs;
        case '*': return lhs * rhs;
        case '/': return rhs == 0 ? 0 : lhs / rhs;
        default:  return 0;
    }
}

bool isOp(const string &token) {
    return token.size() == 1 && (token[0] == '+' || token[0] == '-' ||
token[0] == '*' || token[0] == '/');
}

int evaluatePostfix(const string &expr) {
    Stack st;
    string token;
    stringstream ss(expr);
    while (ss >> token) {
        if (isOp(token)) {
            int rhs = st.peek(); st.pop();
            int lhs = st.peek(); st.pop();
            st.push(applyOp(lhs, rhs, token[0]));
        } else {
            st.push(stoi(token));
        }
    }
    int result = st.peek(); st.pop();
    return result;
}

int evaluatePrefix(const string &expr) {
    vector<string> tokens;
    string token;
    stringstream ss(expr);
    while (ss >> token) {
        tokens.push_back(token);
    }
    Stack st;
    for (int i = static_cast<int>(tokens.size()) - 1; i >= 0; --i) {
        if (isOp(tokens[i])) {
            int lhs = st.peek(); st.pop();
            int rhs = st.peek(); st.pop();
            st.push(applyOp(lhs, rhs, tokens[i][0]));
        } else {
            st.push(stoi(tokens[i]));
        }
    }
    int result = st.peek(); st.pop();
    return result;
}

int main() {
    cout << evaluatePostfix("5 1 2 + 4 * + 3 -") << "\n";
    cout << evaluatePrefix("- + 7 * 4 5 + 2 0") << "\n";
    return 0;
}
```

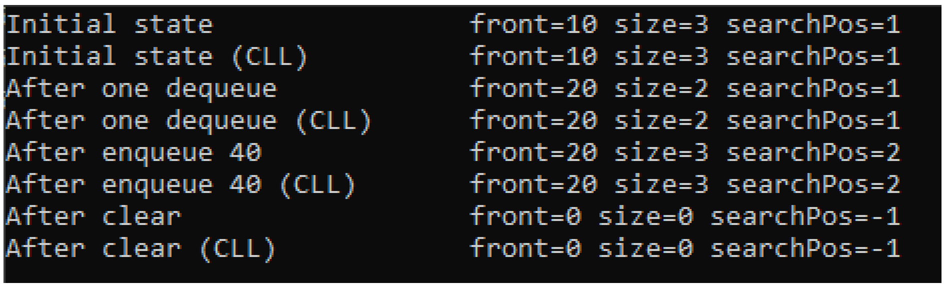


```
14
25

--------------------------------
Process exited after 0.09232 seconds with return value 0
Press any key to continue . . .
```

Implement Queue as an ADT (using array and circular linked list).

```
#include <iostream>
#include <iomanip>

using namespace std;

class ArrayQueue {
    static const int MAX = 32;
    int data[MAX];
    int frontIdx;
    int rearIdx;
    int count;
public:
    ArrayQueue() : frontIdx(0), rearIdx(0), count(0) {}

    bool isEmpty() const {
        return count == 0;
    }

    bool isFull() const {
        return count == MAX;
    }

    void enqueue(int val) {
        if (isFull()) {
            cout << "ArrayQueue overflow\n";
            return;
        }
        data[rearIdx] = val;
        rearIdx = (rearIdx + 1) % MAX;
        count++;
    }

    void dequeue() {
        if (isEmpty()) {
            cout << "ArrayQueue underflow\n";
            return;
        }
        frontIdx = (frontIdx + 1) % MAX;
        count--;
    }

    int peek() const {
        return isEmpty() ? 0 : data[frontIdx];
    }

    int size() const {
        return count;
    }

    void clear() {
        frontIdx = rearIdx = count = 0;
    }
```

```cpp
    int search(int val) const {
        if (isEmpty()) {
            return -1;
        }
        for (int i = 0; i < count; ++i) {
            int idx = (frontIdx + i) % MAX;
            if (data[idx] == val) {
                return i;
            }
        }
        return -1;
    }
};

class Node {
    int val;
    Node *next;
public:
    Node(int v = 0, Node *n = 0) : val(v), next(n) {}
    friend class CircularListQueue;
};

class CircularListQueue {
    Node *tail;
    int count;
public:
    CircularListQueue(Node *t = 0, int c = 0) : tail(t), count(c) {}

    bool isEmpty() const {
        return tail == 0;
    }

    void enqueue(int val) {
        Node *node = new Node(val);
        if (!tail) {
            node->next = node;
            tail = node;
        } else {
            node->next = tail->next;
            tail->next = node;
            tail = node;
        }
        count++;
    }

    void dequeue() {
        if (!tail) {
            cout << "CircularListQueue underflow\n";
            return;
        }
        Node *head = tail->next;
        if (head == tail) {
            tail = 0;
        } else {
            tail->next = head->next;
        }
        delete head;
        count--;
    }

    int peek() const {
        return tail ? tail->next->val : 0;
    }

    int size() const {
        return count;
    }

    void clear() {
        while (!isEmpty()) {
            dequeue();
        }
    }

    int search(int val) const {
        if (!tail) {
            return -1;
        }
        Node *head = tail->next;
        Node *curr = head;
        int idx = 0;
        do {
            if (curr->val == val) {
                return idx;
            }
            curr = curr->next;
            idx++;
        } while (curr != head);
        return -1;
    }
};

void printQueueState(const char *label, int frontVal, int size, int pos) {
    cout << left << setw(28) << label
         << " front=" << frontVal
         << " size=" << size
         << " searchPos=" << pos << "\n";
}

int main() {
    ArrayQueue aq;
    CircularListQueue cq;

    for (int v : {10, 20, 30}) {
        aq.enqueue(v);
        cq.enqueue(v);
    }
    printQueueState("Initial state", aq.peek(), aq.size(), aq.search(20));
    printQueueState("Initial state (CLL)", cq.peek(), cq.size(),
cq.search(20));

    aq.dequeue();
    cq.dequeue();
    printQueueState("After one dequeue", aq.peek(), aq.size(),
aq.search(30));
    printQueueState("After one dequeue (CLL)", cq.peek(), cq.size(),
cq.search(30));

    aq.enqueue(40);
    cq.enqueue(40);
    printQueueState("After enqueue 40", aq.peek(), aq.size(),
aq.search(40));
    printQueueState("After enqueue 40 (CLL)", cq.peek(), cq.size(),
cq.search(40));

    aq.clear();
    cq.clear();
    printQueueState("After clear", aq.peek(), aq.size(), aq.search(10));
    printQueueState("After clear (CLL)", cq.peek(), cq.size(),
cq.search(10));

    return 0;
}
```



```
Initial state                front=10 size=3 searchPos=1
Initial state (CLL)          front=10 size=3 searchPos=1
After one dequeue            front=20 size=2 searchPos=1
After one dequeue (CLL)      front=20 size=2 searchPos=1
After enqueue 40             front=20 size=3 searchPos=2
After enqueue 40 (CLL)       front=20 size=3 searchPos=2
After clear                  front=0 size=0 searchPos=-1
After clear (CLL)            front=0 size=0 searchPos=-1
```