```
//Palak Mishra (210690)

#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};

struct Node* head = NULL;
struct Node* tail = NULL;
int length = 0;

//insertItem - any position, by default, the end
//append - to initiate - No need of this now
//deleteItem - by value
//searchItem
//printItems
//reversePrintItems

void insertItem(int item, int pos) {
    if (head == NULL) {
        struct Node* new = (struct Node*)malloc(sizeof(struct
Node));
        new->data = item;
        new->next = NULL;
        new->prev = NULL;
        head = new;
        tail = head;
    } else if (pos <= 1) {
        struct Node* new = (struct Node*)malloc(sizeof(struct
Node));
        new->data = item;
        new->next = head;
        new->prev = NULL;
        head->prev = new;
        head = new;
    } else if (pos >= length+1) {
        struct Node* new = (struct Node*)malloc(sizeof(struct
Node));
        new->data = item;
        new->next = NULL;
        tail->next = new;
        new->prev = tail;
        tail = new;
    } else {
        struct Node* temp = head;
        for (int i = 0; i < pos-1; i++) {
            temp = temp->next;
        }
        struct Node* new = (struct Node*)malloc(sizeof(struct
Node));
```

```
        new->data = item;
        new->next = temp;
        new->prev = temp->prev;
        temp->prev->next = new;
        temp->prev = new;
    }
    printf("Node inserted.\n");
    length++;
}

void deleteItem(int item) {
    if (head == NULL) {
        printf("List is empty. Can't be deleted.\n");
        return;
    }
    struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
    temp = head;
    while(temp->data != item) {
        temp = temp->next;
        if (temp == NULL) break;
    }
    if (temp == NULL) {
        printf("Isn't present in the list. Can't be deleted.\n");
        return;
    } else if (length == 1) {
        head = NULL;
        tail = NULL;
    } else if (temp == head) {
        head = head->next;
        head->prev = NULL;
    } else if (temp == tail) {
        tail = temp->prev;
        tail->next = NULL;
    } else {
        temp->prev->next = temp->next;
        temp->next->prev = temp->prev;
    }
    printf("Node deleted.\n");
    length--;
}

void printItems() {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    printf("Doubly Linked List: \n");
    struct Node* current = head;
    while(current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}
```

```
void reversePrintItems() {
    if (tail == NULL) {
        printf("List is empty.\n");
        return;
    }
    printf("Reversed Doubly Linked List: \n");
    struct Node* current = tail;
    while(current != NULL) {
        printf("%d ", current->data);
        current = current->prev;
    }
    printf("\n");
}

//STACK
//pop - fetch the last item, and pass it to deleteItem
//push - insertItem, default

void push(int item) {
    insertItem(item, length+1);
}

void pop() {
    if (head == NULL) {
        printf("Stack Underflow!\n");
        return;
    }
    deleteItem(tail->data);
}

//QUEUE
//enQueue - insertItem, default
//deQueue - fetch the first element, pass it to deleteItem

void enQueue(int item) {
    insertItem(item, length+1);
}

void deQueue() {
    if (head == NULL) {
        printf("Queue Underflow!\n");
        return;
    }
    deleteItem(head->data);
}

int main() {
    int choice;
    printf("-----DOUBLY LINKED LIST IMPLEMENTATION-----\n");
    printf("Enter your choice: \n1. Insert\n2. Delete\n3. Print
Items in Original Order\n4. Print Items in Reverse Order\n5.
Exit\n");
    scanf("%d", &choice);
```

```
    while(choice != 5) {
        if (choice == 1) {
            int item;
            printf("Enter the item: ");
            scanf("%d", &item);
            int pos;
            //pos = length+1;
            printf("\nEnter the position: ");
            scanf("%d", &pos);
            insertItem(item, pos);
        } else if (choice == 2) {
            int item;
            printf("Enter the item: ");
            scanf("%d", &item);
            deleteItem(item);
        } else if (choice == 3) printItems();
        else if (choice == 4) reversePrintItems();
        else if (choice == 5) break;
        else printf("Wrong Choice!\n");

        printf("\nEnter your choice: \n1. Insert\n2. Delete\n3.
Print Items in Original Order\n4. Print Items in Reverse Order\n5.
Exit\n");
        scanf("%d", &choice);
    }

    printf("\n-----STACK IMPLEMENTAION-----\n");
    head = NULL;
    tail = NULL;
    length = 0;
    printf("Enter your choice: \n1. Push\n2. Pop\n3. Display\n4.
Exit\n");
    scanf("%d", &choice);

    while(choice != 4) {
        if (choice == 1) {
            int item;
            printf("Enter the item: ");
            scanf("%d", &item);
            push(item);
        } else if (choice == 2) {
            pop();
        } else if (choice == 3) {
            printItems();
        } else if (choice == 4) break;
        else printf("Wrong Choice!\n");

        printf("\nEnter your choice: \n1. Push\n2. Pop\n3.
Display\n4. Exit\n");
        scanf("%d", &choice);
    }

    printf("\n-----QUEUE IMPLEMENTAION-----\n");
```

```
    head = NULL;
    tail = NULL;
    length = 0;
    printf("Enter your choice: \n1. Enqueue\n2. Dequeue\n3.
Display\n4. Exit\n");
    scanf("%d", &choice);

    while(choice != 4) {
        if (choice == 1) {
            int item;
            printf("Enter the item: ");
            scanf("%d", &item);
            enQueue(item);
        } else if (choice == 2) {
            deQueue();
        } else if (choice == 3) {
            printItems();
        } else if (choice == 4) break;
        else printf("Wrong Choice!\n");

        printf("\nEnter your choice: \n1. Enqueue\n2. Dequeue\n3.
Display\n4. Exit\n");
        scanf("%d", &choice);
    }

    printf("\n-----ALL TASKS REVIEWED-----\n");
}
```