Date 29/01/2024
Page

lab-6

1) stack implementation using single linked lists

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

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

void display (struct Node *top){
        if(top != NULL){
                printf("stack elements are:\t");
                while (top != NULL){
                        printf("%d \t", top->data);
                        top = top->next;
                }
                printf("\n");
        }
        else{
                printf("stack is empty\n");
        }
}

struct Node *push (struct Node *top, int data){
        struct Node * newNode= (struct Node*)
                malloc(sizeof(struct Node));
        if (newNode == NULL){
                printf("stack overflow\n");
                return top;
        }
```

```c
        newNode->data = data;
        newNode->next = top;
        top = newNode;
        return top;
}

struct Node* pop( struct Node *top, int *
                                  popData){
    if(top ==NULL){
        printf("Stack underflow\n");
        *popdata = -1;
        return NULL;
    }
    struct Node *temp = top;
    *popData = temp->data;
    top = top->next;
    free(temp);
    return top;
}

int main(){
    int op, n, popElement;
    struct Node * top = NULL;
    printf(" Enter 1. Push \n 2. Pop \n 3. -1
              to stop\n");
    while (1) {
        printf("Enter operation: \n");
        scanf("%d",&op);
        if(op ==-1){
            printf("Execution stopped\n");
            break;
        }
```

[illegible] 29/11/24

```
        switch(op){
            case 1:
              printf("Enter the element to push\n");
              scanf("%d",&n);
              top=push(top,n);
              break;
            case 2:
              top=pop(top,&popElement);
              if(popElement!=-1){
                  printf("Popped Element:%d
                              \n",popElement);
               }
               break;
            default: printf("Invalid choice\n");
        }
        display(top);
    }
    return 0;
    }
```

```
o/p
Enter 1.Push
2. Pop
3. -1 to stop
Enter operation
2
Stack underflow
Stack is empty
Enter operation;
1
Enter element to push 5
Stack elements are:   5
Enter operation 1
Enter element to push  4 :
stack elements are: 4    5
Enter operation: 1
Enter element to push
stack elements are : 6    4 . 5
Enter operatation: 2
popped element: 6
stack elements are:  4    5
Enter operation:-1:    Execution stopped
```

2) Queue implementation using single linked list.

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

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

void display (struct node *front){
    if (front == NULL){
        printf("Queue is empty\n");
        return;
    }
    struct Node * temp = front;
    printf("Queue elements are:\t");
    while(temp!= NULL){
        printf("%d\t", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
```

[illegible] 29/11/24.

Date
Page

```
void enqueue (struct Node **front,
        struct Node ** rear, int data){
    struct Node * newNode = (struct Node *)
        malloc( sizeof (struct Node));
    if (newNode == NULL){
        printf("Queue Overflow\n");
        return;
    }
    newNode -> data = data;
    newNode -> next = NULL;
    if(*rear == NULL){
        *front = *rear = newNode;
        return;
    }
    (*rear) -> next = newNode;
    *rear = newNode;
}

int dequeue(struct Node ** front, struct
                Node **rear ){
    if (*front == NULL){
        printf("Queue Underflow\n");
        return -1;
    }
    struct Node * temp = * front;
    int dequeueData = temp -> data;
    *front = (*front) -> next;
    if (* front = NULL){
        * rear = NULL
    }
    free(temp);
    return dequeueData;
}
```

```
int main(){
    int op, n, dequeueElement;
    struct Node * front = NULL;
    struct Node * rear = NULL;
    printf("Enter 1.Enqueue\n 2. Dequeue \n
              3. -1 to stop\n");
    while(1){
        printf("Enter operation:\n");
        scanf("%d", &op);
        if(op==-1){
            printf("Exicution stoped \n");
            break;
        }
        switch(op){
            case 1:
             printf("Enter the element to
                     enqueue \n");
             scanf("%d", &n);
             enqueue(& front, & rear, n);
             break;
           case 2:
             dequeueElement = dequeue
                   (& front, & rear);
             if(dequeueElement != -1){
                 printf(" Dequeued element
                   : %d \n", dequeueEle
                         ment);
             }
             break;
        }
        display(front);
    }
return 0;
}
```

O/P

```
Enter 1. Enqueue
2. Dequeue
3. -1 to stop
Enter operation: 2
Queue underflow
Queue is empty
Enter operation; 1
Enter the element to enqueue 3
Queue elements are; 3
Enter operation; 1
Enter element to enqueue 2
Queue elements are: 3 2
Enter operation; 1
Enter element to enqueue 1
Queue elements are: 3 2 1
Enter operation : 2
Dequeued element; 3
Queue elements are ; 2 1
Enter operation: -1
Execution stopped.
```

3) Standard linked list sorting

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

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

```
void display (struct Node* head){
        struct Node current=head;
        while(current != NULL){
            printf("%d\t", current->data);
            current= current ->next;
        }
        printf("\n");
}

struct Node sortLinkedList(struct Node *head){
        if(head == NULL || head ->next== NULL){
            return head;
        }
        int swapped;
        struct Node * temp;
        struct Node *end = NULL
        do {
            swapped = 0;
            struct Node * current =head;
            while(current ->next != end){
                if (current->data > current ->next->data)
                {   int tempData= current->data;
                    current->data = current->next->data;
                    current->next->data= tempData;
                    swapped= 1;
                }
                current =current ->next
            }
            end =current;
        }
        while(swapped);
        return head;
}
```

```
int main(){
    struct Node *node1=(structNode*)malloc(sizeof(structNode));
    struct Node *nod2 =(structNode*)malloc(sizeof(structNode));
    struct Node *node3=(structNode*)malloc(sizeof(structNode));
    struct Node *node4=(structNode*)malloc(sizeof(structNode));
    node1->data = 4;
    node2-> data = 2;
    node3-> data = 7;
    node4-> data = 1;
    node1->next = node2;
    node2->next = node3;
    node3->next = node4;
    node4->next = NULL;
    printf("original linked list \n");
    display(node1);
    node1 = sortLinkedList(node1);
    printf("sorted Linked list \n");
    display(node1);
    free(node1);
    free(nod2);
    free(node3);
    free(node4);
    return0;
}
```

[illegible] 29/11/24

```
Original Linked list:
4   2   7   1
Sorted  Linked  list:
1   2   4   7
```

4) standard linked list reversal

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

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

void display (struct Node *head) {
    struct Node current = head;
    while (current != NULL) {
        print ("%d \t", current -> data);
        current = current -> next;
    }
    printf ("\n");
}

struct Node * reverseLinkedList (struct Node * head)
    struct Node *prev, *current, *next;
    prev = NULL;
    current = head;
    while (current != NULL) {
        next = current -> next;
        current -> next = prev;
        prev = current;
        current = next;
    }
    return prev;
}
```

```
int main() {
  struct Node * node1 = (struct Node *)malloc(sizeof(struct Node));
  struct Node * node2 = (struct Node *)malloc(sizeof(struct Node));
  struct Node * node3 = (struct Node *)malloc(sizeof(struct Node));
  node1 -> data = 10;
  node2 -> data = 20;
  node3 -> data = 30;
  node1 -> next = node2;
  node2 -> next = node3;
  node3 -> next = NULL;

      printf("original linked list \n");
      display(node1);

      struct Node * revHead = reverseLinkedList
                                    (node1);
      printf("Reversed linked list \n");
      display(revHead);

      free(node1);
      free(node2);
      free(node3);
      return 0;
}
```

[illegible] 29/11/24

```
original linked list:
10   20   30
Reversed linked list:
30   20   10
```

5) standard linked list concatenation.

```
#include<stdio.h>
#include<stdlib.h>
struct Node{
        int data;
        struct Node *next;
}
void display(struct Node *head){
        struct Node current=head;
        while(current!=NULL){
                printf("%d\t", current->data);
                current=current->next;
        }
        printf("\n");
}

struct Node *::concLinkedList,(struct Node*
                        list1, struct Node* list2){
        if(list1==NULL){
                return list2;
        }
        struct Node *current = list1;
        while(current->next!=NULL){
                current=current->next;
        }
        current->next=list2;
        return list1;
}
```

```
int main(){
    struct Node * list1 = (struct

    list 1 -> data = 10 ;
    list1 -> next = NULL;

    struct Node* list 2 =
    list 2 -> data = 20 .
    list 2 -> next = NULL;

    printf(" original linked list 1 and 2 \n");
    display(list1);
    display(list 2);

    struct Node* concLinkedList = concLinkedList
                          ( list1, list 2);
    printf(" concatenated Linked list \n");
    display(concLinkedList);

    free(list1);
    free(list 2);
    return 0;
}
```

[illegible]

```
original Linked list 1:
 10
original linked list 2:
20
concatenated Linked list:
10   20
```

Demonstration of Leetcode.

Demonstration Shown

NP 29/1/24

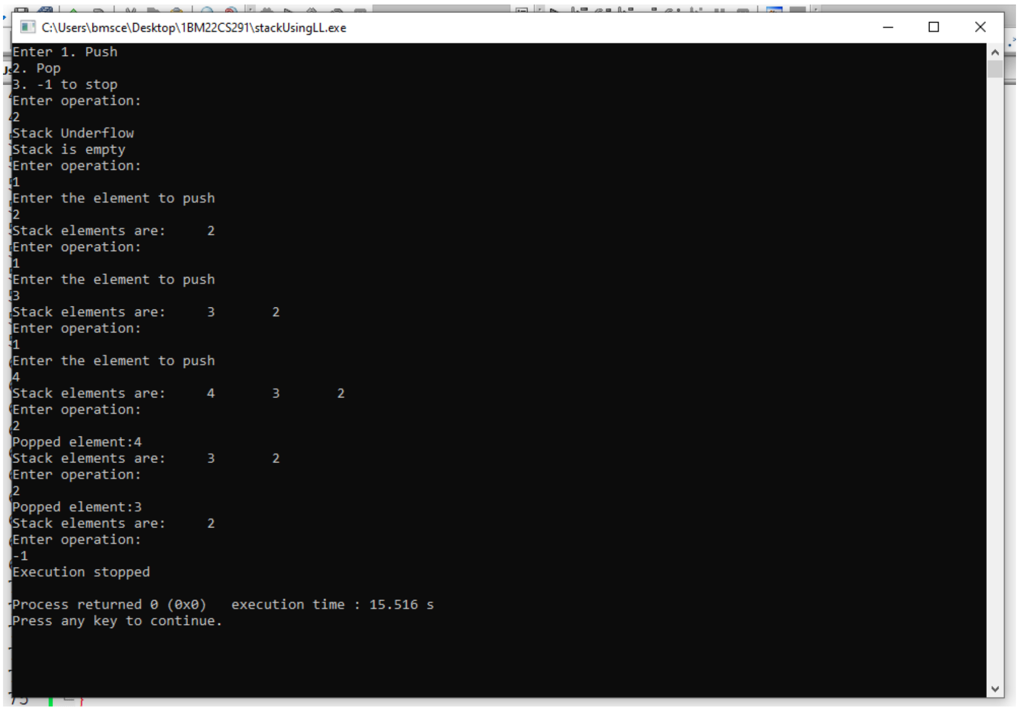
— LO.

—

—

created and loged in account..
For solving quetions we need to consider the parameters given and return the required answer as the platform wants. Then it checks our solutions with some predefined test cases and displays our stand and codes efficiency amoung the global users.

Stack implementation using linked lists:

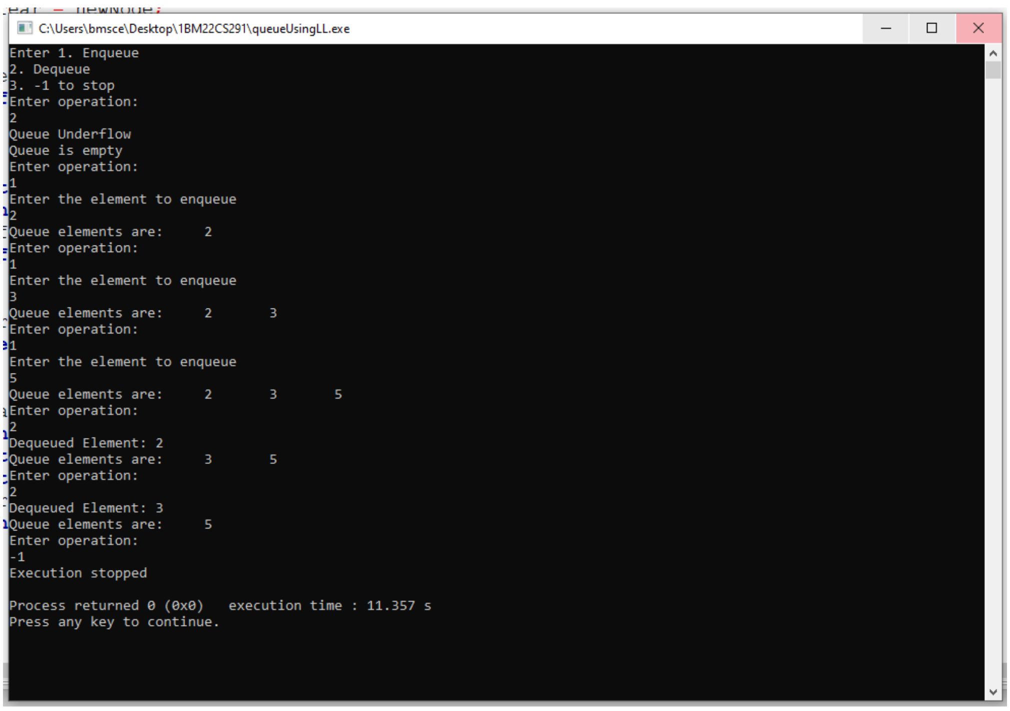


C:\Users\bmsce\Desktop\1BM22CS291\stackUsingLL.exe

```
Enter 1. Push
2. Pop
3. -1 to stop
Enter operation:
2
Stack Underflow
Stack is empty
Enter operation:
1
Enter the element to push
2
Stack elements are:     2
Enter operation:
1
Enter the element to push
3
Stack elements are:     3       2
Enter operation:
1
Enter the element to push
4
Stack elements are:     4       3       2
Enter operation:
2
Popped element:4
Stack elements are:     3       2
Enter operation:
2
Popped element:3
Stack elements are:     2
Enter operation:
-1
Execution stopped

Process returned 0 (0x0)   execution time : 15.516 s
Press any key to continue.
```

Queue implementation using linked lists:

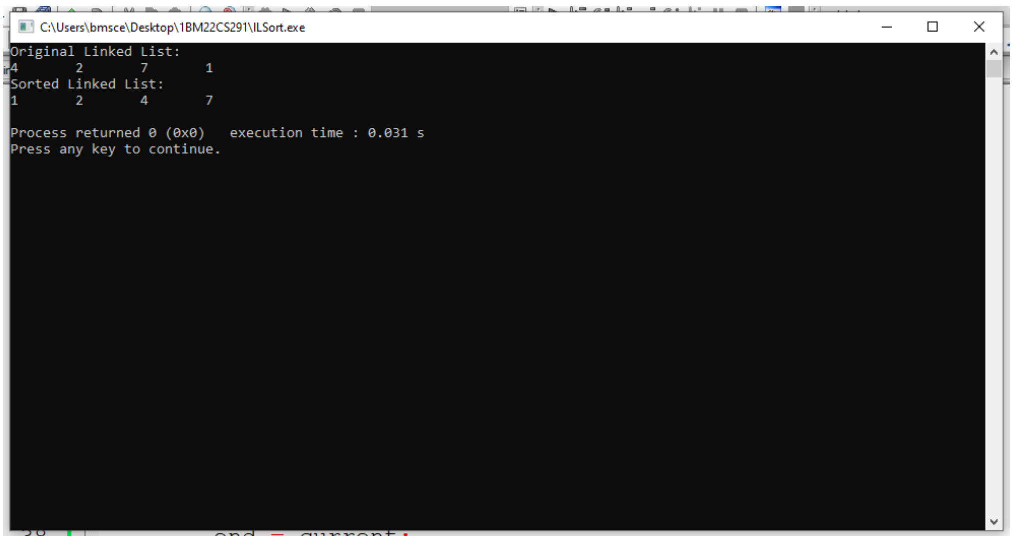


C:\Users\bmsce\Desktop\1BM22CS291\queueUsingLL.exe

```
Enter 1. Enqueue
2. Dequeue
3. -1 to stop
Enter operation:
2
Queue Underflow
Queue is empty
Enter operation:
1
Enter the element to enqueue
2
Queue elements are:     2
Enter operation:
1
Enter the element to enqueue
3
Queue elements are:     2       3
Enter operation:
1
Enter the element to enqueue
5
Queue elements are:     2       3       5
Enter operation:
2
Dequeued Element: 2
Queue elements are:     3       5
Enter operation:
2
Dequeued Element: 3
Queue elements are:     5
Enter operation:
-1
Execution stopped

Process returned 0 (0x0)   execution time : 11.357 s
Press any key to continue.
```

linked list sorting:

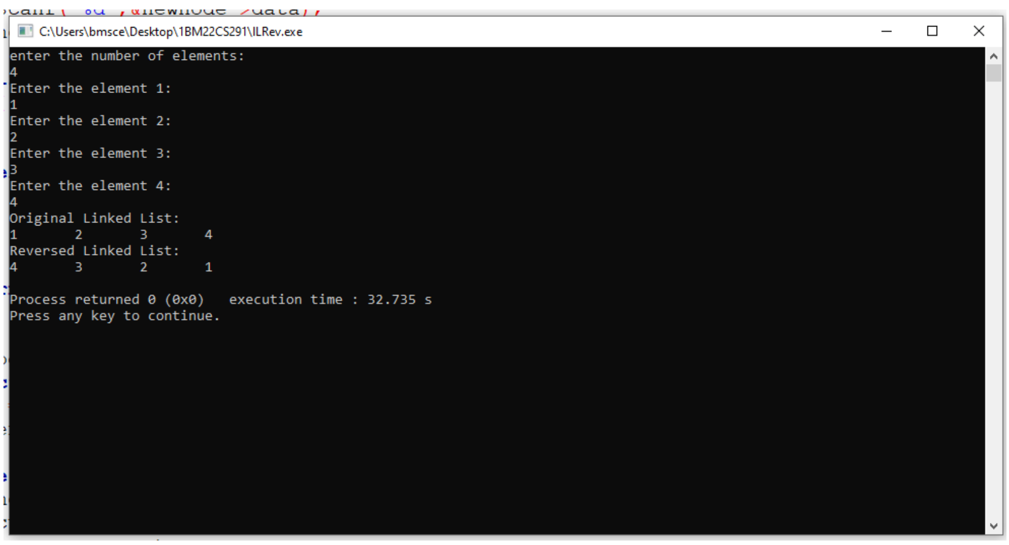


C:\Users\bmsce\Desktop\1BM22CS291\lLSort.exe

```
Original Linked List:
4       2       7       1
Sorted Linked List:
1       2       4       7

Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```

Linked lists reversing:

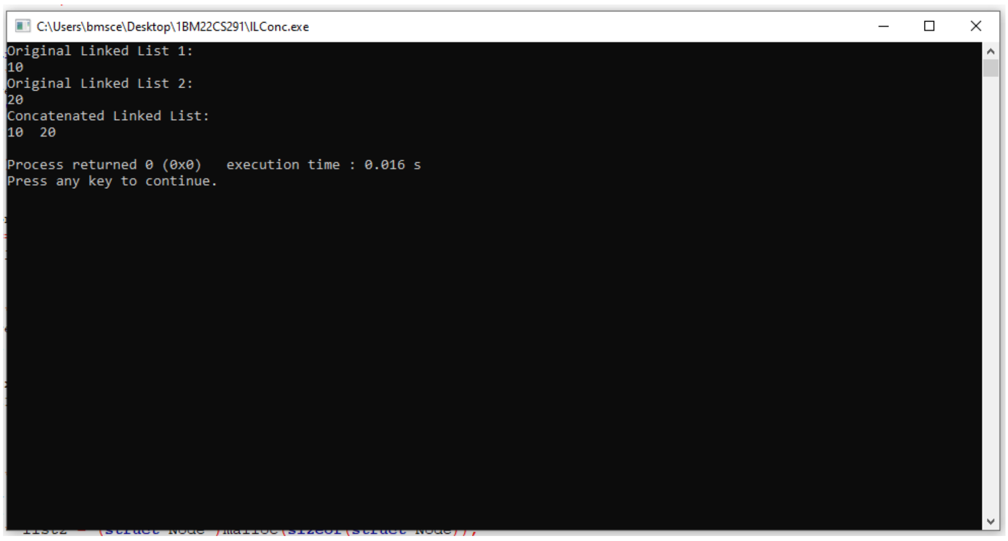


C:\Users\bmsce\Desktop\1BM22CS291\lLRev.exe

```
enter the number of elements:
4
Enter the element 1:
1
Enter the element 2:
2
Enter the element 3:
3
Enter the element 4:
4
Original Linked List:
1       2       3       4
Reversed Linked List:
4       3       2       1

Process returned 0 (0x0)   execution time : 32.735 s
Press any key to continue.
```

Linked lists concatenation:



C:\Users\bmsce\Desktop\1BM22CS291\lLConc.exe

```
Original Linked List 1:
10
Original Linked List 2:
20
Concatenated Linked List:
10  20

Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```