```
}
);
```

```
dequeue from an
emptyqueue.\n");
```

```
;
```

```
    struct Node * Current = queue->front;
    while (Current != Null){
            Printf ("%d <- ", current->data );
          Current = Current->next ;
    }
    Printf ("NULL \n");
}

Void freeQueue (struct Queue * queue) {
     While (!isEmpty(Queue)) {
           dequeue(queue);
     }
}
```

O/P:

```
Queue after enqueuing elements:
10<-20<-30<-NULL
Dequeued value: 10
Queue after dequeuing elements:
20<-30<-NULL
```

29.01.24

```
Void enqueue (struct Queue * queue, int value) {
        Struct Node * newNode = createNode (value);
        if (is empty (queue) {
            queue →front = newNode;
            queue → rear = newNode;
        } else {
            queue → rear → next = newNode;
            queue → rear = new Node;
        }
}

int dequeue (struct Queue * queue) {
        if (is Empty (Queue)) {
            printf ("Queue underflow. cannot dequeue from an
                                                    emptyque. \n");
        exit (1);
        }
        int dequeued value = que→front→data;
        Struct Node * temp = queue →front;
        if (queue→front = = queue→rear ) {
            queue →front = NULL;
            queue → rear = NULL;
        } else {
            queue →front = queue→front →next;
        }
        free (temp);
        return dequeued value;
}

Void displayQueue (struct Que * queue) {
    if (isEmpty (queue)) {
    printf ("Que is empty. \n");
    return;
    }
```

```
    struct Node * (
    while (current
        printf ("
        current = (
    }
    printf ("NULL \
}

Void freeQueue (st
    while (!isEmpt
        dequeue (q
    }
}
```

O/P:

```
Queue after enque
10 ← 20 ← 30 ← NULL
Dequeued value: 10
Queue after dequeu
20 ← 30 ← NULL
```

29.01.24

Implimentation of Queue using Linked List.

```
#include <Stdio.h>
# Include <Stdlib.h>

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

Struct Queue {
    Struct node * front;
    Struct node * rear;
};

Struct Node * CreateNode (int value ) {
      Struct Node* newNode = (Struct node*) malloc (size of(struct Node));
      if (newNode == NULL) {
          printf ("Memory allocation failed. \n");
          exit(1);
      }
      new Node->data = value;
      new Node->next = NULL;
      return newNode;

Void initialize~~[illegible]~~Queue (struct Queue * Queue) {
      queue->front = NULL;
      queue->rear  = NULL;
}
~~Void~~ int  isEmpty (struct Queue* queue) {
        return queue->front == NULL;

  }
```

```
    While (current != NULL) {
        Printf ("%d ->", current->data);
        Current = Current ->next;
    }
    Printf ("NULL \n");
}

Void freeStack (struct stack* stack) {
      While (!isEmpty(stack)) {
            pop (stack);
      }
}
```

O/P:

```
Stack after pushing elements:
30 -> 20 -> 10 -> NULL
poped value : 30
Stack after poping elements:
20 -> 10 -> NULL
Top value without popping: 20
```

Implimentation of

```
# include <Stdio
# include <Stdlib.

Struct Node {
   int data;
   Struct Node * n
};

Struct Queue {
     Struct Node *
     Struct Node *
};

Struct Node* (cre
     Struct Node
     if (newNode
        printf (
        exit(1);
     }
     new Node ->
     new Node ->
     return new

Void initialize~~Queue~~(
     queue ->fr
     queue -> rear
}
~~Void~~ int  isEmpty (
     return q
  }
*)Malloc (sizeof(struct
                Node));

");
```

```
void push(struct Stack* Stack, int value) {
    Struct Node* newNode = Create Node (value);
    newNode->next = Stack -> top;
    Stack->top = newNode;
}

int pop (struct stack * Stack) {
    if (isempty (stack)) {
        Printf ("Stack underflow. Cannot pop from an empty stack \n")
        exit(1);
    }.
    int popped value = Stack -> top ->data;
    Struct Node * temp = Stack ->top;
    Stack ->top = Stack-> top->nx;
    free (temp);
    return poped value;
}

int peek (struct stack * Stack) {
    if (is Empty (stack)) {
        print ("Stack is empty. \n").
        exit (1);
    }
    return stack ->top-> data;
}

void display Stack (struct stack * Stak) {
    if (is Empt(stack)) {
        printf ("Stack is empty .\n");
        return ;
    }
    Struct Node * Current = Stack ->top;
```

Stack implmentation using Linked List

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

Struct Node {
    int data ;
    Struct Node* next ;

Struct Stack {
    Stack node * top;
};

Struct node* CreateNode (int Value) {
        Struct Node * new Node = (struct Node *) Malloc (sizeof(struct
                                                              Node));
        if (newNode == NULL) {
            printf ("Memory allocation failed.\n");
            exit(1);
        }
        newNode -> data = value ;
        new Node -> next = NULL;
        return new Node;

Void initialize Stack (struct stack * Stack) {
        Stack -> top = NULL ;
}

int isEmpty (struct Stack * stack) {
        return stack -> top == NULL;
  }
```

```
Void push (struct Stack * Stac
    Struct Node* newNode
    new Node -> next = Sta
    Stack -> top = newNod
}

int pop (struct Stack
    if (isempty (stack
        Printf ("Stack i
        exit(1);
    }.
    int popped value = St
    Struct Node * temp = S
    Stack -> top = Stack ->
    free (temp);
    return popped value;
}

int peek (struct stack *
    if (is Empty (stac
        Print ("Stack i
        exit (1);
    }
    return Stack -> top -> d
}

Void display Stack (struct
    if (is Empt (stack)
        print ("Stack is
        return ;
    }
    Struct Node * Curre
```

4) {

8 9 11 33 14 2

33

reverse {

```
void reverseList (struct Node ** headRef) {
    Struct Node *Prev, *Current, * next Node;
    prev = Null,
    Current = * headRef;

    While (Current != Null) {
        next Node ⇒ Current→next;
        Current→next = prev;
        prev = Current;
        Current = next Node;
    }
    * headRef = Prev;
}.
```

o/p :

```
Enter data to insert into list : 1 2 3 4 5
List reversed
reversed List : 5 4 3 2 1
```

Sorting [Bubble sort].

```
Void SortList(Struct Node * * head) {
      Struct Node * Current, * nextNode;
      int temp;
      Current = * head;
      While (Current != NULL) {
      nextNode = Current -> next;
       While (nextNode != NULL) {
            if (Current -> data > nextNode -> data) {
               temp = Current -> data;
               Current -> data = nextNode -> data;
               nextNode -> data = temp;
            }
           nextNode = nextNode -> next;
        }
        Current = Current -> next;
    }
}
```

o/p :

```
Enter data to insert into List : 1 3 6 5 4 8 9 11 33 14 2
List sorted
sorted list : 1 2 3 4 5 6 8 9 11 14 33
```

reverse {

```
Void reverseList (St
      Struct Node *Pr
      prev = NULL;
      Current = * head

      While (Current !=
           nextNode =
           Current -> nex
           prev = Curren
           Current = 
      }
      * headRif = Prev;
}.
```

o/p :

```
Enter data to insert
List reversed
reversed List : 5
```

```
Malloc(sizeof(
        Structnode));
;

1, Struct Node* list2){
```

```
        Struct Node* Current = list1;
        while (Current -> next != NULL){
            Current = Current -> next;
        }
        Current -> next = list2;
        return list1;
}

void freelist(Struct Node * head){
    Struct Node* Current = head;
    Struct Node * Next Node;

    While (Current != NULL) {
        next Node = Current -> next;
        free(current);
        Current = next Node;
    }
}
```

O/P

```
First Linked List:
1->2->3->NULL
Second Linked List:
4->5->NULL
Concatenated Linked List:
1->2->3->4->5->NULL
```

Lab 29/01/24. [Lab 6]

[Concatination]

```
#include <Stdio.h>
# include <Stdlib.h>

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

Struct Node * Create Node (int value ) {
     Struct Node * newNode = (Struct Node *) Malloc (sizeof(
                                               Struct node));
     if (newNode == NULL) {
        printf ("Memory allocation failed \n");
       exit (1);
     }
     NewNode → data = value ;
     new Node → next = NULL ;
     return newNode ;

Void displayList (struct Node*head) {
     Struct Node* Current = head ;
     While (current =! NULL) {
        printf ("%d → ", current → data);
         Current = current → next ;
      }
      Print f ("NULL \n");

Struct Node * ConcatenateLists (Struct Node* list1, Struct Node* list2) {
           if (list1 == NULL) {
               return list2;
           }
```

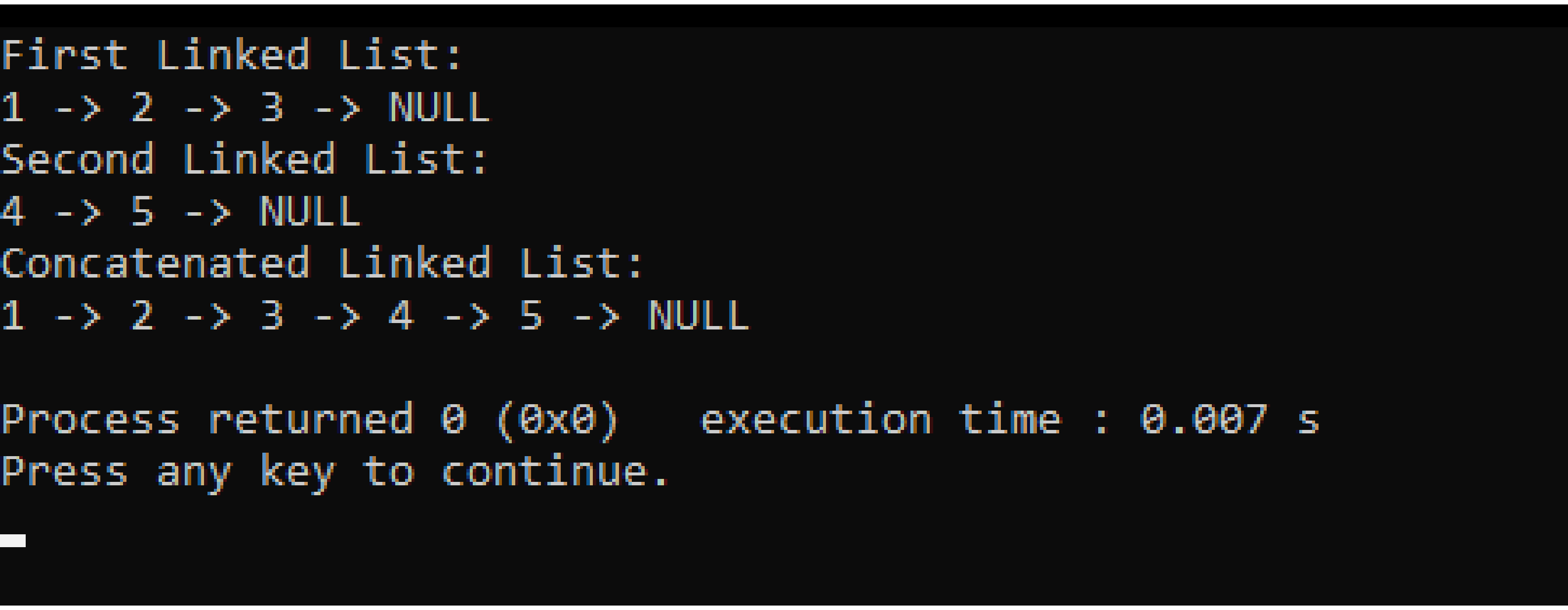
```
       Struct
       while (
           C
         }
       Current
       return
}

Void freeList(Str
    Struct Node
    Struct Node

    While (Curre
       nextN
       free(
       Curren
    }
}
```

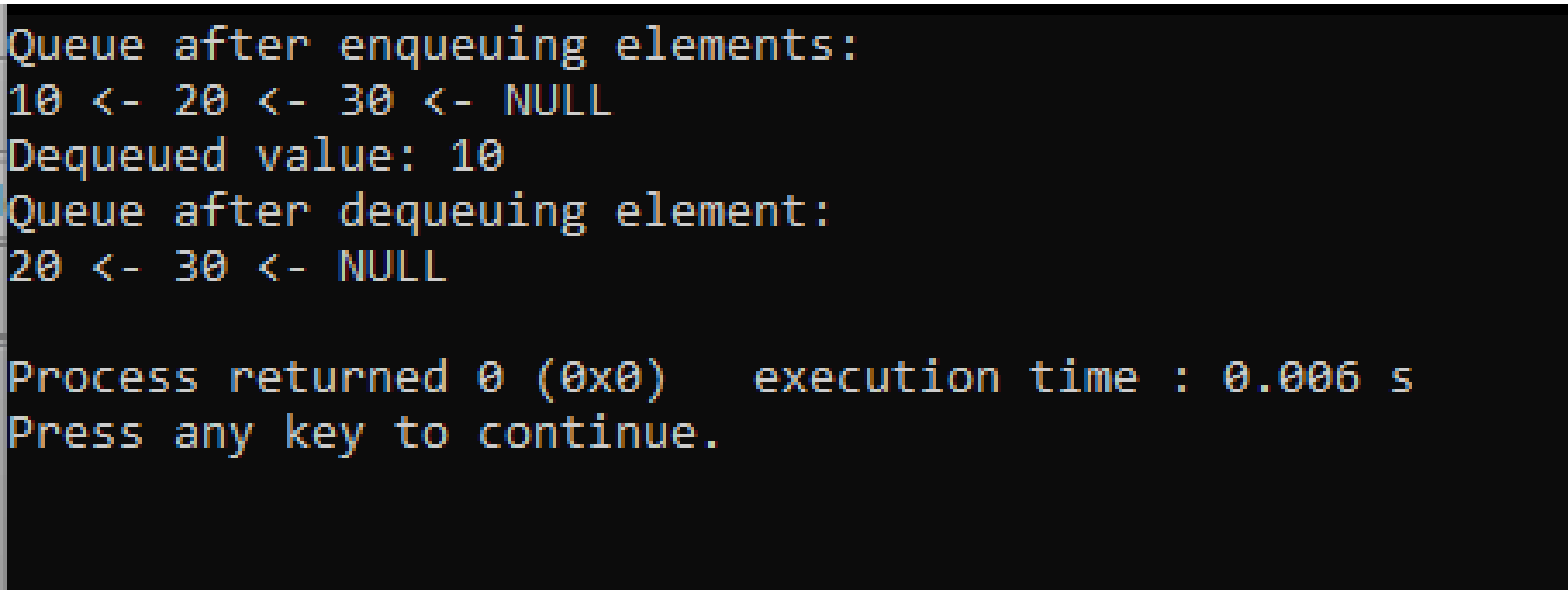
O/P

```
First Linked L
1→2→3→NULL
Second Linked 
4→5→NULL.
Conlatenated Linke
1→2→3→4→5→
```



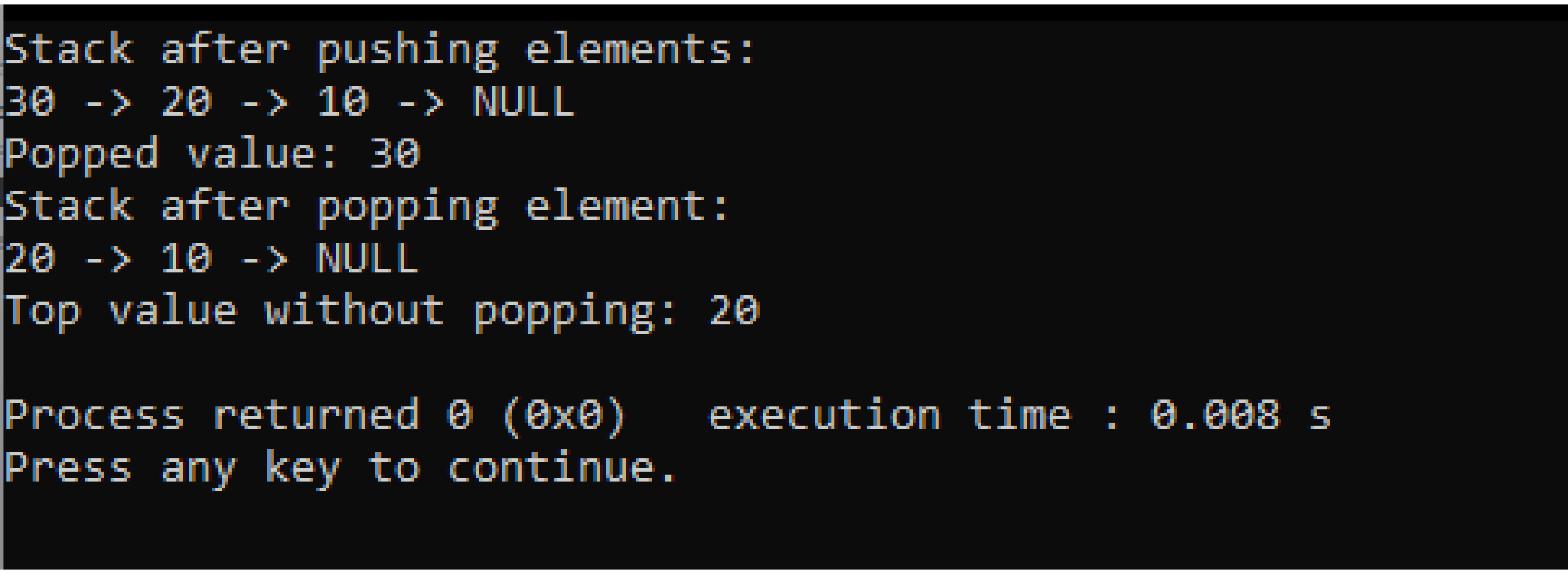
```
First Linked List:
1 -> 2 -> 3 -> NULL
Second Linked List:
4 -> 5 -> NULL
Concatenated Linked List:
1 -> 2 -> 3 -> 4 -> 5 -> NULL

Process returned 0 (0x0)   execution time : 0.007 s
Press any key to continue.
```



```
Queue after enqueuing elements:
10 <- 20 <- 30 <- NULL
Dequeued value: 10
Queue after dequeuing element:
20 <- 30 <- NULL

Process returned 0 (0x0)   execution time : 0.006 s
Press any key to continue.
```



```
Stack after pushing elements:
30 -> 20 -> 10 -> NULL
Popped value: 30
Stack after popping element:
20 -> 10 -> NULL
Top value without popping: 20

Process returned 0 (0x0)   execution time : 0.008 s
Press any key to continue.
```