```
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.
```

```
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.
```

```
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.
```

```
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.
```

```
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.
```

```
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("Execution stopped \n");
        break;
    }

    switch(op){
    case 1:
        printf("Enter the element to enqueue \n");
        scanf("%d", &n);
        enqueue(&front, &rear, n);
        break;
    case 2:
        dequeued Element = dequeue(&front, &rear);
        if (dequeued Element != -1){
            printf("Dequeued Element: %d \n", dequeued
                                                Element);
        }
        break;
    }
    display(front);
}
return 0;
}
```

```
        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 dequeuedData = temp->data;

    *front = (*front)->next;
    if (*front == NULL){
        *rear = NULL;
    }
    free(temp);
    return dequeuedData;
}

int main(){
    int ch, x, dequeuedElement;
    struct Node* front = NULL;
    struct Node* rear = NULL;
```

```
        }
    }
    display(top);
}
return 0;
}
```

(b) Queue implementation using 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");
}

void enqueue(struct Node** front, struct Node** rear, int data){
    struct Node* newNode = (struct Node*) malloc(sizeof(struct Node));
    if(newNode==NULL){
```

```
struct Node * pop(struct Node * top, int * poppedData){
 if (top == NULL){
    printf("Stack Underflow\n");
    *poppedData = -1;
    return NULL;
 }
 struct Node * temp = top;
 *poppedData = temp->data;
 top = top->next;
 free(temp);
 return top;
}
int main(){
   int ch, n, poppedElement;
   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", &ch);

      if(ch == -1){
         printf("Execution stopped\n");
         break;
      }
      switch(ch){
         case 1:
          printf("Enter the element to push\n");
          scanf("%d", &n);
          top = push(top, n);
          break;
         case 2:
          top = pop(top, &poppedElement);
          if(poppedElement != -1){
              printf("Popped Element: %d\n", poppedElement);
```

classmate
Date
Page

(8) Stack implementation using linked list :-

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

SLL - Queues, Stacks, Strings
Emded

29/1/24

```
printf("Original Linked List 2:\n");
display(list2);

struct Node* concatenatedList = concatenateLinkedLists(list1, list2);

printf("Concatenated Linked List:\n");
display(concatenatedList);

free(list1);
free(list2);

return 0;
```

Demonstration

29/11/24

```
        struct Node* next;
};
void display(struct Node *head) {
    struct Node* current = head;
    while (current != NULL){
        printf("%d ->", current->data);
        current = current->next;
    }
    printf("NULL\n");
}
struct Node* concatenateLinkedLists(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 Node*)malloc(sizeof(struct Node));
  list1->data = 10;
  list1->next = NULL;

  struct Node* list2 = (struct Node*)malloc(sizeof(struct Node));
  list2->data = 20;
  list2->next = NULL;

  printf("original LinkedList1:\n");
  display(list1);
```

```
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* reversedHead = reverse Linked List (node1);

printf ("Reversed Linked List : \n");
display (reversed Head);

free (node1);
free (node2);
free (node3);

return 0;
}
```

4) Concatenate :-

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

struct Node {
    int data;
```

```
    free(node4);
    return 0;
}
```

4 Reverse

```
#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 ->", current->data);
        current = current->next;
    }
    printf ("NULL\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() {
```

```
        current->next->data = temp_data;
        swapped = 1;
      }
      current = current->next;
    }
    end = current;
  } while (swapped);

  return head;
}

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));
    struct Node* node4 = (struct Node*) malloc (sizeof (struct Node));

    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 (node2);
    free (node3);
```

29.1.2024

7 SLL → sort, reverse, concantination

sort :-

```
#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("NULL\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;
```