29/01/24 Sort Operation on Single Linked List

Q7).

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
  int data;
  struct Node* next;
};
void insertNode (struct Node** head, int newData)
{
 struct Node* newNode =(struct Node*)malloc (sizeof (struct Node));
 newNode->data = newData;
 newNode->next = NULL;
 if(*head == NULL)
 {
   *head =newNode;
 }
 else
 {
   struct Node * temp = *head;
   while(temp->next != NULL)
   {
     temp = temp->next;
   }
   temp-> next = newNode;
 }
}
void printList (struct Node * head)
{
 struct Node* temp = head;
 while (temp != NULL)
 {
   printf ("%d", temp->data);
```

```
    temp = temp -> next;
  }
  printf ("\n");
}
void bubbleSort (struct Node* head)
{
  int swapped, i;
  struct Node * ptr;
  struct Node * lptr = NULL;
  if (head == NULL)
  {
    return;
  }
  do
  {
    swapped = 0;
    ptr = head;
    while (ptr -> next != lptr)
    {
      if (ptr -> data > ptr -> ~~data~~ next -> data)
      {
        int temp = ptr -> data;
        ptr -> data = ptr -> ~~data~~ next -> data;
        ptr -> next -> data = temp;
        swapped = 1;
      }
      ptr = ptr -> next;
    }
    lptr = ptr;
  }
  while (swapped);
}
int main ()
{
```

```
struct Node* head = NULL;
insertNode(&head, 5);
insertNode(&head, 2);
insertNode(&head, 8);
insertNode(&head, 1);
insertNode(&head, 3);
printf("Original Linked List:");
printList(head);
bubbleSort(head);
printf("Sorted Linked List:");
printList(head);
return 0;
}
```

O/P:

Original Linked List: 5 2 8 1 3

Sorted Linked List: 1 2 3 5 8

```
Original Linked List: 5 2 8 1 3
Sorted Linked List: 1 2 3 5 8

Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

8) Reverse Operation on Single Linked List

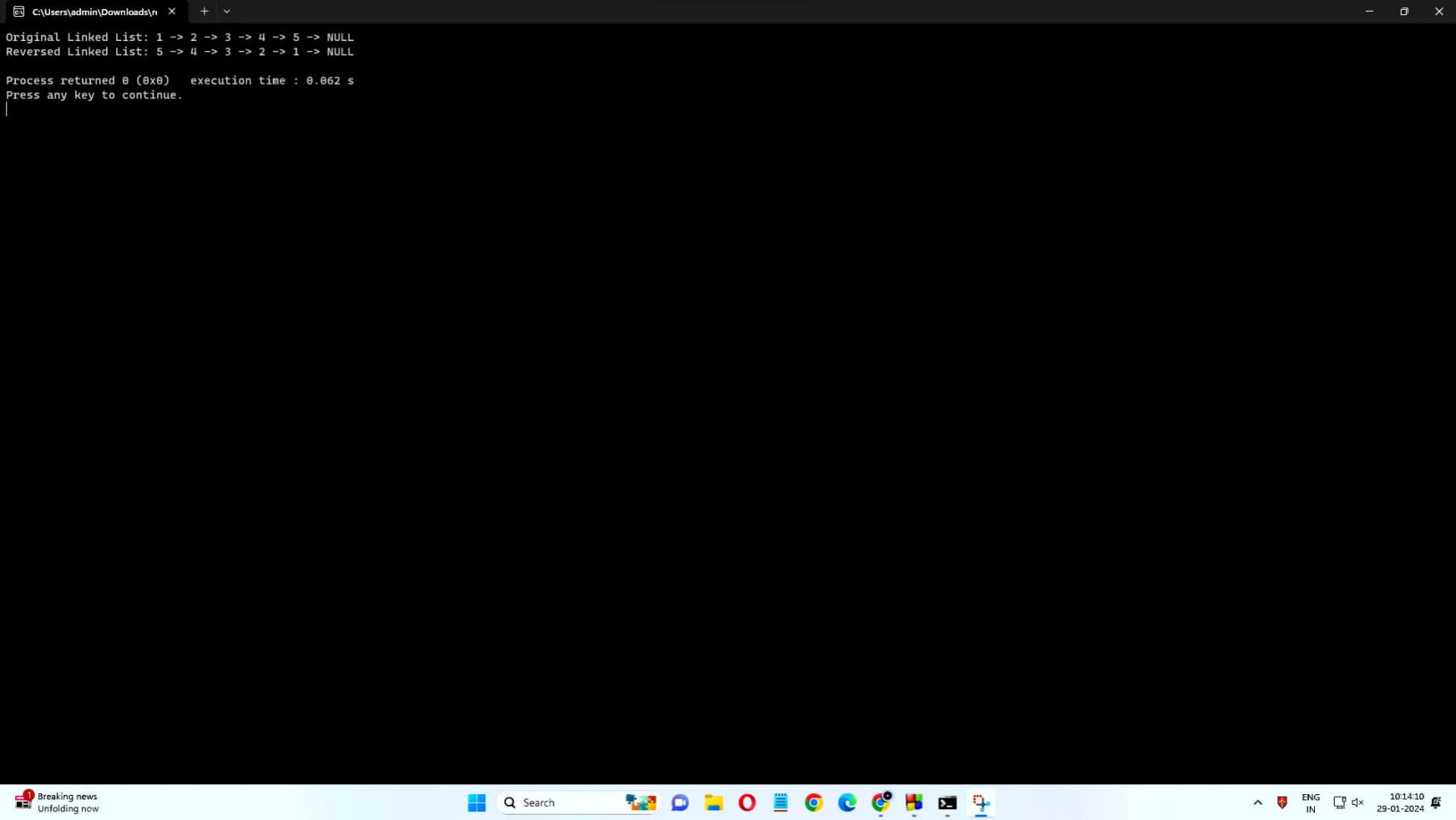
```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
  int data;
  struct Node* next;
};
struct Node* createNode (int value)
{
  struct Node* newNode = (struct Node*) malloc (sizeof (struct
Node));
  if (newNode == NULL)
  {
    printf ("Memory allocation failed \n");
    exit(1);
  }
  newNode -> data = value;
  newNode -> next = NULL;
  return newNode;
}
void insertEnd (struct Node** head, int value)
{
    struct Node* temp = (struct Node*) malloc (sizeof( struct Node));
    temp -> data = data;
    temp -> next = NULL;
    return temp;
}
int main()
{
   struct Node * head = newNode (1);
   head -> next = newNode (2);
   head -> next -> next = newNode (3);
   head -> next -> next -> next = newNode (4);
```

```
head->next->next->next->next = newNode(5);
printf("Original linked list:");
printList(head);
head = reverseLinkedList(head);
printf("Reversed linked list:");
printList(head);
return 0;
}
```

O/P

Original Linked List: 1 → 2 → 3 → 4 → 5 → NULL

Reversed Linked List: 5 → 4 → 3 → 2 → 1 → NULL


C:\Users\admin\Downloads\n
Original Linked List: 1 -> 2 -> 3 -> 4 -> 5 -> NULL
Reversed Linked List: 5 -> 4 -> 3 -> 2 -> 1 -> NULL
Process returned 0 (0x0)   execution time : 0.062 s
Press any key to continue.
Breaking news
Unfolding now
Search
ENG
IN
10:14:10
29-01-2024

Concatenation Operation on Single Linked List

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
  int data;
  struct Node* next;
};
struct Node* createNode(int data)
{
  struct Node* newNode = (struct Node*) malloc(sizeof(struct Node));
  newNode->data = data;
  newNode->next = NULL;
  return newNode;
}
void displayList(struct Node* head)
{
  struct Node* current = head;
  while(current != NULL)
  {
    printf("%d ->", current->data);
    current = current->next;
  }
  printf("NULL\n");
}
struct Node* concatenateLists(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 = createNode(1);
  list1->next = createNode(2);
  list1->next->next = createNode(3);
  struct Node* list2 = createNode(4);
  list2->next = createNode(5);
  list2->next->next = createNode(6);
  printf("First Linked List: ");
  displayList(list1);
  printf("Second Linked List: ");
  displayList(list2);
  struct Node* concatenatedList = concatenateLists(list1, list2);
  printf("Concatenated Linked List: ");
  displayList(concatenatedList);
  return 0;
}
```

O/P:

First Linked List: 1 -> 2 -> 3 -> NULL
Second Linked List: 4 -> 5 -> 6 -> NULL
Concatenated Linked List: 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> NULL

C:\Users\admin\Downloads\c

```
First Linked List: 1 -> 2 -> 3 -> NULL
Second Linked List: 4 -> 5 -> 6 -> NULL
Concatenated Linked List: 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> NULL

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

8). Stack implementation using single linked list.

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
  int data;
  struct Node* next;
};
struct Node * createNode (int data)
{
  structNode* newNode = (struct Node*)malloc (sizeof (struct Node));
  if (newNode ==NULL)
  {
    printf ("Memory allocation failed\n");
    exit (Exit_Failure);
  }
  newNode -> data =data;
  newNode -> next= NULL;
  return newNode;
}
struct Node * pop( Struct Node *top)
{
  if (top == NULL)
  {
    printf ("Stalk underflow = cannot pop \n");
    return NULL;
  }
  struct Node *temp=top;
  top = top -> next;
  free (temp);
  return top;
}
```

```
void displaystack(struct Node* top)
{
  printf("Stack: ");
  while (top!=NULL)
  {
    printf("%d", top->data);
    top = top->next;
  }
  printf("\n");
}
void freestack(struct Node* top)
{
  while (top!=NULL)
  {
    struct Node* temp = top;
    top = top->next;
    free(temp);
  }
}
int main()
{
  struct Node* top = NULL;
  int choice, data;
  do
  {
    printf("\n Menu \n");
    printf("1. Push \n");
    printf("2. Pop \n");
    printf("3. Display \n");
    printf("4. Exit \n");
    printf("Enter your choice: ");
    scanf("%d", &choice);
    switch(choice)
    {
```

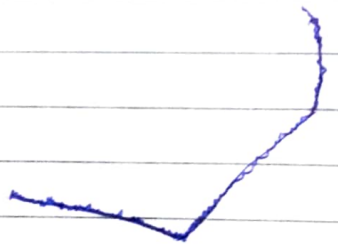
```
case 1:
    printf("Enter data to push");
    scanf("%d", &data);
    top = push(top, data);
    break;

case 2:
    top = pop(top);
    break;

case 3:
    display stack(top);
    break;

case 4:
    printf("Entering the program \n");
    break;

default:
    printf("Invalid Choice");
  }
}
while(choice != 4)
free stack(top);
return 0;
}
```



O/P:

```
Menu                    -> Enter your choice: 1
1. Push                    Enter data to push: 6
2. Pop                     Enter your choice: 2
3. Display                 Popped successfully
4. Exit                    Enter your choice: 1
                           Enter data to push: 7
                           Enter your choice: 3
                           7 6
```

C:\Users\admin\Downloads\s

```
10 pushed to the stack.
20 pushed to the stack.
30 pushed to the stack.
Stack elements: 30 20 10
Top element: 30
Popped element: 30
Stack elements: 20 10

Process returned 0 (0x0)   execution time : 0.062 s
Press any key to continue.
```

9. Queue Implementation 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 data)
{
  struct Node* newNode = (struct Node*) malloc (size of (
  struct Node);
if (newNode = NULL)
  {
    printf ("Memory allocation failed ");
    exit ( Exit_Failure);
  }
  newNode → data = data;
  newNode → next = Null;
  return newNode;
}
struct Queue * initialize Queue ()
{
  struct Queue * queue = ( struct queue *) malloc (size of (struct queue));
  if (queue == NULL)
  {
    printf (" Memory allocation failed ");
    exit (Exit_Failure);
```

```
}
queue -> front = queue -> rear = NULL;
return queue;
}
void enqueue (struct Queue* queue, int data)
{
struct Node *newNode = createNode (data);
if (queue -> rear = NULL)
{
queue -> front = queue -> rear = newNode;
return;
queue -> rear -> next = newNode;
queue -> rear = newNode;
}
void dequeue (struct Queue * queue)
{
if (queue -> front == NULL)
{
printf ("Underflow cannot dequeue \n");
return;
}
struct Node * temp = queue -> front;
queue -> front = queue -> front -> next;
if ( queue -> front == NULL)
{
queue -> rear = NULL};
}
free (temp);
}
void display queue (struct queue * queue)
{
if ( queue -> front == NULL)
{
printf (" Queue is empty \n");
```

```
    return;
}
struct Node* current = queue->front;
printf("Queue:");
while(current != NULL)
{
    printf("%d", current->data);
    current = current->next;
}
printf("\n");
}
int main()
{
struct Queue* queue = initializeQueue();
int choice, data;
do
{
    printf("Menu");
    printf("1. Enqueue\n");
    printf("2. Dequeue\n");
    printf("3. Display\n");
    printf("4. Exit\n");
    printf("Enter your choice");
    scanf("%d", &choice);
    switch(choice)
    {
    case 1:
        printf("Enter data to enqueue:");
        scanf("%d", &data);
        enqueue(queue, data);
        break;
    case 2:
        dequeue(queue);
        break;
```

```
        case 3:
            displayQueue(queue);
            break;
        case 4:
            printf("Exiting the program\n");
            break;
        default:
            printf("Invalid choice!");
    }
    while(choice != 4)
    }
}
return 0;
}
```

O/P:

Menu
1. Enqueue
2. Dequeue
3. Display
4. Exit

Enter your choice : 3
Queue is Empty
Enter your choice : 1
Enter data to enqueue : 4
Enter your choice : 1
Enter data to enqueue : 6
Enter your choice : 3
4, 6

C:\Users\admin\Downloads\q

```
Queue: 10 20 30
Dequeued element: 10
Queue: 20 30

Process returned 0 (0x0)   execution time : 0.062 s
Press any key to continue.
```