LAB - 4

29/1/24

1) sort, Reverse and concatenate the given ll

```
#include<stdio.h>
#include<stdlib.h>
struct Node {
    int data;
    struct Node *next;
};
void insertAtBeginning (struct Node * headRef, int data)
{
  struct Node *newNode = (struct Node*) malloc (sizeof ( struct node));
  newnode->data = data;
  newNode->next = *headRef;
  *headRef = newNode;
  void printList (struct Node *head) {
    while (head != NULL) {
      printf ("%d", head->data);
      head = head->next;
    }
    printf ("\n");
}
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;
}
uct node]);
}

Void reverselist (struct Node *headRef) {
    Struct Node *prev, *current, *nextNode;
    prev = NULL;
    current = *headRef;
    while (current != NULL) {
        nextNode = current->next;
        current->next = prev
        prev = current;
        current = nextNode;
    }
    *headRef = prev;
}
```

```
void concatenation (struct Node *list1, struct Node *list2
    if (*list1 == NULL) {
        *list1 = list2;
    }
    struct Node *temp = *list1;
    while (temp -> next1 = NULL) {
        temp = temp -> next;
    }
    temp -> next = list2;
}

int main() {
    struct Node * list1 = NULL;
    struct Node * list2 = NULL;
    int choice;
    int data;
    while(1)
    {
    printf("\n 1. Insert into list1 \n 2. Insert into list2 \n
            3. sort list1 \n 4. Reverse list2 \n
            5. concatenate lists \n 6. print list \n 7. exit");
    printf("Enter your choice : ");
    scanf("%d", &choice);
    switch (choice) {
```

```
case 1 : printf("Enter data (list 1) ");
         scanf("%d", &data);
         insertAtBeginning(&list1, data);
         break;
case 2 : printf("Enter data to insert (list 2)");
         scanf("%d", &data);
         insertAtBeginning(&list2, data);
         break;
case 3 : sortList(&list1);
         printf("List 1 sorted.\n");
         break;
case 4 : reverseList(&list1);
         printf("List 1 reversed\n");
         break;
case 5 : concatenation(&list1, list2);
         printf("lists concatenated\n");
         break;
case 6 : printf("List 1 : ");
         printList(list1);
         printf("List 2 : ");
         printList(list2);
         break;
case 7 : exit(0); break;
default : printf("Invalid choice. Try again");
    }
}
return 0;
```

2a) stack implementation using 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));
    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        exit(Exit_Failure);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
struct Node *push(struct Node *top, int data) {
    struct Node *newNode = createNode(data);
    newNode->next = top;
    return newNode;
}
struct Node *pop(struct Node *top) {
    if (top == NULL) {
        printf("stack 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 2. Pop \n 3. Display \n 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 : displaystack (top) ;
              break ;
   case 4 : printf (" ending the program \n ");
              break ;
   default : printf (" Invalid choice ")
  }
 }
while (choice != 4) ;
freestack (top) ;
return 0 ;
}
```

26) 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 * initializequeue() {
    struct Queue * queue = (struct Queue *) malloc (sizeof
                                        (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 deque (struct queue * queue) {
    if (queue->front == NULL) {
        printf("underflow. can not deque\n");
        return;
    }
    struct Node * temp = queue->front;
```

```
    queue->front = queue->front->next;
    if (queue->front == NULL)
        queue->rear = NULL;
    free(temp);
}
void displayqueue(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("main menu\n");
        Printf("1. Enqueue\n 2. Dequeue\n 3. Display\n
                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");
             break;
    default : printf("Invalid choice!");

}
while(choice != 4)

return 0;
}
```

output for program 1 (concatenation, reverse, sort)

1. Insert into list 1
2. Insert into list 2
3. sort list 1
4. Reverse list 2
5. concatenate lists
6. print lists
7. Exit

→ Enter your choice : 1
Enter data to insert (list 1) : 10

→ Enter your choice : 2
Enter data to insert (list 2) : 20

→ Enter your choice : 3
list 1 sorted

→ Enter your choice : 1
Enter data to insert (list 1) : 6

→ Enter your choice : 3
list 1 sorted

→ Enter your choice : 6
List1 : 6 10
List2 : 20

→ Enter your choice : 5
Lists concatenated

→ Enter your choice : 6
List 1 : 6 10 20

2(a) output for stack implementation using LS

Menu
1. Push
2. POP
3. Display
4. Exit

→ Enter your choice : 1
Enter data to push : 60

→ Enter your choice : 1
Enter data to push : 70

→ enter your choice : 3
60 70

→ Enter your choice : 2
poped succesfully

→ Enter your choice : 3
10

2. b) Output for queue implementation using u

Main menu
1. Enqueue
2. Dequeue
3. Display
4. Exit

→ Enter your choice : 1
Enter data to Enqueue : 10

→ Enter your choice : 1
Enter data to Enqueue : 20

→ Enter your choice : 3
10 20

→ Enter your choice : 2
→ Enter your choice : 3
20

30.01.24