Lab - 6

1. WAP to Implement Single linked list with following operations Sort the linked list, Reverse the linked list, Concatenation of two linked list.

```
#include <stdio.h>
#include <stdlib.h>
Struct Node {
        int data;
        Struct Node* next;
      };
Void append (Struct Node** head_ref, int new_data){
    Struct Node* new_node = (Struct Node*) malloc (sizeof
                            (Struct Node));
    Struct Node* last = *head_ref;
    Struct Node* new_node -> data = new_data;
                 new_node -> next = NULL;
    if (*head_ref == NULL) {
        *head_ref = new_node;
         return;
      }
    While (last -> next != NULL) {
        last = last -> next;
     }
    last -> next = new_node;
  }
Void PrintList (Struct Node* node) {
     while (node != NULL) {
        Printf ("%d -> ", node -> data);
        node = node -> next;
     }
    Printf ("NULL\n");
}
Void SortList (Struct Node** head_ref) {
        if (*head_ref == NULL) {
            return;
         }
    int Swapped, temp;
    Struct Node* ptr1;
    Struct Node* lPtr = NULL;
```

Scanned with OKEN Scanner

```
do {
    swapped = 0;
    Ptr1 = *head_ref;
while (Ptr -> next != lPtr) {
    if (Ptr1 -> data > Ptr1 -> next -> data) {
        temp = Ptr1 -> data;
        Ptr1 -> data = Ptr1 -> next -> data;
        Ptr1 -> next -> data = temp;
        swapped = 1;
    }
    Ptr = Ptr1 -> next;
}
lPtr = Ptr1;
} while (swapped);
}
void reverseList (Struct Node** head_ref) {
    Struct Node* Prev = NULL;
    Struct Node* current = *head_ref;
    Struct Node* next = NULL;
while (current != NULL) {
    next = current -> next;
    current -> next = Prev;
    Prev = current;
    current = next;
}
*head_ref = Prev;
}
void concatenateLists (Struct Node** head1,
        Struct Node* head2) {
    if (*head1 == NULL) {
        *head1 = head2;
        return;
    }
    Struct Node* temp = *head1;
    while (temp -> next != NULL) {
        temp = temp -> next;
    }
    temp -> next = head2;
}
```

25/11/24

Scanned with OKEN Scanner

```
int main() {
    struct Node* list1 = NULL;
    struct Node* list2 = NULL;
    int n, data;
    printf("Enter the number of elements for list 1:");
    scanf("%d", &n);
    printf("Enter the elements for list 1:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &data);
        append(&list1, data);
    }
    printf("Enter the number of elements for list 2:");
    scanf("%d", &n);
    printf("Enter the elements for List 2:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &data);
        append(&list2, data);
    }
    printf("\n Original List 1:");
    printList(list1);
    printf("Original List 2:");
    printList(list2);
    sortList(&list1);
    sortList(&list2);
    printf("\n sorted List 1:");
    printList(list1);
    printf("Sorted List 2:");
    printList(list2);
    concatenateLists(&list1, list2);
    printf("\n Concatenated List:");
    printList(list1);
    printf("\n Reversed List:");
    printList(list1);
    return 0;
}
```

Scanned with OKEN Scanner

Output:

```
Enter the number of Element for list 1 : 3
Enter the Elements for List 1:
10
20
30
Enter the number of Elements for list 2 : 3
Enter the Elements for List 2:
40
50
60
Original List 1: 10 → 20 → 30 → NULL
Original List 2: 40 → 50 → 60 → NULL

Sorted List 1 : 10 → 20 → 30 → NULL
Sorted List 2 : 40 → 50 → 60 → NULL

Concatenated List : 10 → 20 → 30 → 40 → 50 → 60 → NULL
Reverse List : 60 → 50 → 40 → 30 → 20 → 10 → NULL
```

Program - 2

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
void Push (struct Node** top_ref, int new_data) {
    struct Node* new_node = (struct Node*) malloc
        (sizeof (struct Node));
    new_node -> data = new_data;
    new_node -> next = *top_ref;
    *top_ref = new_node;
}
int Pop (struct Node** top_ref) {
    if (*top_ref == NULL) {
        printf ("Stack is Empty. Cannot Pop.\n");
        return -1;
    }
```

```
Struct Node* temp = *top_ref;
int PoPped_data = temp -> data;
 * top_ref = temp -> next;
    free (temp);
  return Popped_data;
}
void Print Stack (struct Node * top) {
      Printf ("Stack :");
      While (top != NULL) {
      Printf ("%d -> ", top -> data);
       top = top -> next;
    }
     Print f ("NULL \n");
  }
int main () {
   Struct Node *   Stack TOP = NULL;
   int choice, data;
do {
     Printf ("\n Stack Menu: \n");
     Printf ("1. Push \n");
     Printf ("2. Pop \n");
     Printf ("3. Print Stack \n");
     Print f ("4. Exit \n");
     Print f (" Enter your choice: ");
    Scanf ("%d", &choice);
Switch (choice) {
     Case 1:
        Print f ("Enter the Element to Push:");
        Scanf ("%d", &data);
        Push (&stack Top, data);
        Printf ("Element %d Pushed onto the Stack.)
        break;
    Case 2:
       data = Pop (&Stack Top);
        if (data != -1) {
          Printf ("Element %d Popped from the
                  Stack . \n", data);
        }
       break;
```

Scanned with OKEN Scanner

Week -6
Lab - 6
Prg - 2

```
Case 3:
    PrintStack (StackTop);
    break;

Case 4:
    Printf ("Exiting the Program.\n");
    break;

default:
    Printf ("invalid choice. Please Enter a valid
            Option \n");
    }
} while (choice != 4);

While (StackTop != NULL) {
    Struct Node * temp = StackTop;
    StackTop = StackTop -> next;
    free (temp);
}
return 0;
}
```

Output: Stack Menu:

1. Push
2. POP
3. Print Stack
~~3. Print Stack~~
4. Exit

'Enter your choice: 1
Enter the element to PUSH: 10
Element 10 pushed onto the stack

Scanned with OKEN Scanner

Enter Your choice - 1
Element Enter the Element to Push: 20
Element 20 Pushed onto the stack.
Enter your choice - 1
Enter the Element to Push: 30
Element- 30 Pushed onto the stack
Enter your Choice - 2
Enter the Element 30 Popped from Pushed onto the Stack.
Enter your choice - 3
Stack : 20 → 10 → NULL
Enter your Choice 4
Exit.

24/Jan/2024

Week - 6

Lab - 6

Program - 3

```
#include <stdio.h>
#include <stdlib.h>
Linked
Struct Node {
    int data;
    Struct Node * next;
};
Element into the Queue
Void Enqueue (Struct Node ** front_ref,
Struct Node ** rear_ref, int new_data)
{
  Struct Node * New_node = (Struct Node*)
  malloc (sizeof (Struct Node));
  new_node -> data = new_data;
  new_node -> next = NULL;

if (* rear_ref == NULL ) {
  * front_ref = * rear_ref = new_node;
} else {
    (* rear_ref) -> next = new_node;
    * rear_ref = new_node;
}
}
from int dequeue (Struct Node ** front_ref,
  Struct Node ** rear_ref) {
  if (* front_ref == NULL) {
  Printf ("Queue is Empty. Cannot dequeue \n");
  return -1;
}
```

```
struct Node * temp = *front_ref;
int dequeued_data = temp->data;
*front_ref = temp->next;
if (*front_ref == NULL)
    *rear_ref = NULL;
}
free(temp);
return dequeued_data;
}
void PrintQueue (struct Node *front) {
    Printf("Queue:");
    while (front != NULL) {
        Printf ("%d ->", front->data);
        front = front->next;
    }
    Printf ("NULL \n");
}
int main() {
    struct Node * QueueFront = NULL;
    struct Node * QueueRear = NULL;
    int choice, data;
```

25/11/24

```
do {
    Printf ("\n Queue Menu : \n");
    Printf ("1. Enqueue \n");
    Printf ("2. Dequeue \n");
    Printf ("3. Print Queue \n");
    Printf ("4. Exit \n");
    Printf ("Enter your choice: ");
    Scanf ("%d", &choice);
```

```
switch (choice){
  case 1:
    Printf ("Enter the Element to Enqueue:");
    Scanf ("%d", &data);
    Enqueue (&QueueFront,
      &QueueRear, data);
    Printf ("Element %d Enqueued into the
        Queue.\n", data);
      break;
case 2:
  data = dequeue (&QueueFront, &QueueRear);
  If (data != -1){
  Printf ("Element %d dequeued from the Queue
  break;
case 3:
  PrintQueue (QueueFront);
  break;
  case 4;
  Printf ("Existing the Programming");
    break;
case 5;
  Printf ("Existing the Programming");
  break;

default:
  Printf ("invalid choice. Please Enter a
      valid option.\n");
  }
} while (choice != 4);
While (QueueFront != NULL){
  Struct Node* temp = QueueFront;
  QueueFront = QueueFront->next;
  free (temp);
```

Scanned with OKEN Scanner

}
return 0;
}

Output: Queue menu:
1) Enqueue
2) Dequeue
3) Print Queue
4) Exit

Enter your choice: 1
Enter the element to Enqueue: 10
In" data); Element 10 Enqueued into Queue.

Queue Menu:-
Enter your choice: 1
Enter the element to Enqueue: 20
Element 20 Enqueued into Queue

Enter your choice: 1
Enter the element to Enqueue: 30
Element 30 Enqueued into Queue.

Enter your choice: 2
Element 10 dequeued from the Queue.

Enter your choice: 3
Queue: 20 → 30 → NULL

Enter your choice: 4
Exit.

Scanned with OKEN Scanner

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

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

void insertNode(struct Node** head, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    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("NULL\n");
}

void sortList(struct Node* head) {
    if (head == NULL || head->next == NULL) {
        return;
    }

    int swapped;
    struct Node* temp;
    struct Node* end = NULL;

    do {
```

Scanned with OKEN Scanner

```
        swapped = 0;
        temp = head;

        while (temp->next != end) {
            if (temp->data > temp->next->data) {

                int tempData = temp->data;
                temp->data = temp->next->data;
                temp->next->data = tempData;

                swapped = 1;
            }
            temp = temp->next;
        }
        end = temp;

    } while (swapped);
}


void reverseList(struct Node** head) {
    struct Node* prev = NULL;
    struct Node* current = *head;
    struct Node* next = NULL;

    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }

    *head = prev;
}


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

Scanned with OKEN Scanner

```c
int main() {
    struct Node* list1 = NULL;
    struct Node* list2 = NULL;
    int choice, data;

    do {
        printf("\n1. Insert into List 1\n2. Insert into List 2\n3. Sort List 1\n4. Reverse List 1\n5. Concatenate Lists\n6. Print List 1\n7. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data for List 1: ");
                scanf("%d", &data);
                insertNode(&list1, data);
                break;

            case 2:
                printf("Enter data for List 2: ");
                scanf("%d", &data);
                insertNode(&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:
                concatenateLists(&list1, list2);
                printf("Lists concatenated.\n");
                break;

            case 6:
                printf("List 1: ");
                printList(list1);
                break;

            case 7:
                printf("Exiting program.\n");
                break;
```

Scanned with OKEN Scanner

```
            scanf("%d", &data);
            insertNode(&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:
            concatenateLists(&list1, list2);
            printf("Lists concatenated.\n");
            break;

        case 6:
            printf("List 1: ");
            printList(list1);
            break;

        case 7:
            printf("Exiting program.\n");
            break;

        default:
            printf("Invalid choice. Please try again.\n");
            break;
    }

} while (choice != 7);

return 0;
```

Scanned with OKEN Scanner

```
1. Insert into List 1
2. Insert into List 2
3. Sort List 1
4. Reverse List 1
5. Concatenate Lists
6. Print List 1
7. Exit
Enter your choice: 1
Enter data for List 1: 10

1. Insert into List 1
2. Insert into List 2
3. Sort List 1
4. Reverse List 1
5. Concatenate Lists
6. Print List 1
7. Exit
Enter your choice: 1
Enter data for List 1: 20

1. Insert into List 1
2. Insert into List 2
3. Sort List 1
4. Reverse List 1
5. Concatenate Lists
6. Print List 1
7. Exit
Enter your choice: 1
Enter data for List 1: 2

1. Insert into List 1
2. Insert into List 2
3. Sort List 1
4. Reverse List 1
5. Concatenate Lists
6. Print List 1
7. Exit
Enter your choice: 4
List 1 reversed.

1. Insert into List 1
2. Insert into List 2
3. Sort List 1
4. Reverse List 1
5. Concatenate Lists
6. Print List 1
7. Exit
Enter your choice: 3
List 1 sorted.

1. Insert into List 1
2. Insert into List 2
3. Sort List 1
4. Reverse List 1
5. Concatenate Lists
6. Print List 1
7. Exit
Enter your choice: 2
Enter data for List 2: 15
```

Scanned with OKEN Scanner

```
. Insert into List 1
. Insert into List 2
. Sort List 1
. Reverse List 1
. Concatenate Lists
. Print List 1
. Exit
nter your choice: 2
nter data for List 2: 15

. Insert into List 1
. Insert into List 2
. Sort List 1
. Reverse List 1
. Concatenate Lists
. Print List 1
. Exit
nter your choice: 2
nter data for List 2: 13

. Insert into List 1
. Insert into List 2
. Sort List 1
. Reverse List 1
. Concatenate Lists
. Print List 1
. Exit
nter your choice: 5
ists concatenated.

. Insert into List 1
. Insert into List 2
. Sort List 1
. Reverse List 1
. Concatenate Lists
. Print List 1
. Exit
nter your choice: 6
ist 1: 2 -> 10 -> 20 -> 15 -> 13 -> NULL

. Insert into List 1
. Insert into List 2
. Sort List 1
. Reverse List 1
. Concatenate Lists
. Print List 1
. Exit
nter your choice: 7
xiting program.

rocess returned 0 (0x0)   execution time : 72.326 s
ress any key to continue.
```

Scanned with OKEN Scanner

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

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

void push(struct Node** top_ref, int new_data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = *top_ref;
    *top_ref = new_node;
}

int pop(struct Node** top_ref) {
    if (*top_ref == NULL) {
        printf("Stack is empty. Cannot pop.\n");
        return -1;
    }

    struct Node* temp = *top_ref;
    int popped_data = temp->data;
    *top_ref = temp->next;
    free(temp);

    return popped_data;
}

void printStack(struct Node* top) {
    printf("Stack: ");
    while (top != NULL) {
        printf("%d -> ", top->data);
        top = top->next;
    }
    printf("NULL\n");
}

int main() {
    struct Node* stackTop = NULL;
    int choice, data;
```

Scanned with OKEN Scanner

```c
do {
    printf("\nStack Menu:\n");
    printf("1. Push\n");
    printf("2. Pop\n");
    printf("3. Print Stack\n");
    printf("4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter the element to push: ");
            scanf("%d", &data);
            push(&stackTop, data);
            printf("Element %d pushed onto the stack.\n", data);
            break;

        case 2:
            data = pop(&stackTop);
            if (data != -1) {
                printf("Element %d popped from the stack.\n", data);
            }
            break;

        case 3:
            printStack(stackTop);
            break;

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

        default:
            printf("Invalid choice. Please enter a valid option.\n");
    }
} while (choice != 4);


while (stackTop != NULL) {
    struct Node* temp = stackTop;
    stackTop = stackTop->next;
    free(temp);
}

return 0;
}
```

Scanned with OKEN Scanner

```
Stack Menu:
1. Push
2. Pop
3. Print Stack
4. Exit
Enter your choice: 1
Enter the element to push: 10
Element 10 pushed onto the stack.

Stack Menu:
1. Push
2. Pop
3. Print Stack
4. Exit
Enter your choice: 1
Enter the element to push: 20
Element 20 pushed onto the stack.

Stack Menu:
1. Push
2. Pop
3. Print Stack
4. Exit
Enter your choice: 1
Enter the element to push: 30
Element 30 pushed onto the stack.

Stack Menu:
1. Push
2. Pop
3. Print Stack
4. Exit
Enter your choice: 2
Element 30 popped from the stack.

Stack Menu:
1. Push
2. Pop
3. Print Stack
4. Exit
Enter your choice: 3
Stack: 20 -> 10 -> NULL

Stack Menu:
1. Push
2. Pop
3. Print Stack
4. Exit
Enter your choice: 4
Exiting the program.

Process returned 0 (0x0)   execution time : 18.034 s
Press any key to continue.
```

Scanned with OKEN Scanner

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


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


void enqueue(struct Node** front_ref, struct Node** rear_ref, int new_data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = NULL;

    if (*rear_ref == NULL) {

        *front_ref = *rear_ref = new_node;
    } else {

        (*rear_ref)->next = new_node;
        *rear_ref = new_node;
    }
}


int dequeue(struct Node** front_ref, struct Node** rear_ref) {
    if (*front_ref == NULL) {
        printf("Queue is empty. Cannot dequeue.\n");
        return -1;
    }

    struct Node* temp = *front_ref;
    int dequeued_data = temp->data;


    *front_ref = temp->next;


    if (*front_ref == NULL) {
        *rear_ref = NULL;
    }

    free(temp);

    return dequeued_data;
}
```

Scanned with OKEN Scanner

```
    }

    free(temp);

    return dequeued_data;


oid printQueue(struct Node* front) {
    printf("Queue: ");
    while (front != NULL) {
        printf("%d -> ", front->data);
        front = front->next;
    }
    printf("NULL\n");


nt main() {
    struct Node* queueFront = NULL;
    struct Node* queueRear = NULL;
    int choice, data;

    do {
        printf("\nQueue Menu:\n");
        printf("1. Enqueue\n");
        printf("2. Dequeue\n");
        printf("3. Print Queue\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the element to enqueue: ");
                scanf("%d", &data);
                enqueue(&queueFront, &queueRear, data);
                printf("Element %d enqueued into the queue.\n", data);
                break;

            case 2:
                data = dequeue(&queueFront, &queueRear);
                if (data != -1) {
                    printf("Element %d dequeued from the queue.\n", data);
                }
                break;

            case 3:
```

Scanned with OKEN Scanner

```c
do {
    printf("\nQueue Menu:\n");
    printf("1. Enqueue\n");
    printf("2. Dequeue\n");
    printf("3. Print Queue\n");
    printf("4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter the element to enqueue: ");
            scanf("%d", &data);
            enqueue(&queueFront, &queueRear, data);
            printf("Element %d enqueued into the queue.\n", data);
            break;

        case 2:
            data = dequeue(&queueFront, &queueRear);
            if (data != -1) {
                printf("Element %d dequeued from the queue.\n", data);
            }
            break;

        case 3:
            printQueue(queueFront);
            break;

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

        default:
            printf("invalid choice please enter valid option\n");
    }
} while (choice != 4);


while (queueFront != NULL) {
    struct Node* temp = queueFront;
    queueFront = queueFront->next;
    free(temp);
}

return 0;
```

Scanned with OKEN Scanner

```
Queue Menu:
1. Enqueue
2. Dequeue
3. Print Queue
4. Exit
Enter your choice: 1
Enter the element to enqueue: 10
Element 10 enqueued into the queue.

Queue Menu:
1. Enqueue
2. Dequeue
3. Print Queue
4. Exit
Enter your choice: 1
Enter the element to enqueue: 20
Element 20 enqueued into the queue.

Queue Menu:
1. Enqueue
2. Dequeue
3. Print Queue
4. Exit
Enter your choice: 1
Enter the element to enqueue: 30
Element 30 enqueued into the queue.

Queue Menu:
1. Enqueue
2. Dequeue
3. Print Queue
4. Exit
Enter your choice: 2
Element 10 dequeued from the queue.

Queue Menu:
1. Enqueue
2. Dequeue
3. Print Queue
4. Exit
Enter your choice: 3
Queue: 20 -> 30 -> NULL

Queue Menu:
1. Enqueue
2. Dequeue
3. Print Queue
4. Exit
Enter your choice: 4
Exiting the program.

Process returned 0 (0x0)   execution time : 22.450 s
Press any key to continue.
```

Scanned with OKEN Scanner