

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));
    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 Print List (Struct Node* node) {
    while (node != NULL) {
        printf ("%d ", node->data);
        node = node->next;
    }
    printf ("NULL\n");
}

void Sort List (Struct Node** head_ref) {
    if (*head_ref == NULL) {
        return;
    }

    int swapped = 1;
    Struct Node* p = *head_ref;
    Struct Node* lptr = NULL;
```

```

do {
    swapped = 0; // flag to check if swap happened
    ptr1 = *head_ref; // head of first list
    (ptr1->next != lptr) &
    (lptr->data > ptr1->next->data) &
    temp = ptr1->data;
    ptr1->data = ptr1->next->data;
    ptr1->next->data = temp;
    swapped = 1; // swap happened
}
while (swapped);

ptr1 = 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 connectLists (struct Node** head1, struct Node** head2) {
    if (*head1 == NULL) &
        *head1 = head2;
    else
        return;
    struct Node* temp = *head1;
    while (temp->next != NULL) {
        temp = temp->next;
        temp->next = head2;
    }
    temp->next = head2;
}

```

N
2011 by

```

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("In Original List 1: ");
    PrintList(list1);
    printf("Original List 2: ");
    PrintList(list2);

    SortList(&list1);
    SortList(&list2);

    printf("In sorted List 1: ");
    PrintList(list1);
    printf("sorted List 2: ");
    PrintList(list2);

    ConcatenateLists(&list1, list2);
    printf("In Concatenated List: ");
    PrintList(list1);

    printf("In Reversed List: ");
    PrintList(list2);
    return 0;
}

```



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 <iostream.h> // for cout, cin
#include <stdlib.h> // for malloc, free
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;
  *top_ref = (*top_ref)->next;
  free(temp);
  return temp->data;
}
  
```

```

Struct Node *tumb = *top_ref;
int PopPed_data = tumb->data;
*tumb = tumb->next;
free (tumb);
return PopPed_data;
}

void Print Stack (Struct Node * top) {
    printf ("Stack : ");
    while (top != NULL) {
        printf ("%d \rightarrow ", top->data);
        top = top->next;
    }
    printf ("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");
        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 (&stack_top, data);
                printf ("Element %d pushed onto the stack\n");
                break;
            case 2:
                data = POP (&stack_top);
                if (data != -1) {
                    printf ("Element %d popped from the
stack .\n", data);
                }
                break;
        }
    } while (choice != 4);
}

```

Case 3:

PrintStack (StackTop);
break;

Case 4:

Print f ("Exiting the Program.\n");
break;

Default:

Print f ("invalid choice. Please Enter a valid
option\n");

}
{ while (choice != 4);

while (StackTop1 == NULL) {

Struct Node * tempB = StackTop;

StackTop = StackTop → next;

free (tempB);

}

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

M	T	W	T	F	S	S
Page No.:	132	Date:	YOUVA			

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 ^{poped from} Pushed ^{into the stack}.

30 * about 30

Enter your choice - 3

Stack : 20 → 10 → NULL ^{size 2} ^{front}

→ pr-front * * about 30

Enter your choice 34: pr-rear * * about 30

Exit.

(about 30) = about 30 * about 30

((about 30) (0, 30)) return

: about 30 ← about 30 ←

: (30) = front ← about 30

↳ (30 = pr-rear *) fi

: about 30 = pr-rear * pr-front *

: about 30 ← front ← (pr-rear *)

: about 30 = pr-rear *

: (pr-front) * about 30) ^{size 2} ^{front} → about 30

↳ (pr-rear * about 30)

↳ (30 = pr-front *) fi

{(i/sumpt forward). print si sumpt") } fi

29 Jan / 2024

M	T	W	T	F	S	S
Page No.:						
Date:						YOUVA

Week - 6

Lab - 6

Program - 3

```
#include < stdio.h >
#include < stdlib.h >
linker
Struct Node {
    int data;
    Struct Node * next;
};
```

Element into the Queue or (- of : 100)

```
Void Enqueue (Struct Node** front-rgf,
Struct Node** rear-rgf, int new-data)
{
```

```
Struct Node* new-node = (Struct Node*)
malloc (Size of (Struct Node));
new-node -> data = new-data;
new-node -> next = NULL;
```

```
if (* rear-rgf == NULL) {
    * front-rgf = * rear-rgf = new-node;
}
else {
    (* rear-rgf) -> next = new-node;
    * rear-rgf = new-node;
}
```

```
From int dequeue (Struct Node** front-rgf,
Struct Node** rear-rgf) {
if (* front-rgf == NULL) {
    Print f ("Queue is Empty . Cannot dequeue \n");
    return -1;
}
```



↳ (Circular) deletion

```

Struct Node * delNode = *front->next;
int dequeued_data = front->data; // first
cout << "Dequeued : " << dequeued_data << endl;
*front->next = front->next->next; // remove
if (*front->next == NULL) break; // if
else if (*front->next->next == NULL) break;
}
cout << "Deleted : " << dequeued_data << endl;
free (front);

```

return dequeued_data;

↳ (Circular) insertion : at nth

```

Void PrintQueue (Struct Node *front) {
    printf ("Front : %d\n"); // front
    while (front != NULL) {
        printf ("%d -> ", front->data);
        front = front->next;
    }
    printf ("NULL\n");
}

```

int main (void) {
 int choice;

Struct Node * Queue Front = NULL;

Struct Node * Queue Rear = NULL; // last

int choice, data; // initialized

choice

They
will

do {

printf ("1. Insert Queue : \n"); // 1st

scanf ("%d", &choice); // 1st

printf ("2. Delete Queue : \n"); // 2nd

printf ("3. Print Queue : \n"); // 3rd

printf ("4. Exit : \n"); // 4th

printf ("Enter to your choice : "); // 5th

scanf ("%d", &choice); // 6th

if (choice == 1) {
 front = front->next; // front = front->next
}

else if (choice == 2) {
 front = front->next; // front = front->next
}

else if (choice == 3) {
 front = front->next; // front = front->next
}

else if (choice == 4) {
 front = front->next; // front = front->next
}



Switch (choice) &

Case 1: ~~for next = front * queueFront~~

Printf ("Enter the Element to Enqueue: ");

Scanf ("%d", &data);

Engineer (&queueFront, &front *

& Queue Rear, data); });

Printf ("Element %d Enqueued into the
Queue.\n", data);

break; ~~(front) ==~~

case 2: ~~for next = front * queueFront~~

data = dequeue (&queueFront, &queueRear);

? If (data == -1) ~~return false~~ break;

Printf ("Element %d dequeued from the Queue
break; ~~if (next == 1) front =~~);

~~if (front == -1) front = 0; else front = front + 1;~~

case 3: ~~for next = front = -1~~

Print Queue (queueFront);

break; ~~(front) == -1~~);

case 4:

Printf ("Exiting the Program");

~~break; return 0; but now~~

case 5: ~~for next = front = -1~~

Printf ("Exiting the Program");

break; ~~return 0; but now~~

default: ~~want sum(1) if true~~

Printf ("Invalid choice! Please Enter a
valid option. In");

~~if (sum == 1) front = 1; else front = 0;~~

? while (choice != 4);

while (queueFront != NULL);

Struct Node* temph = queueFront;

queueFront = queueFront -> next;

free (temp);

M	T	W	T	F	S	S
Page No.:		Date:				
					YOMYA	

}
queue 0;
3

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 deque 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.

```
#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 {
```



```

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;
    }
}

```



```
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;
        }
    } while (choice != 7);
}
```



```
    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;
```



```
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
```

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

. Insert into List 1
. Insert into List 2
. Sort List 1
. Reverse List 1
. Concatenate Lists
. Print List 1
. Exit
Enter your choice: 6
List 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
Enter your choice: 7
Exiting program.

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

```
#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;
```

```
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;
}
```

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

```
#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;
}
```

```

    *rear_rear = 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;

    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 enqueue 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:

```



```

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 enqueue 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;

```



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.