K. J. Somaiya College of Engineering, Mumbai-77
(Autonomous College Affiliated to University of Mumbai)

Batch: B1 **Roll No.: 1711072**

Experiment No. 1

Grade: AA / AB / BB / BC / CC / CD /DD

Title: Linked Lists

Objective: Implementation of different operations on linked list-concatenate, reverse, count number of nodes.

Expected Outcome of Experiment:

CO	Outcome
CO2	Use linear and non-linear data structure in domain like compiler construction, DBMS, etc.

Books/ Journals/ Websites referred:

Thomas Cormen, Charles Leiserson, Ronald Rivest, Clifford Stein; (CLRS) "Introduction to Algorithms", Third Edition, The MIT Press.

Abstract:-

DATA NODE:

1. data <integer>
2. next <node pointer>

OPERATIONS:

1. **Create List**

This function is used to initiate the process of creating a linked list by adding the first element of the linked list. It returns pointer to the start of the linked list.

2. **Insert a node in the beginning**

This function is used to insert a new node at the beginning of the linked list regardless of whether the linked list is of size 1 or size n. It takes the head pointer as parameter and returns the new head pointer.

3. **Insert node in the end**

This function is used to insert a new node at the end of the linked list regardless of whether the linked list is of size 1 or size n. It takes the head pointer as parameter and has no return type.

4. **Insert before a node**

This function is used to insert a new node before another node already present in the linked list. It takes head pointer as parameter, asks users for data before which data should be entered and has no return type.

5. **Insert after a node**

This function is used to insert a new node after another node already present in the linked list. It takes head pointer as parameter, asks users for data after which data should be entered and has no return type.

6. **Delete first node**

This function is used to delete the first node of the linked list. It takes head pointer as parameter, deletes the first node and then reassigns new head pointer and returns it.

7. Delete last node

This function is used to delete the last node of the linked list. It takes head pointer as parameter, deletes the last node and has no return type.

8. Delete before a node

This function is used to delete a node before another node of the linked list. It takes head pointer as parameter, asks user for the data before which deletion should be performed, and then deletes the necessary node. It has no return type.

9. Delete after a node

This function is used to delete a node after another node of the linked list. It takes head pointer as parameter, asks user for the data after which deletion should be performed, and then deletes the necessary node. It has no return type.

10. Search for a node

This function is used to search for a node in a linked list. We pass a head pointer to the function and then it asks the user to enter data to search for. If the element is present, we display the address of the element, else we display a message saying "No such data found".

11. Display the linked list

This function is used to display the data elements of the linked list on the output screen. It takes head pointer as parameter and has no return type.

12. Reverse the linked list

This function is used to actually reverse the links of nodes of the linked list. The function then calls for the display() function to show the reversed linked list. It takes the head pointer as parameter and returns a new head pointer.

13. Count nodes

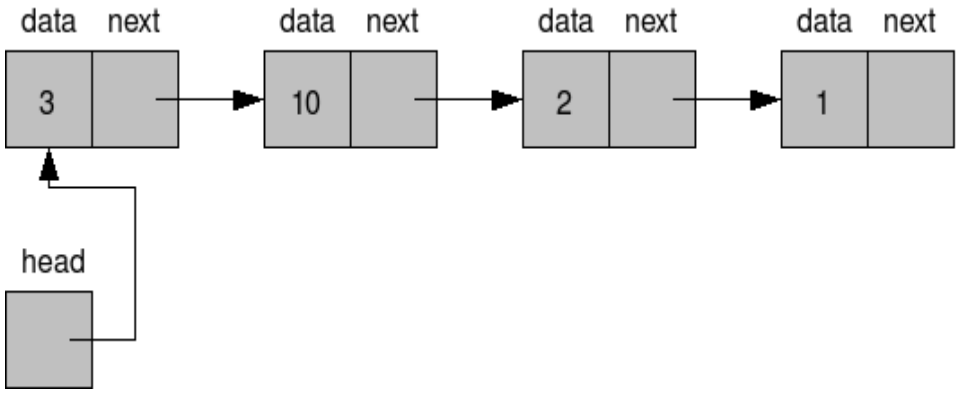
This function is used to count the number of nodes present in the linked list. It takes head pointer as parameter and has no return type.

14. Concatenate two linked lists

This function is used to connect two linked lists and display the new elongated linked list. It connects the end of first linked list to the start of second linked list. It takes the head pointers of two linked lists and has no return type.

RELATED THEORY:

- Linked list is a data structure that is free from the aforementioned restrictions. A linked list does not store its elements in consecutive memory locations and the user can add any number of elements to it.
- Unlike an array, a linked list does not allow random access of data. Elements in a linked list can be accessed only in a sequential manner. But like an array, insertions and deletions can be done at any point in the list in a constant time.
- A linked list, in simple terms, is a linear collection of data elements. These data elements are called nodes. A linked list can be perceived as a train or a sequence of nodes in which each node contains one or more data fields and a pointer to the next node.
- Linked list is a data structure which in turn can be used to implement other data structures. Thus, it acts as a building block to implement data structures such as stacks, queues, priority queues, circular queues, double ended queues, etc.
- An example of a linked list of students is represented diagrammatically below:



- The head pointer is a data-less (dummy) pointer of node type which is used to maintain address of first node.
- The left part of the node which contains data may include a simple data type, an array, or a structure.
- The right part of the node contains a pointer to the next node (or address of the next node in sequence). The last node will have no next node connected to it, so it will store a special value called NULL. A NULL pointer denotes the end of the list.

- Since in a linked list, every node contains a pointer to another node which is of the same type, it is also called a self-referential data type.
- In linked lists, nodes can be added as well as deleted one by one. While adding a node, memory occupied by one node is allocated at a time, not more, not less. While deleting a node from the linked list, the memory occupied by that node will be made free.
- Hence, a dynamic data structure (one which can grow or shrink) can be implemented without shortage or waste of memory (These are the two possible problems that may arise while using arrays). Moreover, insertion and deletion can be implemented efficiently by adjusting a couple of links without having the need to shift any element.
- The only disadvantage of linked lists is that it requires a link pointer with every node and hence requires more space. But, this space is negligible in real life applications where the data field may be a structure of 100s of bytes whereas the link pointer will consume just 2 to 4 bytes.

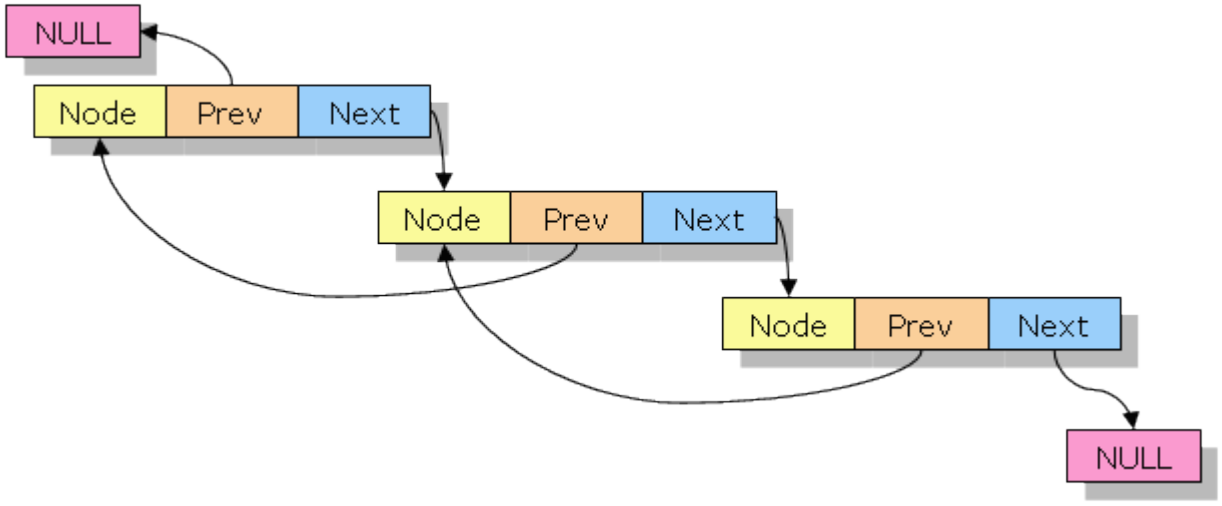
TYPES OF LINKED LISTS:

1. Singly Linked Lists

2. Doubly Linked Lists

3. Circular Singly Linked Lists

4. Circular Doubly Linked Lists



DOUBLY LINKED LIST

Implementation Details:

CODE:

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

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

node *create(node *head, node *start2){
  if(head==NULL){
    node *newnode;
    int data;
    newnode=(node*)malloc(sizeof(node));
    printf("\nEnter data for the node 1:");
    scanf("%d", &data);
    newnode->data=data;
    newnode->next=NULL;
    head=newnode;
    return head;
  }
  else {
    node *newnode;
    int data;
    newnode=(node*)malloc(sizeof(node));
    printf("\nEnter data for the node 2: ");
    scanf("%d", &data);
    newnode->data=data;
    newnode->next=NULL;
    start2=newnode;
    return start2;
  }
}
node *insert_beg(node *head){
  node *temp, *newnode;
  int data;
  temp=head;
  newnode=(node*)malloc(sizeof(node));
  printf("\nEnter data for node: ");
  scanf("%d", &data);
  newnode->data=data;
  newnode->next=temp;
  head=newnode;
```

```
  return head;
}
void insert_end(node *head, node *head2){
  node *temp, *newnode, *temp2;
  int data;
  temp=head;
  temp2=head2;
  while(temp->next!=NULL){
    temp=temp->next;
  }
  if(head2!=NULL){
  while(temp2->next!=NULL){
    temp2=temp2->next;
  }
  }
  if(head2==NULL){
  newnode=(node*)malloc(sizeof(node));
  printf("\nEnter data for node: ");
  scanf("%d", &data);
  newnode->data=data;
  newnode->next=NULL;
  temp->next=newnode;
  }
  else{
  newnode=(node*)malloc(sizeof(node));
  printf("\nEnter data for node 2: ");
  scanf("%d", &data);
  newnode->data=data;
  newnode->next=NULL;
  temp2->next=newnode;
  }
}

void insert_bef(node* head){
  int existing_data, newdata;
  node *prevnode=head, *newnode,*temp=head;
printf("\nEnter the data before which node should be inserted: ");
scanf("%d", &existing_data);
while(temp->data!=existing_data){
   prevnode=temp;
   temp=temp->next;
   if(temp->next==NULL){
      //printf("Data not found");
      break;
```

```
}
}
if(temp->data==existing_data){
printf("Enter data for new node: ");
scanf("%d", &newdata);
newnode=(node*)malloc(sizeof(node));
newnode->data=newdata;
newnode->next=temp;
prevnode->next=newnode;
}
else{
  printf("No such data found");
}
}
node *delete_beg(node *head){
  node *ptr=head;
  head=head->next;
  free(ptr);
  return head;
}
void delete_end(node *head){
  node *ptr=head, *prev=head;
  while(ptr->next!=NULL){
    prev=ptr;
    ptr=ptr->next;
  }
  prev->next=NULL;
  free(ptr);
}
void delete_bef(node *head){
   int existing_data;
  node *prevnode=head,*temp=head;
printf("\nEnter the data whose node should be deleted: ");
scanf("%d", &existing_data);
while(temp->data!=existing_data){
   prevnode=temp;
   temp=temp->next;
   if(temp->next==NULL){
      //printf("Data not found");
      break;
}
}
if(temp->data==existing_data){
prevnode->next=temp->next;
```

```
temp->next=NULL;
free(temp);
}
else
  printf("No data found.");
}
void delete_aft(node *head){
   int existing_data, newdata;
  node *nextnode=head, *temp=head;
printf("\nEnter the data after which node should be deleted: ");
scanf("%d", &existing_data);
while(temp->data!=existing_data){
   nextnode=temp;
   temp=temp->next;
   if(temp->next==NULL){
      //printf("Data not found");
      break;
}
}
if(temp->data==existing_data){
nextnode=temp;
temp=temp->next;
nextnode->next=temp->next;
temp->next=NULL;
free(temp);
}
}
void insert_aft(node *head){
   int existing_data, newdata;
  node *nextnode=head, *newnode,*temp=head;
printf("\nEnter the data after which node should be inserted: ");
scanf("%d", &existing_data);
while(temp->data!=existing_data){
   nextnode=temp;
   temp=temp->next;
   if(temp->next==NULL){
      //printf("Data not found");
      break;
}
}
if(temp->data==existing_data){
printf("Enter data for new node: ");
scanf("%d", &newdata);
nextnode=temp;
```

```
temp=temp->next;
newnode=(node*)malloc(sizeof(node));
newnode->data=newdata;
newnode->next=temp;
nextnode->next=newnode;
}
else{
  printf("No such data found");
}
}
void display(node *head){
  node *ptr=head;
  if(ptr==NULL)
  printf("\nLinked list is empty.");
  else{
    printf("Data is: ");
    while(ptr!=NULL){
    printf(" %d ",ptr->data);
    ptr=ptr->next;
  }
  }
}
void search(node *head){
  int d;
  printf("\nEnter the data to be searched: ");
  scanf("%d",&d);
  node *ptr;
  ptr=head;
  while(ptr->data!=d){
    ptr=ptr->next;
  }
  if(ptr->next==NULL)
    printf("\n Data not found.");
  else
    printf("\n Data is found at %p address.",ptr->next);
}
node *reverse_list(node *head){
  node *prev=NULL, *curr=head, *next=NULL;
  while(curr!=NULL){
    next=curr->next;
    curr->next=prev;
    prev=curr;
    curr=next;
  }
```

```
  head=prev;
  display(head);
  return head;
}
void count(node *head){
  node *temp=head;
  int count=0;
  while(temp!=NULL){
    count++;
    temp=temp->next;
  }
  printf("Number of nodes: %d", count);
}
void concatenate(node *start, node *start2){
  node *s1=start, *s2=start2;
  while(s1->next!=NULL){
    s1=s1->next;
  }
  s1->next=s2;
  display(start);
}
int main(){
int choice = 5;
node *start=NULL;
node *start2=NULL;
while(choice!=15){
  printf("\n1. Create first node\n2. Insert node at start\n3. Insert node
at end\n4. Insert before a node\n5. Insert after a node\n6. Delete first
node\n7. Delete last node\n8. Delete particular node\n9. Delete after a
node\n10. Search for a node\n11. Display Linked List\n12. Reverse the
linked list\n13. Count\n14. Concatenate\n15. Exit\n");
  printf("Enter a choice: ");
  scanf("%d", &choice);
  if(start==NULL && start2==NULL && choice!=1)
  {
    printf("Create a linked list first.");
    continue;
  }
  switch(choice){
    case 1:
    if(start==NULL)
    start=create(start, start2);
    else
    start2=create(start,start2);
```

```
break;
case 2:
start=insert_beg(start);
break;
case 3:
insert_end(start, start2);
break;
case 4:
insert_bef(start);
break;
case 5:
insert_aft(start);
break;
case 6:
start=delete_beg(start);
break;
case 7:
delete_end(start);
break;
case 8:
delete_bef(start);
break;
case 9:
delete_aft(start);
break;
case 10:
search(start);
break;
case 11:
display(start);
break;
case 12:
start=reverse_list(start);
break;
case 13:
count(start);
break;
case 14:
concatenate(start,start2);
break;
case 15:
break;
default:
printf("Invalid Choice. Please try again.\n");
```

```
    break;
  }
}
}
return 0;
}
```

OUTPUT SCREENS:

```
Enter a choice:  1

Enter data for the node 1: 20

1. Create first node
2. Insert node at start
3. Insert node at end
4. Insert before a node
5. Insert after a node
6. Delete first node
7. Delete last node
8. Delete before a node
9. Delete after a node
10. Search for a node
11. Display Linked List
12. Reverse the linked list
13. Count
14. Concatenate
15. Exit
Enter a choice:  11
Data is:  20
```

```
Enter a choice:  2

Enter data for node:  10

1. Create first node
2. Insert node at start
3. Insert node at end
4. Insert before a node
5. Insert after a node
6. Delete first node
7. Delete last node
8. Delete before a node
9. Delete after a node
10. Search for a node
11. Display Linked List
12. Reverse the linked list
13. Count
14. Concatenate
15. Exit
Enter a choice:  11
Data is:  10  20
```

```
Enter a choice:  3

Enter data for node:  30

1. Create first node
2. Insert node at start
3. Insert node at end
4. Insert before a node
5. Insert after a node
6. Delete first node
7. Delete last node
8. Delete before a node
9. Delete after a node
10. Search for a node
11. Display Linked List
12. Reverse the linked list
13. Count
14. Concatenate
15. Exit
Enter a choice:  11
Data is:  10  20  30
```

```
Enter a choice:  6

1. Create first node
2. Insert node at start
3. Insert node at end
4. Insert before a node
5. Insert after a node
6. Delete first node
7. Delete last node
8. Delete before a node
9. Delete after a node
10. Search for a node
11. Display Linked List
12. Reverse the linked list
13. Count
14. Concatenate
15. Exit
Enter a choice:  11
Data is:  15  20  25  30
```

```
Enter a choice:  4

Enter the data before which node should be inserted:  30
Enter data for new node:  25

1. Create first node
2. Insert node at start
3. Insert node at end
4. Insert before a node
5. Insert after a node
6. Delete first node
7. Delete last node
8. Delete before a node
9. Delete after a node
10. Search for a node
11. Display Linked List
12. Reverse the linked list
13. Count
14. Concatenate
15. Exit
Enter a choice:  11
Data is:  10  20  25  30
```

```
Enter the data after which node should be inserted:  10
Enter data for new node:  15

1. Create first node
2. Insert node at start
3. Insert node at end
4. Insert before a node
5. Insert after a node
6. Delete first node
7. Delete last node
8. Delete before a node
9. Delete after a node
10. Search for a node
11. Display Linked List
12. Reverse the linked list
13. Count
14. Concatenate
15. Exit
Enter a choice:  11
Data is:  10  15  20  25  30
```

```
Enter a choice:  7

1. Create first node
2. Insert node at start
3. Insert node at end
4. Insert before a node
5. Insert after a node
6. Delete first node
7. Delete last node
8. Delete before a node
9. Delete after a node
10. Search for a node
11. Display Linked List
12. Reverse the linked list
13. Count
14. Concatenate
15. Exit
Enter a choice:  11
Data is:  15  20  25
```

```
Data is:  20  50  60  80
1. Create first node
2. Insert node at start
3. Insert node at end
4. Insert before a node
5. Insert after a node
6. Delete first node
7. Delete last node
8. Delete particular node
9. Delete after a node
10. Search for a node
11. Display Linked List
12. Reverse the linked list
13. Count
14. Concatenate
15. Exit
Enter a choice:  10

Enter the data to be searched:  50

 Data is found at 0x2054870 address.
```

```
Enter a choice:  8

Enter the data whose node should be deleted:  25

1. Create first node
2. Insert node at start
3. Insert node at end
4. Insert before a node
5. Insert after a node
6. Delete first node
7. Delete last node
8. Delete particular node
9. Delete after a node
10. Search for a node
11. Display Linked List
12. Reverse the linked list
13. Count
14. Concatenate
15. Exit
Enter a choice:  11
Data is:  10  15  20  30
```

```
Enter a choice:  9

Enter the data after which node should be deleted:  15

1. Create first node
2. Insert node at start
3. Insert node at end
4. Insert before a node
5. Insert after a node
6. Delete first node
7. Delete last node
8. Delete particular node
9. Delete after a node
10. Search for a node
11. Display Linked List
12. Reverse the linked list
13. Count
14. Concatenate
15. Exit
Enter a choice:  11
Data is:  10  15  30
```

```
Data is:  10  20  40  50
1. Create first node
2. Insert node at start
3. Insert node at end
4. Insert before a node
5. Insert after a node
6. Delete first node
7. Delete last node
8. Delete particular node
9. Delete after a node
10. Search for a node
11. Display Linked List
12. Reverse the linked list
13. Count
14. Concatenate
15. Exit
Enter a choice:  12
Data is:  50  40  20  10
```

```
Enter a choice:  1

Enter data for the node 2:  80
```

```
Enter a choice:  3

Enter data for node 2:  100
```

```
Enter a choice:  3

Enter data for node 2:  120
```

```
Enter a choice:  14
Data is:  20  40  60  80  100  120
```

```
Enter a choice:  14
Data is:  20  40  60  80  100  120
1. Create first node
2. Insert node at start
3. Insert node at end
4. Insert before a node
5. Insert after a node
6. Delete first node
7. Delete last node
8. Delete particular node
9. Delete after a node
10. Search for a node
11. Display Linked List
12. Reverse the linked list
13. Count
14. Concatenate
15. Exit
Enter a choice:  13
Number of nodes: 6
```

CONCLUSION:

Numerous operations on a linked list have been performed. We were able to insert and delete nodes at any possible locations, search the list for any data, traverse the list and find the count of nodes in the list. We were also able to concatenate two lists, as well as reverse a linked list.