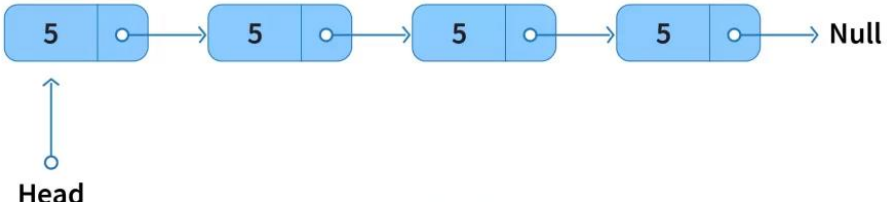
Name – Prerna Sunil Jadhav **SAP ID - 60004220127**

Experiment No – 02

AIM: Perform Insertion Deletion, Sorting, Searching And Traversal in Linear Linked List.

Theory:

A Linked List is a sequence of elements such that each element in the linked list points to the adjacent element in the sequence.



Each element of the Linked List is called a Node. A Linked List is formed by connecting multiple nodes. A node consists of two parts,

- Data
- Pointer to another node

Algorithm:

```
Insertion at the beginning:
1.  Allocate memory for new node
2.  Store data
3.  Change next of new node to point to head
4.  Change head to point to recently created node
struct node *newNode;
newNode = malloc(sizeof(struct node));
newNode->data = 4;
newNode->next = head;
head = newNode;

Insertion at the End:
1.  Allocate memory for new node
2.  Store data
3.  Traverse to last node
4.  Change next of last node to recently created node
struct node *newNode;
newNode = malloc(sizeof(struct node));
newNode->data = 4;
newNode->next = NULL;
struct node *temp = head;
while(temp->next != NULL){
```

```
  temp = temp->next;
}
temp->next = newNode;

Insertion at the Middle:
1.  Allocate memory and store data for new node
2.  Traverse to node just before the required position of new node
3.  Change next pointers to include new node in between
struct node *newNode;
newNode = malloc(sizeof(struct node));
newNode->data = 4;
struct node *temp = head;
for(int i=2; i < position; i++) {
  if(temp->next != NULL) {
    temp = temp->next;
  }
}
newNode->next = temp->next;
temp->next = newNode;

Delete from beginning:
1.  IF HEAD = NULL
      Write UNDERFLOW
      Go to Step 5
2.  [END OF IF]
3.  SET PTR = HEAD
4.  SET HEAD = HEAD -> NEXT
5.  FREE PTR
6.  EXIT

 Delete from end:
1.  IF HEAD = NULL
      Write UNDERFLOW
      Go to Step 8
      [END OF IF]
2.  SET PTR = HEAD
3.  Repeat Steps 4 and 5 while PTR -> NEXT!= NULL
4.  SET PREPTR = PTR
5.  SET PTR = PTR -> NEXT
6.  [END OF LOOP]
7.  SET PREPTR -> NEXT = NULL
8.  FREE PTR
9.  EXIT

Delete from middle
1.  IF HEAD = NULL
```

```
      WRITE UNDERFLOW
      GOTO STEP 10
      END OF IF
2.  SET TEMP = HEAD
3.  SET I = 0
4.  REPEAT STEP 5 TO 8 UNTIL I
5.  TEMP1 = TEMP
6.  TEMP = TEMP → NEXT
7.  IF TEMP = NULL
      WRITE "DESIRED NODE NOT PRESENT"
      GOTO STEP 12
      END OF IF
8.  I = I+1
9.  END OF LOOP
10. TEMP1 → NEXT = TEMP → NEXT
11. FREE TEMP
12. EXIT
```

Traversal:

```
1.  SET PTR = HEAD
2.  IF PTR = NULL
WRITE "EMPTY LIST"
GOTO STEP 7
END OF IF
3.  REPEAT STEP 5 AND 6 UNTIL PTR != NULL
4.  PRINT PTR→ DATA
5.  PTR = PTR → NEXT
6.  [END OF LOOP]
7.  EXIT
```

Searching:

```
1.  SET PTR = HEAD
2.  Set I = 0
3.  IF PTR = NULL
      WRITE "EMPTY LIST"
      GOTO STEP 8
      END OF IF
4.  REPEAT STEP 5 TO 7 UNTIL PTR != NULL
5.  if ptr → data = item
      write i+1
      End of IF
6.  I = I + 1
7.  PTR = PTR → NEXT
8.  [END OF LOOP]
9.  EXIT
```

Sorting:

1. Make the head as the current node and create another node index for later use.
2. If head is null, return.
3. Else, run a loop till the last node (i.e. NULL).
4. In each iteration, follow the following step 5-6.
5. Store the next node of current in index.
6. Check if the data of the current node is greater than the next node. If it is greater, swap current and index.

Program:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
typedef struct link
{
    int data;
    struct link *next;
} node;

void insert_begin(node **temp, int num);
void insert_mid(node **temp, int num, int index);
void insert_end(node **temp, int num);

void delete_beg(node **temp);
void delete_mid(node **temp, int index);
void delete_end(node **temp);

void sort(node *start);
void swap(node *a, node *b);
void search_link(node **temp,int x);

void dis(node *ptr);

void main()
{
    int num, ch,x ,n, i, index;
    char chh;
    node *start, *ptr, *temp;
    printf("Prerna Jadhav - 60004220127\n");
    printf("Enter the number of nodes you want to insert: \n");
    scanf("%d", &n);
    printf("Values for nodes:");
    for (i = 0; i < n; i++)
    {
        if (i == 0)
```

```
        {
            scanf("%d", &num);
            start = (node *)malloc(sizeof(node));
            start->data = num;
            start->next = NULL;
            temp = start;
        }
        else
        {
            scanf("%d", &num);
            ptr = (node *)malloc(sizeof(node));
            ptr->data = num;
            ptr->next = NULL;
            temp->next = ptr;
            temp = ptr;
        }
    }
    dis(start);
A:
    printf("\nEnter The operation you want to perform \n1) Insertion \n2) Deletion \n3) Search \n4) Sort \n5) Exit \n");
    scanf("%d", &ch);
    switch (ch)
    {
    case 1:
    {
        printf("\n\nWhere to perform operation \na) Begining \nb) Middle \nc) End \n");
        scanf("%s", &chh);
        switch (chh)
        {
        case 'a':
        {
            printf("\nEnter the element to be inserted at begining:\n");
            scanf("%d", &num);
            insert_begin(&start, num);
            dis(start);
            goto A;
        }

        case 'b':
        {
            printf("\nEnter the index where to put the node: ");
            scanf("%d", &index);
            printf("Enter the element to be inserted at that index:\n");
            scanf("%d", &num);
            insert_mid(&start, num, index);
```

```
            dis(start);
            goto A;
        }

        case 'c':
        {
            printf("\nEnter the element to be inserted at end:\n");
            scanf("%d", &num);
            insert_end(&start, num);
            dis(start);
            goto A;
        }
        }
    }
    case 2:
    {
        printf("\n\nWhere to perform operation \na) Begining \nb) Middle \nc) End \n");
        scanf("%s", &chh);
        switch (chh)
        {
        case 'a':
        {
            delete_beg(&start);
            printf("\nLink list after deletion at begining: \n");
            dis(start);
            goto A;
        }

        case 'b':
        {
            printf("\nEnter the number of node to be deleted:\n");
            scanf("%d", &index);
            delete_mid(&start, index);
            dis(start);
            goto A;
        }

        case 'c':
        {
            delete_end(&start);
            printf("\nLink list after deletion at end: \n");
            dis(start);
            goto A;
        }
        }
    }
```

```
    case 3:
    {
        printf("Enter the number to be searched; ");
        scanf("%d", &x);
        search_link(&start, x);
        goto A;
    }
    case 4:{
    sort(start);
    printf("\n Linked list after sorting ");
    dis(start);
    }

    }
    getch();
}

void insert_begin(node **temp, int num)
{
    node *ptr;
    ptr = (node *)malloc(sizeof(node));
    ptr->data = num;
    if (*temp == NULL)
        ptr->next = NULL;
    else
        ptr->next = *temp;
    *temp = ptr;
}
void insert_mid(node **temp, int num, int index)
{
    node *ptr, *loc;
    int i;
    ptr = (node *)malloc(sizeof(node));
    ptr->data = num;
    if (*temp == NULL)
    {
        ptr->next = NULL;
        *temp = ptr;
        return;
    }
    loc = *temp;
    while (i < index - 1)
    {
        *temp = (*temp)->next;
        if (*temp == NULL)
        {
```

```
            printf("array size exceeded");
            return;
        }
        i++;
    }
    ptr->next = loc->next;
    loc->next = ptr;
}
void insert_end(node **temp, int num)
{
    node *ptr, *loc;
    ptr = (node *)malloc(sizeof(node));
    ptr->data = num;
    ptr->next = NULL;

    loc = *temp;
    while (loc->next != NULL)
    {
        loc = loc->next;
    }
    loc->next = ptr;
}

void delete_mid(node **temp, int index)
{
    int i = 0;
    node *loc, *prev;
    loc = *temp;
    if (*temp == NULL)
    {
        printf("THERE ARE NO NODES-");
        return;
    }
    while (i < index - 1)
    {
        prev = loc;
        loc = loc->next;
        i++;
    }
    prev->next = loc->next;
    free(loc);
}
void delete_end(node **temp)
{
    node *loc, *prev;
    loc = *temp;
```

```
    if (*temp == NULL)
    {
        printf("THERE ARE NO NODES-");
        return;
    }
    while (loc->next != NULL)
    {
        prev = loc;
        loc = loc->next;
    }
    prev->next = loc->next;
    free(loc);
}
void delete_beg(node **temp)
{
    node *loc;
    if (*temp == NULL)
    {
        printf("THERE ARE NO NODES-");
        return;
    }
    loc = *temp;
    *temp = loc->next;
    free(loc);
}

void search_link(node **temp, int x)
{
    node *loc;
    loc = *temp;
    while (loc != NULL)
    {
        if (loc->data == x)
        {
            printf("Element is present.");
            return;
        }
        loc = loc->next;
    }
    printf("Element is not present.");
}

void dis(node *ptr)
{
    while (ptr != NULL)
    {
```

```
        printf("%d\t", ptr->data);
        ptr = ptr->next;
    }
}

void sort(node *start)
{
    int swapped, i;
    node *ptr1;
    node *lptr = NULL;
    if (start == NULL)
        return;
    do
    {
        swapped = 0;
        ptr1 = start;

        while (ptr1->next != lptr)
        {
            if (ptr1->data > ptr1->next->data)
            {
                swap(ptr1, ptr1->next);
                swapped = 1;
            }
            ptr1 = ptr1->next;
        }
        lptr = ptr1;
    }
    while (swapped);
}

void swap(node *a,node *b)
{
    int temp = a->data;
    a->data = b->data;
    b->data = temp;
}
```

OUTPUT:

```
Prerna Jadhav - 60004220127
Enter the number of nodes you want to insert:
4
Values for nodes:12 67 909 45
12      67      909     45
```

```
Enter The operation you want to perform
1) Insertion
2) Deletion
3) Search
4) Sort
5) Exit
4

 Linked list after sorting 12   45      67      909
```

Conclusion:

Linked List can be:

- Used to store single or bivariable polynomials.
- Act as a base for certain data structures like Queue, Stack, Graph.
- Strategy for file allocation schemes by Operating System.
- Keep track of free space in the secondary disk. All the free spaces can be linked together.
- Turn-based games can use a circular linked list to decide which player is about to be played. Once the player finishes its turn we move to the next player.
- To keep records of items such as music, videos, images, web pages, etc which link to one another and allows to traverse between them sequentially.