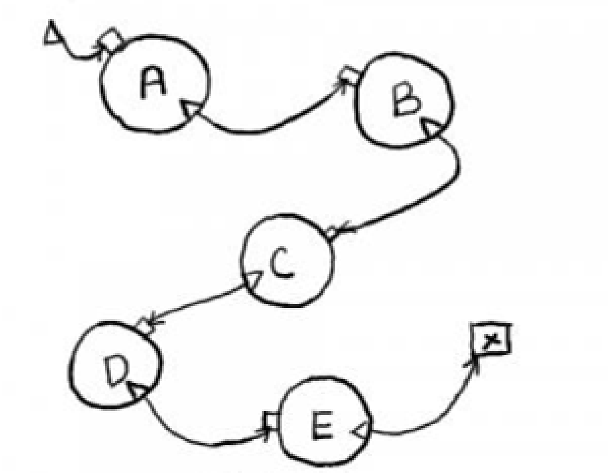
UNIT-4-LINKLIST



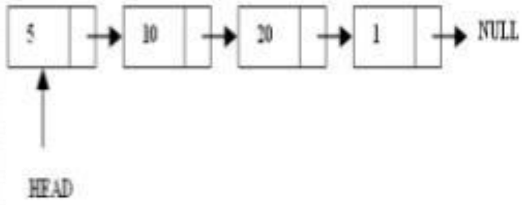
Linked List

- Introduction
- Singly linked lists.
    - Implementation of linked list
    - Insertion of a node at the beginning
    - Insertion of a node at the end
    - Insertion of a node after a specified node
    - Traversing the entire linked list
    - Deletion of a node from linked list
- Concatenation of linked lists
- Merging of linked lists
- Reversing of linked list
- Doubly linked list.
    - Implementation of doubly linked list
- Circular linked list

Applications of the linked lists

WhatisLinkedlist?

- A linked list is a linear data structure.
- Nodes make up linked lists.
- Nodes are structures made up of data and a pointer to another node.
- Usually the pointer is called next.



Linked List

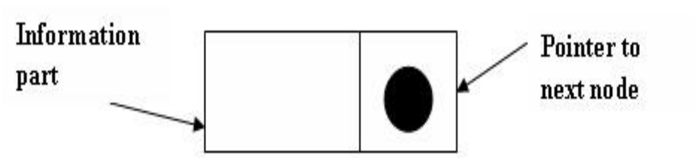
One disadvantage of using arrays to store data is that arrays are static structures and therefore cannot be easily extended or reduced to fit the data set. Arrays are also expensive to maintain new insertions and deletions.

The data structure called Linked Lists that addresses some of the limitations of arrays. A linked list is a **dynamic data structure**. The number of nodes in a list is not fixed and can grow and shrink on demand.

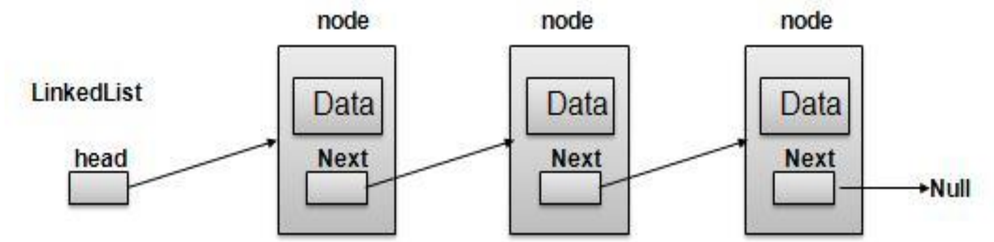
A linked list is a linear data structure where each element is a separate object. In computer science Linked list is one of the fundamental data structure and can be used to implement another data structure like tree and graph.
Linked list is defined as a collection of nodes. Each node has two parts:

1. Information

2. Pointer to next node



1. Information part consist of one or more than one fields.
2. Pointer to next node contains the address of the location where next information is stored. The last node of the list contains NULL in the pointer field.



Each node consists of its own data and the address of the next node and forms a chain. Linked is called **self referential data type** because it contains pointer or link to another data of same type.

Advantages of Linked Lists

- They are a dynamic in nature which allocates the memory when required.
- Insertion and deletion operations can be easily implemented.
- Stacks and queues can be easily executed.
- Linked List reduces the access time.

Disadvantages of Linked Lists

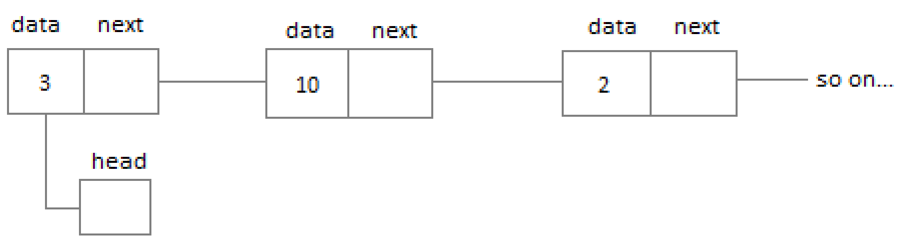
- The memory is wasted as pointers require extra memory for storage.
- No element can be accessed randomly; it has to access each node sequentially.
- Reverse Traversing is difficult in linked list.

Applications of Linked Lists

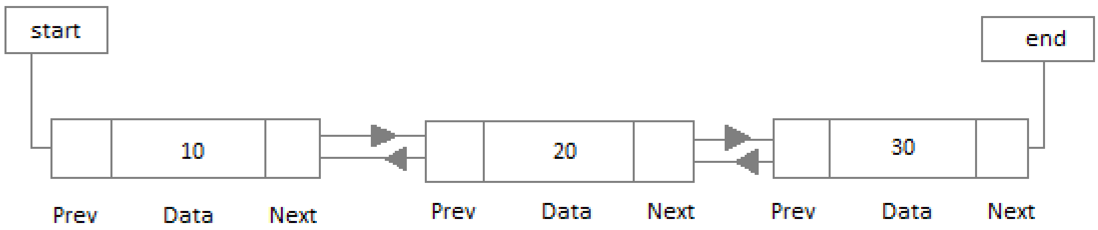
- Linked lists are used to implement stacks, queues, graphs,tree etc.
- Linked lists let you insert elements at the beginning and end of the list.
- In Linked Lists we don't need to know the size in advance.

Types of Linked Lists

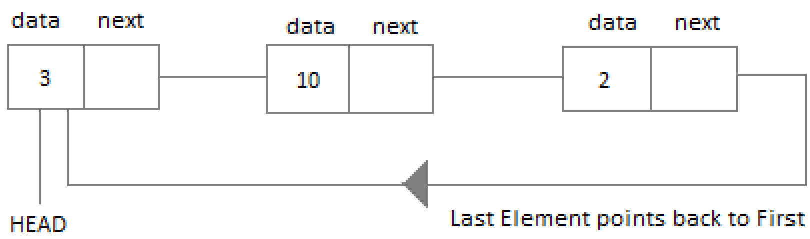
- **Singly Linked List :** Singly linked lists contain nodes which have a data part as well as an address part i.e. next, which points to the next node in sequence of nodes. The operations we can perform on singly linked lists are insertion, deletion and traversal.



- **Doubly Linked List :** In a doubly linked list, each node contains two links the first link points to the previous node and the next link points to the next node in the sequence.

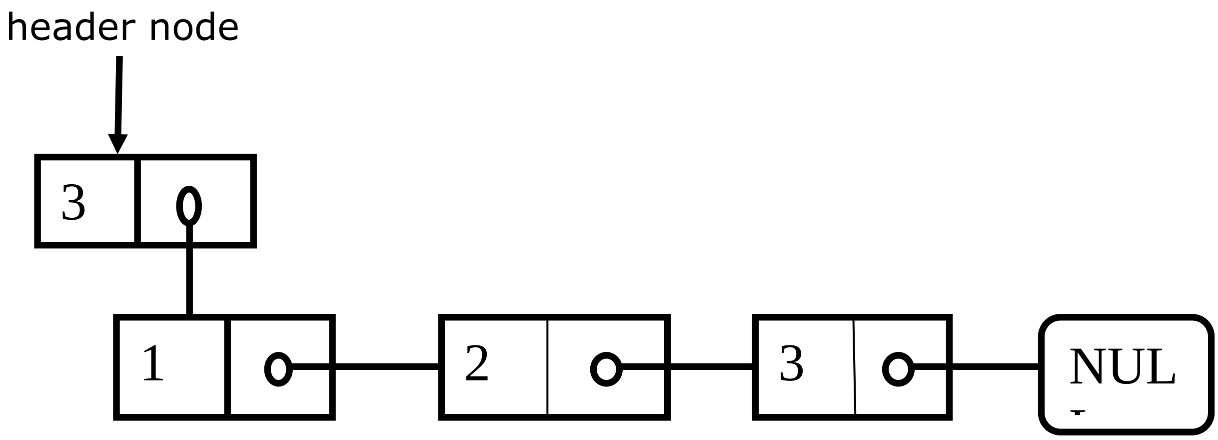


- **Circular Linked List :** In the circular linked list the last node of the list contains the address of the first node and forms a circular chain.



HEADER LINK LIST

- In header linked list a list contains special node which found always at the front of the list and it called header node.
- A header node contains special type of inforemation such as total numbers of node in a linked list.
- this is additional information :-



- types of header link list

  A. grounded header link list
  B. circular header link list

I. grounded header link list:-In this header link list start pointer always points to header node. If its empty than pointer contains NULL and last node is NULL.

II. circular header link list:-In this link list last node always points to header node.

Basic Operations

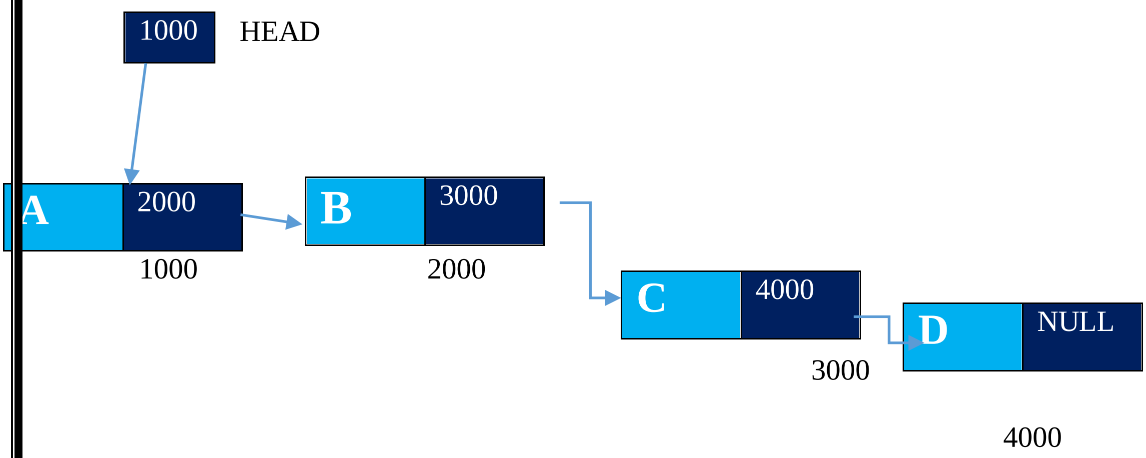
Following are the basic operations supported by a list.

- **Traversing** – access each element of the link list.
- **Insertion** – Add a new element to the link list.
- **Deletion** –removes the existing elements
- **Search** – Searches an element using the given key.

Traversing :-

Link list requires access to its nodes through sequential traversal.

The process of visiting each node of the list once is called traversing.



Insertion :-

Add a new element in Link List. Insertion at the front of LL.

1. At the beginning.
2. At the end.
3. In the middle.

Delete :-

Deletes an element using the given key.

1. At the beginning.
2. At the end.

3. In the middle.

➢ Singly Linked List

Create Linked List and traverse node in Linked List

Algorithm : create singly linked list

Step-1 : [Initially list empty]
next [Start] = NULL
node = start

Step-2 : [allocate space to newly created node]
next [node] = created node

Step-3 : [assign value to information part of a node]
no [node] = value

Step-4 : [assign value to the address part]
next [node] = NULL

Step-5 : exit

Algorithm : Algorithm to display

Step-1 : [Initialize]
Node=next [Start]
Step-2 : [display information]
repeate while node # NULL
print "information",no[node]
node = next [node]

Step-3 : exit

- **Insert first node in singly linked list**

Algorithm : Insert first node

Step-1 : [Initialazation]
node = next [start]

[pointer to first node of the list]
pre = start [Assign address of start to previous]

Step-2 : [Allocate space to newly create node]
new 1 = allocation

Step-3 : [Read value of information part of new node]
no [new1] = value

Step-4 : [Link currently created node]
next [pre] = new1
next [new1] = node

Step-5: exit

- **insert last node in singly linked list**

Algotithm : Insert last node

Step-1 : [Initialazation]
node = next [start]
pre = start

Step-2 : [Move the pointer to the end of the list]
repeat while node # NULL
node = next [node]
pre = next [pre]

Step-3 : [Allocate space for newly created node]
new 1 = create node (Allocation)

Step-4 : [Read value of information part of a new node]
no [new 1] = value

Step-5 : [Link currently created node with the last node]
next [pre] = new 1
next [new 1] = node

Step-5 : exit

- **<u>Insert a node in singly link list at a desire position</u>**

Algorithm : Insert node at desire position

```
Step-1 : [Initialization]
              node = next [start]
              pre = start
              node_no = 1
Step-2 : [Read node position where we want to insert]
              input insert_no
Step-3 : [Perform insertion operation]
              repeat through  Step-4 while node # NULL

Step-4 : [check if insert_node is equal to node_no]
              If  node_no = insert_no
                     new 1 = Allocation
                     no [new 1] = value
                     next[new 1] = node
         next[pre]=new1
       else
                     [move the pointer foeword]
                            node = next [node]
                            pre = next [pre]
       end if
                     node_no = node_no + 1

Step-5 : exit
```

➢ Deletion in singly linked list

- ***Delete first node in a singly linked list***

Algorithm : Delete first node

Step-1: [Initialization]
node = next [start] [points to first node of the list]
previous = Address of start [Assign address of start to previous]
Step-2: [Check for the list is empty or not]
if node = NULL
output "Underflow"
exit
Step-3: [Perform deletion operation]
else [Delete first node]
next [previous] = next [node]
[move the position to next node in the list and free space associated with node which you delete]

Step-4: exit

- **Deletion last node in singly linked list**

Algorithm: [Initialization]
node =next [start] [point to first node of the list]
previous = address of start [Assign address of start to previous]
node_no = 0

Step-2: [Check for the list is empty or not]
if node = NULL
output "Underflow"
exit

Step-4: [scan the list to count the no of node in the list]
repeat while node # NULL
node = next [node]
pre = next [pre]
node_no = node_no+1

Step-4: [Initialization once again]
node = next [start]
pre = address of start

Step-5: [scan the list for 1 less to size of the list]
repeat while node_no # 1
node = next [pre]

pre = next [pre]
node_no = node_no+1

Step-6: [check if the last node is arising]
if node_no = 1
next [pre] = next [node]
free (node)

Step-7: exit

- **Delete node based node number(desire position)**

Algorithm: Delete node based node number

Step-1: [Initialization]
node = start next [points to first node of the list]

pre = address of start [Assign address of start to previous]

Step-2: [Initialization node counter]
node_no = 1

Step-3: [Read node number]
delete_node = no

Step-4: [Check for the list is empty or not]
if node = NULL
output " Underflow"
exit

Step-5: [perform deletion operation]
repeat through stap-6 while node # NULL
if node_no = delete_node
next [pre] = next [node]
(Mack link of previous node to the next node)
delete (node)
(delete current node)
exit

Step-6: node_no = node_no + 1

Step-7: exit

➢ Doubly Linked List

Algorithm to create and display in doubly linked list

Algorithm: Create linked list

Step-1: [Initialization]
pre [start] =NULL
next[start] =NULL
node=start

stet-2: [allocate space far newly create node]
next [node]=create node

[Assign next and pre link]
pre[next [node]]=node
node =next [node]

Step-3: [enter data]
no[node]=value

Step-4: [assign last node as a NULL]
next [node]=NULL

Step-5: exit

Algorithm: Display Linked List

Step-1: [Initialization]
next[start]=node

Step-2: [display information]
repeat while node # NULL
print information
node = next[node]

Step-3: exit

Algorithm to insert first node

Step-1: [Initialization]
    node = next[start]

Step-2: [create a new node]
    new1= allocate space

Step-3: [input value about newly created node]
        info [new1] = value

Step-4: [Make link of create node]
        next [pre[node]] = new1
        pre [new1] = pre [node]
        next [new1] = node
        pre [node] = new1
Step-5: exit

Algorithm to insert Last node

Step-1: [Initialization]
    node = start

Step-2: [move the pointer to the end of the list]
    Repeat while node # NULL
        node=next[node]

Step-3: [allocation space for newly create node]
    new1 = allocate space

Step-4: [read info associated with newly create node]
    info [new1] = value

Step-5: [assign link]
    node [next] = new1
    pre [new1] =node
    next [new1] = NULL
Step-6: exit

Algorithm to insert desire position

```
Step-1: [Initialization]
            node = next [start]
            no [node] = 1
Step-2: [read node no that want to insert]
            input insert_no
Step-3: [perform insertion]
            repeat through step-4
              while node # NULL
Step-4: [check is insert_no = node_no]
            if insert_no = node_no
                  new1 = allocation
                  no[value] = value
                  [next [pre[node]]] = new1
                  pre [new1] = node
                  pre[node] = new1
            return
            else
                  [Move the pointer]
                        node = next [node]
            end if
                        node_no = node_no + 1

Step-5: exit
```

Deletion in doubly linked list

Algorithm to delete first node in doubly linked list

```
Step-1: [Initialization]
            node= next[start]  [pointer to the first node in the list]
Step-2: [perform deletion operation]
            if node = NULL
                  output " list is empty"
            exit

Step-3: [perform deletion]
            next [per [node]] = next [node]
            pre [next [node]] = pre [node]
            delete [node]

Step-4: exit
```

Algorithm to delete last node in the doubly linked list

```
Step-1: [Initialization]
            node = next [start]

Step-2: [Initialization counter]
            node_no = 0

Step-3: [Check the list]
            if node = NULL
                  output  "list  is empty"
            exit

Step-4: [count the number of node available in the list]
            repeat while node # NULL
                  node = next [node]
                  node_no = node_no +1

Step-5: [perform deletion]
            node = start
            repeat through while
                  node_no # 1
                  node = next [node]
                  node_no = node_no -1

Step-6: [check the list]
            if node_no = 1
                  next [pre [node]] = next [node]
                  pre [next [node]] = pre [node]
                  next [node] = NULL
                  delete [node]

Step-7: exit
```

Algorithm to delete at desire position in doubly linked list]

```
Step-1: [Initialization]
        node = next [start]

Step-2: [set the counter]
        node_no = 1
        flag = 0
Step-3: [check the list]
        if node = NULL
            output "list is empty"
        exit
Step-4: [Input the node number to be deleted]
      delete_no = value
Step-5: [perform deletion]
        repeat while node # NULL
        if delete_no = node_no
            next [pre [node]] = next [node]
            pre [next [node]] = pre [node]
            delete [node]
        else
            node =next [node]
            node_no = node_no + 1

Step-6: exit.
```

Program 1:

// Create Linked List and traverse node in linked list

```
#include <stdio.h>

#include <stdlib.h>

struct link {

    int no;

    struct link *next;

};


struct link *node, *start;

void create_link(struct link *node);

void display(struct link *node);


int main() {

    node = (struct link *)malloc(sizeof(struct link));

    start = node; // Set start to the first node

    printf("\nCreate linked list\n");

    create_link(node);

    printf("\nOutput\n");

    display(start->next); // Pass the first node after start to display

    free(start); // Free the memory allocated for start

    return 0;

}


void create_link(struct link *node) {
```

```
    char ans;
    printf("\nEnter 'n' for break: ");
    while ((ans = getchar()) != 'n') {
        node->next = (struct link *)malloc(sizeof(struct link));
        node = node->next;
        printf("\nEnter data for node: ");
        scanf("%d", &node->no);
        node->next = NULL;
        fflush(stdin);
        printf("\nEnter 'n' for break: ");
    }
}
void display(struct link *node) {
    while (node != NULL) {
        printf("%d -> ", node->no);
        node = node->next;
    }
    printf("NULL\n");                 }
```

Output:

```
Create linked list
Enter 'n' for break:
Enter data for node: 5
Enter 'n' for break:
Enter data for node: 10
Enter 'n' for break:
Enter data for node: 15
Enter 'n' for break: n

Output
5 -> 10 -> 15 -> NULL
```

Certainly! Let's break down the output step by step:

1. **Create linked list**: This is the initial message indicating the start of the process to create a linked list.

2. **Enter 'n' for break**: This prompt asks for user input. You're prompted to enter data for each node of the linked list. If you enter 'n', the process will stop.

3. **Enter data for node**: After the prompt, you enter the data for the first node, which is 5.

4. **Enter 'n' for break**: After entering the data for the first node, you're again prompted to enter 'n' if you want to stop creating nodes or continue adding more nodes.

5. **Enter data for node**: Since you didn't enter 'n', you proceed to add data for the second node, which is 10.

6. **Enter 'n' for break**: Again prompted for 'n', indicating whether you want to stop adding nodes or continue.

7. **Enter data for node**: Once more, you continue adding data for the third node, which is 15.

8. **Enter 'n' for break: n**: This time you enter 'n', indicating that you want to stop adding nodes.

9. **Output**: After you finish inputting data, the program displays the elements of the linked list: `5 -> 10 -> 15 -> NULL`. This shows the data contained in each node of the linked list, separated by arrows, with 'NULL' indicating the end of the list.

So, the output reflects the successful creation and display of a linked list containing three nodes with the values 5, 10, and 15.

Program 2:

// Insertion in singly linked list (Insert First, Last and node at desire position)

```
#include<stdio.h>

#include<stdlib.h>


struct link {

   int no;
```

```
    struct link *next;
};

struct link *start;

void create_link();
void display();
void insert_first();
void insert_last();
void insert_desired_position();

int main() {
    int ch;
    start = NULL;
    do {
        printf("\nSingly Linked List");
        printf("\n---------------------------");
        printf("\n1. Create Link");
        printf("\n2. Traverse Link");
        printf("\n3. Insert First node");
        printf("\n4. Insert Last node");
        printf("\n5. Insert node at Desired Position");
        printf("\n6. Exit");
        printf("\n-----------------------------");
```

```
printf("\nEnter your choice: ");
scanf("%d", &ch);
switch(ch) {
   case 1:
      create_link();
      break;
   case 2:
      printf("\nOutput\n");
      display();
      break;
   case 3:
      insert_first();
      printf("\nAfter Inserting First Node\n");
      display();
      break;
   case 4:
      insert_last();
      printf("\nAfter Inserting Last Node\n");
      display();
      break;
   case 5:
      insert_desired_position();
      printf("\nAfter Inserting At Desired Position\n");
      display();
```

```
            break;
          case 6:
            exit(0);
          default:
            printf("\nWrong Choice");
        }
    } while(ch != 6);

    return 0;
}

void create_link() {
    struct link *node, *temp;
    char ans;

    start = (struct link *)malloc(sizeof(struct link));
    start->next = NULL;
    temp = start;

    printf("\nEnter 'n' for break: ");
    fflush(stdin);
    ans = getchar();

    while(ans != 'n') {
```

```
        node = (struct link *)malloc(sizeof(struct link));
        printf("\nEnter data for node: ");
        scanf("%d", &node->no);
        node->next = NULL;
        temp->next = node;
        temp = temp->next;
        fflush(stdin);
        printf("\nEnter 'n' for break: ");
        ans = getchar();
    }
}

void display() {
    struct link *temp = start->next;
    while(temp != NULL) {
        printf("%d -> ", temp->no);
        temp = temp->next;
    }
    printf("NULL\n");
}

void insert_first() {
    struct link *node = (struct link *)malloc(sizeof(struct link));
    printf("\nInsert data for first node: ");
```

```
    scanf("%d", &node->no);
    node->next = start->next;
    start->next = node;
}

void insert_last() {
    struct link *node = (struct link *)malloc(sizeof(struct link));
    struct link *temp = start;
    printf("\nInsert data for last node: ");
    scanf("%d", &node->no);
    while(temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = node;
    node->next = NULL;
}

void insert_desired_position() {
    int position, node_no = 1;
    struct link *node, *temp = start;
    printf("\nEnter position where you want to insert new node: ");
    scanf("%d", &position);
    while(temp != NULL && node_no < position) {
        temp = temp->next;
```

```
        node_no++;
    }
    if(temp == NULL) {
        printf("\nPosition not found\n");
        return;
    }
    node = (struct link *)malloc(sizeof(struct link));
    printf("\nInsert data for new node: ");
    scanf("%d", &node->no);
    node->next = temp->next;
    temp->next = node;
}
```

Output:

```
Singly Linked List
--------------------------
1. Create Link
2. Traverse Link
3. Insert First node
4. Insert Last node
5. Insert node at Desired Position
6. Exit
----------------------------
Enter your choice: 1
Enter 'n' for break:
```

```
Enter data for node: 5

Enter 'n' for break: n

Singly Linked List

---------------------------

1. Create Link

2. Traverse Link

3. Insert First node

4. Insert Last node

5. Insert node at Desired Position

6. Exit

-----------------------------

Enter your choice: 2

Output

5 -> NULL

Singly Linked List

---------------------------

1. Create Link

2. Traverse Link

3. Insert First node

4. Insert Last node

5. Insert node at Desired Position

6. Exit

-----------------------------
```

```
Enter your choice: 3

Insert data for first node: 3

After Inserting First Node

3 -> 5 -> NULL

Singly Linked List

--------------------------

1. Create Link

2. Traverse Link

3. Insert First node

4. Insert Last node

5. Insert node at Desired Position

6. Exit


Enter your choice: 4

Insert data for last node: 8

After Inserting Last Node

3 -> 5 -> 8 -> NULL

Singly Linked List

--------------------------

1. Create Link

2. Traverse Link

3. Insert First node

4. Insert Last node

5. Insert node at Desired Position
```

```
6. Exit

-----------------------------

Enter your choice: 5

Enter position where you want to insert new node: 2

Insert data for new node: 7

After Inserting At Desired Position

3 -> 7 -> 5 -> 8 -> NULL

Singly Linked List

---------------------------

1. Create Link

2. Traverse Link

3. Insert First node

4. Insert Last node

5. Insert node at Desired Position

6. Exit

-----------------------------

Enter your choice: 6
```

Program 3:

// Deletion in singly linked list (Delete First, Last and node at desire position)

```
#include<stdio.h>

#include<stdlib.h>

struct link   {

   int no;
```

```
    struct link *next;
};
struct link *node,*start,*pre;
void create_link(struct link *node);
void display(struct link *node);
void delete_first(struct link *node);
void delete_last(struct link *node);
void delete_desireposition(struct link *node);

int main()
{
    int ch;
    start = (struct link *)malloc(sizeof(struct link)); // Allocate memory for start
    start->next = NULL; // Initialize start->next
    do
    {
        printf("\n Singly Linked List");
        printf("\n---------------------------");
        printf("\n 1. Create Link");
        printf("\n 2. Traverse Link");
        printf("\n 3. Delete First node");
        printf("\n 4. Delete Last node");
        printf("\n 5. Delete node at DesirePosition");
        printf("\n 6. Exit");
```

```
printf("\n-----------------------------");
printf("\n Enter your choice:-");
scanf("%d",&ch);
switch(ch)
{
   case 1:
      node=(struct link *)malloc(sizeof(struct link));
      create_link(node);
      break;
   case 2:
      printf("\n Output\n");
      display(node);
      break;
   case 3:
      delete_first(node);
      printf("\n After Deleting First Node");
      display(node);
      break;
   case 4:
      delete_last(node);
      printf("\n After Deleting Last Node");
      display(node);
      break;
   case 5:
```

```
            delete_desireposition(node);
            printf("\n After Deleting At Desire Position");
            display(node);
            break;
          case 6:
            exit(0);
            break;
          default:
            printf("\n Wrong Choice");
        }
    } while(ch!=6);

    free(start); // Free the memory allocated for start
    return 0;
}
void create_link(struct link *node)
{
    char ans;
    start->next=NULL;
    node=start;
    fflush(stdin);
    printf("\n Enter 'n' for break:-");
    ans=getchar();
    while(ans!='n')
```

```
    {
        node->next=(struct link *)malloc(sizeof(struct link));
        node=node->next;
        printf("\n Enter data for node:-");
        scanf("%d",&node->no);
        node->next=NULL;
        fflush(stdin);
        printf("\n Enter 'n' for break:-");
        ans=getchar();
    }
}

void display(struct link *node)
{
    node=start->next;
    while(node)
    {
        printf("\n %d",node->no);
        node=node->next;
    }
}

void delete_first(struct link *node)
{
```

```
    if(start->next == NULL)
    {
        printf("\n List is empty");
        return;
    }

    node=start->next;
    start->next=node->next;
    free(node);
}

void delete_last(struct link *node)
{
    if(start->next == NULL)
    {
        printf("\n List is empty");
        return;
    }

    pre=start;
    node=start->next;
    while(node->next != NULL)
    {
        pre=node;
```

```
        node=node->next;
    }
 pre->next=NULL;
    free(node);
}

void delete_desireposition(struct link *node)
{
    if(start->next == NULL)
    {
        printf("\n List is empty");
        return;
    }
int delete_no;
    printf("\n Enter position where you want to delete node:-");
    scanf("%d",&delete_no);

    pre=start;
    node=start->next;
    int pos = 1;
    while(node != NULL && pos < delete_no)
    {
        pre=node;
        node=node->next;
```

```
        pos++;
    }
     if(node == NULL)
    {
        printf("\n Position not found");
        return;
    }
   pre->next=node->next;
    free(node);
}
```

Output:

```
Singly Linked List
--------------------------
1. Create Link
2. Traverse Link
3. Delete First node
4. Delete Last node
5. Delete node at DesirePosition
6. Exit
----------------------------
Enter your choice:-1

Enter 'n' for break:-2
Enter data for node:-2
```

```
Enter 'n' for break:-n


 Singly Linked List
--------------------------
1. Create Link
2. Traverse Link
3. Delete First node
4. Delete Last node
5. Delete node at DesirePosition
6. Exit
----------------------------
Enter your choice:-2


 Output


 2
--------------------------
1. Create Link
2. Traverse Link
3. Delete First node
4. Delete Last node
5. Delete node at DesirePosition
6. Exit
----------------------------
Enter your choice:-3
```

```
After Deleting First Node
---------------------------
1. Create Link
2. Traverse Link
3. Delete First node
4. Delete Last node
5. Delete node at DesirePosition
6. Exit
-----------------------------
Enter your choice:-2

Output
---------------------------
1. Create Link
2. Traverse Link
3. Delete First node
4. Delete Last node
5. Delete node at DesirePosition
6. Exit
-----------------------------
Enter your choice:-6
```

Program 4:

// create and traverse doubly linked list

```
#include <stdio.h>

#include <stdlib.h>


struct link

{

   int no;

   struct link *next;

   struct link *pre;

};


struct link *start;


void create_link();

void display();


int main()

{

   printf("\nCreating doubly linked list\n");

   start = (struct link *)malloc(sizeof(struct link));

   if (start == NULL)

   {

      printf("Memory allocation failed.");
```

```
        return 1;
    }
    create_link();
    printf("\nOutput\n");
    display();
    free(start); // Free allocated memory
    return 0;
}

void create_link()
{
    char ans;
    struct link *node = start;
    printf("\nEnter 'n' for break: ");
    while ((ans = getchar()) != 'n')
    {
        node->next = (struct link *)malloc(sizeof(struct link));
        if (node->next == NULL)
        {
            printf("Memory allocation failed.");
            exit(1);
        }
        node->next->pre = node;
        node = node->next;
```

```
        printf("Enter data for node: ");

        scanf("%d", &node->no);

        node->next = NULL;

        printf("\nEnter 'n' for break: ");

        getchar(); // Consume newline character

    }

}

void display()

{     struct link *node = start->next;

    while (node)

    {

        printf("%d\n", node->no);

        node = node->next;                        }

}
```

Output:-

```
Creating doubly linked list
Enter data for node: 5
Enter 'n' for break:
Enter data for node: 10
Enter 'n' for break:
Enter data for node: 15
Enter 'n' for break: n

Output
5
10
15
```

Sure, let's break down the output step by step:

1. The program starts by prompting the user to enter data for each node of the linked list.

2. The user enters '5' as data for the first node, then '10' for the second node, and finally '15' for the third node.

3. After each data entry, the program prompts the user whether to continue entering data for the next node. Since the user enters 'n' after the third node, the loop terminates.

4. The program then proceeds to display the elements of the linked list.

5. It starts traversing the linked list from the `start` pointer's `next` node.

6. It prints the data of each node: 5, 10, and 15.

7. Once it reaches the end of the list (where `next` pointer is NULL), the loop terminates.

So, the output simply displays the data entered for each node of the doubly linked list, which are 5, 10, and 15, respectively.

Program 5:

// Insertion in doubly linked list (Insert First, Last and node at desire position)

```
#include<stdio.h>

#include<stdlib.h>


struct link

{

   int no;

   struct link *next;

   struct link *pre;

};

struct link *start;

void create_link();

void display();

void insert_first();

void insert_last();

void insert_desireposition();


int main()

{

   int ch;

   start = (struct link *)malloc(sizeof(struct link)); // Allocating memory for start pointer

   if(start == NULL) {

      printf("Memory allocation failed.");
```

```
    return 1;
  }
  do {
    printf("\n Doubly Linked List");
    printf("\n--------------------------");
    printf("\n 1. Create Linked list");
    printf("\n 2. Traverse Linked list");
    printf("\n 3. Insert First node");
    printf("\n 4. Insert Last node");
    printf("\n 5. Insert node at Desire Position");
    printf("\n 6. Exit");
    printf("\n----------------------------");
    printf("\n Enter your choice:-");
    scanf("%d",&ch);

    switch(ch)
    {
      case 1:
          create_link();
          printf("\n List is as follow");
          display();
          break;
      case 2:
          printf("\n Output\n");
```

```
            display();
            break;
        case 3:
            insert_first();
            printf("\n After Inserting First Node");
            display();
            break;
        case 4:
            insert_last();
            printf("\n After Inserting Last Node");
            display();
            break;
        case 5:
            insert_desireposition();
            printf("\n After Inserting At Desire Position");
            display();
            break;
        case 6:
            exit(0);
            break;
        default:
            printf("\n Wrong Choice");
    }
} while(ch!=6);
```

```
    free(start); // Free allocated memory
    return 0;
}
void create_link()
{
    char ans;
    struct link *node = start;
    start->next=NULL;
    start->pre=NULL;

    printf("\n Enter 'n' for break:-");
    scanf(" %c", &ans); // Correct way to read a single character

    while(ans!='n')
    {
        node->next=(struct link *)malloc(sizeof(struct link));
        node->next->pre=node;
        node=node->next;
        printf("\n Enter data for node:-");
        scanf("%d",&node->no);
        node->next=NULL;
        printf("\n Enter 'n' for break:-");
        scanf(" %c", &ans); // Correct way to read a single character
```

```
   }
}
void display()
{
    struct link *node = start->next;
    while(node)
    {
        printf("\n %d",node->no);
        node=node->next;
    }
}
void insert_first()
{
    struct link *node = start->next;
    struct link *new1 = (struct link *)malloc(sizeof(struct link));
    printf("\n Insert data for first node:-");
    scanf("%d",&new1->no);
    if(node == NULL) { // If list is empty
        new1->next = NULL;
        new1->pre = NULL;
        start->next = new1;
        return;
    }
```

```
    node->pre->next=new1;
    new1->pre=node->pre;
    new1->next=node;
    node->pre=new1;
}
void insert_last()
{
    struct link *node = start->next;
    struct link *new1 = (struct link *)malloc(sizeof(struct link));
     printf("\n Insert data for last node:-");
    scanf("%d",&new1->no);
    if(node == NULL) { // If list is empty
        new1->next = NULL;
        new1->pre = NULL;
        start->next = new1;
        return;
    }
     while(node->next)
    {
        node=node->next;
    }
    node->next=new1;
    new1->pre=node;
    new1->next=NULL;
```

```
}
void insert_desireposition()
{
    int node_no=1,insert_no,flag=0;
    struct link *node = start->next;
    struct link *new1;
     printf("\n Enter position where you want to insert new node:-");
    scanf("%d",&insert_no);

    while(node)
    {
        if(node_no==insert_no)
        {
            new1=(struct link *)malloc(sizeof(struct link));
            printf("\n Insert data for new node:-");
            scanf("%d",&new1->no);
              if(node->pre == NULL) {          // If inserting at the beginning
                new1->pre = NULL;
                new1->next = node;
                node->pre = new1;
                start->next = new1;
            } else {
                node->pre->next=new1;
                new1->pre=node->pre;
```

```
            new1->next=node;
            node->pre=new1;
          }
          flag=1;
          break;
        }
        else
        {
          node=node->next;
        }
        node_no++;
    }  if(flag==0)
    { printf("\n Position not found");
    }
}
```

Output:

```
Doubly Linked List
---------------------------
1. Create Linked list
2. Traverse Linked list
3. Insert First node
4. Insert Last node
5. Insert node at Desire Position
6. Exit
```

```
-----------------------------
Enter your choice:-1


Enter 'n' for break:-y
Enter data for node:-10
Enter 'n' for break:-y
Enter data for node:-20
Enter 'n' for break:-y
Enter data for node:-30
Enter 'n' for break:-n
List is as follows:
10
20
30


Doubly Linked List
---------------------------
1. Create Linked list
2. Traverse Linked list
3. Insert First node
4. Insert Last node
5. Insert node at Desire Position
6. Exit
-----------------------------
```

```
Insert data for first node:-5
After Inserting First Node:
5
10
20
30

Doubly Linked List
---------------------------
1. Create Linked list
2. Traverse Linked list
3. Insert First node
4. Insert Last node
5. Insert node at Desire Position
6. Exit
-----------------------------
Enter your choice:-4
Insert data for last node:-40
After Inserting Last Node:
5
10
20
30
40
```

```
Doubly Linked List

--------------------------

1. Create Linked list

2. Traverse Linked list

3. Insert First node

4. Insert Last node

5. Insert node at Desire Position

6. Exit

----------------------------

Enter your choice:-5


Enter position where you want to insert new node:-3

Insert data for new node:-15

After Inserting At Desire Position:

5

10

15

20

30

40


Doubly Linked List

--------------------------

1. Create Linked list

2. Traverse Linked list
```

```
3. Insert First node

4. Insert Last node

5. Insert node at Desire Position

6. Exit

----------------------------

Enter your choice:-6
```

Program 6:

// Deletion in doubly linked list(Delete First, Last and node at desire position)

```
#include<stdio.h>

#include<stdlib.h>


struct link

{

   int no;

   struct link *next;

   struct link *pre;

};


struct link *node, *start;


void create_link(struct link *node);

void display(struct link *node);

void delete_first(struct link *node);
```

```
void delete_last(struct link *node);

void delete_desireposition(struct link *node);


int main()

{

   int ch;

   start = (struct link *)malloc(sizeof(struct link)); // Allocating memory for start pointer

   if(start == NULL) {

      printf("Memory allocation failed.");

      return 1;

   }


   do

   {

      printf("\n Doubly Linked List");

      printf("\n---------------------------");

      printf("\n 1. Create Link");

      printf("\n 2. Traverse Link");

      printf("\n 3. Delete First node");

      printf("\n 4. Delete Last node ");

      printf("\n 5. Delete node at Desire Position ");

      printf("\n 6. Exit");

      printf("\n-----------------------------");

      printf("\n Enter your choice:-");
```

```
scanf("%d",&ch);

switch(ch)
{
   case 1:
      node=(struct link *)malloc(sizeof(struct link));
      create_link(node);
      printf("\n Linked list is as follow");
      display(node);
      break;
   case 2:
      printf("\n Output\n");
      display(node);
      break;
   case 3:
      delete_first(node);
      printf("\n After Deleting First Node");
      display(node);
      break;
   case 4:
      delete_last(node);
      printf("\n After Deleting Last Node");
      display(node);
      break;
```

```
        case 5:
          delete_desireposition(node);
          printf("\n After Deleting At Desire Position");
          display(node);
          break;
        case 6:
          exit(0);
          break;
        default:
          printf("\n Wrong Choice");
      }
  } while(ch!=6);

  free(start); // Free allocated memory
  return 0;
}

void create_link(struct link *node)
{
  char ans;
  start->next=NULL;
  start->pre=NULL;
  node=start;
```

```
    printf("\n Enter 'n' for break:-");
    while(getchar() != '\n'); // Clear input buffer properly
    ans=getchar();

    while(ans!='n')
    {
        node->next=(struct link *)malloc(sizeof(struct link));
        node->next->pre=node;
        node=node->next;
        printf("\n Enter data for node:-");
        scanf("%d",&node->no);
        node->next=NULL;
        while(getchar() != '\n'); // Clear input buffer properly
        printf("\n Enter 'n' for break:-");
        ans=getchar();
    }
}

void display(struct link *node)
{
    node=start->next;
    while(node)
    {
        printf("\n %d",node->no);
```

```
        node=node->next;
    }
}

void delete_first(struct link *node)
{
    if(start->next == NULL) {
        printf("\n List is empty");
        return;
    }

    node=start->next;
    start->next=node->next;
    if(node->next != NULL)
        node->next->pre=start;
    free(node);
}

void delete_last(struct link *node)
{
    if(start->next == NULL) {
        printf("\n List is empty");
        return;
    }
```

```
    node=start->next;
    while(node->next != NULL)
        node=node->next;

    if(node->pre == start) {
        free(node);
        start->next = NULL;
    } else {
        node->pre->next=NULL;
        free(node);
    }
}

void delete_desireposition(struct link *node)
{
    if(start->next == NULL) {
        printf("\n List is empty");
        return;
    }

    int delete_no, flag = 0;
    printf("\n Enter position where you want to delete node:-");
    scanf("%d",&delete_no);
```

```
node=start->next;
int node_no=1;

while(node)
{
   if(node_no==delete_no)
   {
      if(node->next != NULL)
         node->next->pre = node->pre;
      node->pre->next = node->next;
      free(node);
      flag = 1;
      break;
   }
   else
   {
      node=node->next;
   }
   node_no++;
}

if(flag == 0)
{
```

```
        printf("\n Position not found");
    }
}
```

Output:

```
Doubly Linked List
--------------------------
1. Create Link
2. Traverse Link
3. Delete First node
4. Delete Last node
5. Delete node at Desire Position
6. Exit
----------------------------
Enter your choice:-1
Enter 'n' for break:-y
Enter data for node:-10
Enter 'n' for break:-y
Enter data for node:-20
Enter 'n' for break:-y
Enter data for node:-30
Enter 'n' for break:-n
Linked list is as follow
10
20
```

```
30

Doubly Linked List

--------------------------

1. Create Link

2. Traverse Link

3. Delete First node

4. Delete Last node

5. Delete node at Desire Position

6. Exit

----------------------------

Enter your choice:-3

After Deleting First Node

20

30

Doubly Linked List

--------------------------

1. Create Link

2. Traverse Link

3. Delete First node

4. Delete Last node

5. Delete node at Desire Position

6. Exit

----------------------------

Enter your choice:-4
```

```
After Deleting Last Node

20


Doubly Linked List

---------------------------

1. Create Link

2. Traverse Link

3. Delete First node

4. Delete Last node

5. Delete node at Desire Position

6. Exit

-----------------------------

Enter your choice:-5

Enter position where you want to delete node:-1

After Deleting At Desire Position

Position not found


Doubly Linked List

---------------------------

1. Create Link

2. Traverse Link

3. Delete First node

4. Delete Last node

5. Delete node at Desire Position
```

```
6. Exit
----------------------------
Enter your choice:-6
```

Program 6:

// Searching in a linked list

```
#include<stdio.h>

#include<stdlib.h>


struct link {

    int no;

    struct link *next;

};


struct link *start = NULL;


void create_link();

void display();

void search();


int main() {

    create_link();
```

```
    printf("\nOutput\n");
    display();
    search();
    return 0;
}

void create_link() {
    char ans;
    struct link *node;
    start = (struct link *)malloc(sizeof(struct link));
    if(start == NULL) {
        printf("Memory allocation failed.");
        exit(1);
    }
    start->next = NULL;
    node = start;
    printf("\nEnter 'n' for break: ");
    ans = getchar();
    while(ans != 'n') {
        node->next = (struct link *)malloc(sizeof(struct link));
        if(node->next == NULL) {
            printf("Memory allocation failed.");
            exit(1);
        }
```

```
        node = node->next;
        printf("\nEnter data for node: ");
        scanf("%d", &node->no);
        node->next = NULL;
        fflush(stdin);
        printf("\nEnter 'n' for break: ");
        ans = getchar();
    }
}

void display() {
    struct link *node;
    node = start->next;
    while(node) {
        printf("\n%d", node->no);
        node = node->next;
    }
}

void search() {
    struct link *node;
    int search_node, node_no = 1, flag = 0;
    node = start->next;
    printf("\nEnter information which you want to search: ");
```

```
    scanf("%d", &search_node);
    while(node) {
        if(node->no == search_node) {
            printf("\nSearch is successful");
            printf("\nInformation %d is found at position %d", search_node, node_no);
            flag = 1;
            break;
        } else {
            node = node->next;
        }
        node_no++;
    }
    if(flag == 0) {
        printf("\nInformation not found");
    }
}
```

Output:

```
Enter 'n' for break:
Enter data for node: 10
Enter 'n' for break:
Enter data for node: 20
Enter 'n' for break:
Enter data for node: 30
Enter 'n' for break: n

Output
10
20
30

Enter information which you want to search: 20

Search is successful
Information 20 is found at position 2
```

Program 7:

//merge LinkList

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

struct link {
    int no;
    struct link *next;
};

void create_link(struct link *node);
void display(struct link *node);
void concat_link(struct link *list1, struct link *list2, struct link *list3);

int main() {
    struct link *list1 = (struct link *)malloc(sizeof(struct link));
    struct link *list2 = (struct link *)malloc(sizeof(struct link));
    struct link *list3 = (struct link *)malloc(sizeof(struct link));

    printf("\nCreate list -1 ");
```

```
    create_link(list1);
    printf("\nCreate list -2 ");
    create_link(list2);

    printf("\nOutput List -1\n");
    display(list1);
    printf("\nOutput List -2\n");
    display(list2);

    concat_link(list1, list2, list3);
    printf("\nList concatenation output\n");
    display(list3);

    // Free allocated memory
    free(list1);
    free(list2);
    free(list3);

    return 0;
}
```

```
void create_link(struct link *node) {
    char ans;
    printf("\nEnter 'n' for break: ");
    ans = getchar();
    while (ans != 'n') {
        printf("\nEnter data for node: ");
        scanf("%d", &node->no);
        node->next = (struct link *)malloc(sizeof(struct link));
        if (node->next == NULL) {
            printf("Memory allocation failed.");
            exit(1);
        }
        node = node->next;
        node->next = NULL;
        fflush(stdin);
        printf("\nEnter 'n' for break: ");
        ans = getchar();
    }
}
```

```
void display(struct link *node) {
    while (node != NULL) {
        printf("\n%d", node->no);
        node = node->next;
    }
}

void concat_link(struct link *list1, struct link *list2, struct link *list3) {
    // copy first list
    while (list1->next != NULL) {
        list3->no = list1->no;
        list3->next = (struct link *)malloc(sizeof(struct link));
        if (list3->next == NULL) {
            printf("Memory allocation failed.");
            exit(1);
        }
        list1 = list1->next;
        list3 = list3->next;
    }
```

```
    // copy second list
    while (list2->next != NULL) {
        list3->no = list2->no;
        list3->next = (struct link *)malloc(sizeof(struct link));
        if (list3->next == NULL) {
            printf("Memory allocation failed.");
            exit(1);
        }
        list2 = list2->next;
        list3 = list3->next;
    }
    list3->next = NULL;
}
```

Output:

```
Create list -1
Enter 'n' for break:
Enter data for node: 10
Enter 'n' for break:
Enter data for node: 20
Enter 'n' for break: n

Create list -2
Enter 'n' for break:
Enter data for node: 30
Enter 'n' for break:
Enter data for node: 40
Enter 'n' for break: n

Output List -1
10
20

Output List -2
30
40

List concatenation output
10
20
30
40
```

Program 8:

```
#include<stdio.h>

#include<stdlib.h>


struct link {

    int no;

    struct link *next;

};


void create_link(struct link **start);

void display(struct link *start);

struct link *reverse(struct link *start);


int main() {

    struct link *start = NULL;

    create_link(&start);

    printf("\nList before reversing:\n");

    display(start);


    start = reverse(start);


    printf("\nList after reversing:\n");
```

```
    display(start);

    return 0;
}

void create_link(struct link **start) {
    char ans;
    struct link *temp;
    printf("Enter data for node (enter 'n' to stop):\n");
    while (1) {
        temp = (struct link*)malloc(sizeof(struct link));
        if (!temp) {
            printf("Memory allocation failed.\n");
            exit(1);
        }
        scanf("%d", &temp->no);
        temp->next = NULL;

        if (*start == NULL) {
            *start = temp;
        } else {
            struct link *current = *start;
```

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

        printf("Enter 'n' to stop or any other key to continue: ");
        scanf(" %c", &ans);
        if (ans == 'n')
            break;
    }
}
void display(struct link *start) {
    while (start) {
        printf("%d -> ", start->no);
        start = start->next;
    }
    printf("NULL\n");
}
struct link *reverse(struct link *start) {
    struct link *prev = NULL;
    struct link *current = start;
```

```
  struct link *next = NULL;

  while (current != NULL) {

     next = current->next;

     current->next = prev;

     prev = current;

     current = next;

  }     start = prev;

  return start;

}
```

Output:

```
Enter data for node (enter 'n' to stop):
1
Enter 'n' to stop or any other key to continue: y
Enter data for node (enter 'n' to stop):
2
Enter 'n' to stop or any other key to continue: y
Enter data for node (enter 'n' to stop):
3
Enter 'n' to stop or any other key to continue: y
Enter data for node (enter 'n' to stop):
4
Enter 'n' to stop or any other key to continue: n

List before reversing:
1 -> 2 -> 3 -> 4 -> NULL

List after reversing:
4 -> 3 -> 2 -> 1 -> NULL
```