RCOEM

Shri Ramdeobaba College of Engineering and Management, Nagpur

Data Structure & Algorithms (DSA) : CCT203

Semester - III, B.Tech. CSE (Cyber Security)

UNIT III

GC Code: dyb4gii

Prof. Charanjeet Dadiyala

Course Objectives

1. CO1: To impart to students the basic concepts of data structures and algorithms.
2. CO2: To familiarize students on different searching and sorting techniques.
3. CO3: To prepare students to use linear (stacks, queues, linked lists) and nonlinear (trees, graphs) data structures.
4. CO4: To enable students to devise algorithms for solving real-world problems.

Text Books & Reference Books

Text Books:

1. Ellis Horowitz, Sartaj Sahni & Susan Anderson-Freed, Fundamentals of Data Structures in C, Second Edition, Universities Press, 2008.
2. Mark Allen Weiss; Data Structures and Algorithm Analysis in C; Second Edition; Pearson Education; 2002.
3. G.A.V. Pai; Data Structures and Algorithms: Concepts, Techniques and Application; First Edition; McGraw Hill; 2008.

Reference Books:

1. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein; Introduction to Algorithms; Third Edition; PHI Learning; 2009.
2. Ellis Horowitz, Sartaj Sahni and Sanguthevar Rajasekaran; Fundamentals of Computer Algorithms; Second Edition; Universities Press; 2008.
3. A. K. Sharma; Data Structures using C, Second Edition, Pearson Education, 2013.

Course Outcomes

On completion of the course the student will be able to

1. Recognize different ADTs and their operations and specify their complexities.
2. Design and realize linear data structures (stacks, queues, linked lists) and analyze their computation complexity.
3. Devise different sorting (comparison based, divide-and-conquer, distributive, and tree-based) and searching (linear, binary) methods and analyze their time and space requirements.
4. Design traversal and path finding algorithms for Trees and Graphs.

Singly Linked Lists:

- representation in memory
- Algorithms of several operations:
  traversing, searching, insertion, deletion, reversal, ordering, etc.

Doubly and Circular Linked Lists:

- operations and algorithmic analysis
- Linked representation of stacks and queues
- Header node linked lists

Linked Lists

- A linked list is a collection of data elements called **nodes** in which the linear representation is given by **links** from one node to the next node

Motivation:

- While declaring arrays, we have to **specify the size** of the array, which will **restrict the number of elements** that the array can store.
- But what if we are **not sure of the number of elements** in advance?
- There must be a data structure that **removes the restrictions on the maximum number** of elements and the **storage condition** to write efficient programs.
- **Answer** is **LINKED LIST**
- **Linked list is a data structure that is free from the aforementioned restrictions.**

Linked Lists

RCOEM
Shri Ramdeobaba College of Engineering and Management, Nagpur

- A linked list is a linear collection of data elements. These data elements are called **nodes**
- 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, and their variations.
- 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 contains two parts: Data and Pointer to the Address of next node.
- Last node, as not connected to any node, has NULL pointer (x), usually defined as -1.
- NULL defines the end of the list.
- A pointer variable START that stores the address of the first node in the list.

Linked Lists: C Code

- In C, we can implement a linked list using the following code:

```
struct node {
        int data;
        struct node *next;
};
```

Linked Lists: Memory Representation

- To create Linked List, you must have a NODE.
- NODE, i.e., DATA and NEXT
- DATA = store the information part
- NEXT = store the address of the next node in sequence.
- The computer maintains a list of all free memory cells. This list of available space is called the **free pool.**

In this fig,

- START = 1 Starting address (Identify the start of the list)
- First data stored = H and NEXT stores 4
- Next data stored = E and NEXT stores 7
- Next data stored = L and NEXT stores 8
- Next data stored = L and NEXT stores 10
- Next data stored = 0 and NEXT stores -1 (NULL)
- NULL denotes end of the list.
- Here, Shaded memory can be used by other programs.

START
1

	Data	Next
1	H	4
2		
3		
4	E	7
5		
6		
7	L	8
8	L	10
9		
10	0	-1

Linked Lists: Two linked lists simultaneously maintained in the memory

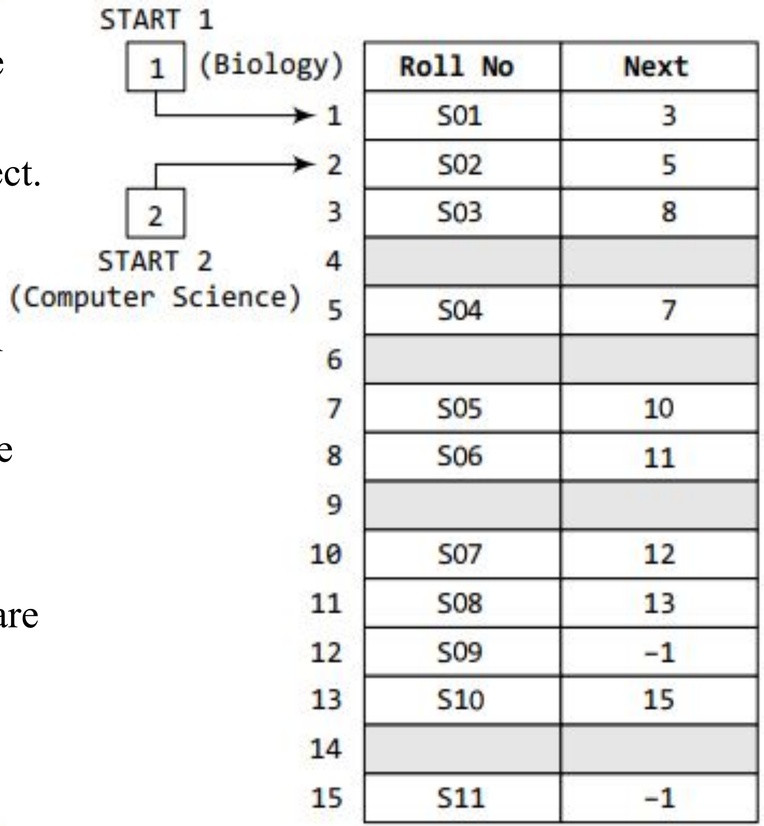
- For example, the students of Class XI of Science group are asked to choose between Biology and Computer Science
- Now, we will maintain **two linked lists**, one for each subject.
- One for Biology and One for CS

In fig,

- Two different linked lists are simultaneously maintained in the memory.
- There is no ambiguity in traversing through the list because each list maintains a **separate Start pointer**

We can conclude that,

- Roll numbers of the students who have opted for Biology are S01, S03, S06, S08, S10, and S11.
- Roll numbers of the students who chose CS are S02, S04, S05, S07, and S09.



	Roll No	Next
1	S01	3
2	S02	5
3	S03	8
4		
5	S04	7
6		
7	S05	10
8	S06	11
9		
10	S07	12
11	S08	13
12	S09	-1
13	S10	15
14		
15	S11	-1

Linked Lists: How the NEXT pointer is used to store the data alphabetically

	Roll No	Name	Aggregate	Grade	Next
1	S01	Ram	78	Distinction	6
2	S02	Shyam	64	First division	14
3					
4	S03	Mohit	89	Outstanding	17
5					
6	S04	Rohit	77	Distinction	2
7	S05	Varun	86	Outstanding	10
8	S06	Karan	65	First division	12
9					
10	S07	Veena	54	Second division	-1
11	S08	Meera	67	First division	4
12	S09	Krish	45	Third division	13
13	S10	Kusum	91	Outstanding	11
14	S11	Silky	72	First division	7
15					
16					
17	S12	Monica	75	Distinction	1
18	S13	Ashish	63	First division	19
19	S14	Gaurav	61	First division	8

START $\boxed{18}$

Array vs Linked Lists

- Both arrays and linked lists are a linear collection of data elements.
- Insertions and deletions can be done at any point in the array in a constant time.

- **Arrays:**
    - An array stores its elements in consecutive memory locations.
    - An array allows random access of data using the value of index.
    - Array has restriction of number of elements added, if specified MAX_SIZE.

- **Linked List:**
    - A linked list does not store its nodes in consecutive memory locations.
    - A linked list does not allow random access of data.
    - Nodes in a linked list can be accessed only in a sequential manner.
    - We can add any number of elements in the list
    - Extra space is required for storing the address of next nodes.

Memory Allocation and Deallocation for a Linked List

Before and After Insertion

RCOEM
Shri Ramdeobaba College of Engineering and Management, Nagpur

(a) Students' linked list

START = 1 (Biology)

	Roll No	Marks	Next
1	S01	78	2
2	S02	84	3
3	S03	45	5
4			
5	S04	98	7
6			
7	S05	55	8
8	S06	34	10
9			
10	S07	90	11
11	S08	87	12
12	S09	86	13
13	S10	67	15
14			
15	S11	56	-1

(b) linked list after the insertion of new student's record

START = 1 (Biology)

	Roll No	Marks	Next
1	S01	78	2
2	S02	84	3
3	S03	45	5
4	S12	75	-1
5	S04	98	7
6			
7	S05	55	8
8	S06	34	10
9			
10	S07	90	11
11	S08	87	12
12	S09	86	13
13	S10	67	15
14			
15	S11	56	4

Free Pool in Linked List

- When we delete a node from a linked list, **OS** changes the status of the memory occupied by it **from occupied to available.**
- The computer does it on its own **without any intervention** from the user or the programmer.
- As a programmer, you just have to take care of the code to perform insertions and deletions in the list
- The computer maintains a list of all free memory cells and this list is called the **free pool.**
- For the free pool (which is a linked list of all free memory cells), we have a pointer variable **AVAIL** which stores the **address of the first free space.**

START
1
(Biology)

4
AVAIL

	Roll No	Marks	Next
1	S01	78	2
2	S02	84	3
3	S03	45	5
4			6
5	S04	98	7
6			9
7	S05	55	8
8	S06	34	10
9			14
10	S07	90	11
11	S08	87	12
12	S09	86	13
13	S10	67	15
14			-1
15	S11	56	-1

Free Pool & Garbage Collection in Linked List

- When we delete a particular node from an existing linked list or delete the entire linked list, the space occupied by it must be given back to the free pool so that the memory can be reused by some other program that needs memory space
- The operating system scans through all the memory cells and marks those cells that are being used by some program
- Then it collects all the cells which are not being used and adds their address to the free pool, so that these cells can be reused by other programs. This process is called **garbage collection.**

START
1
(Biology)

4
AVAIL

	Roll No	Marks	Next
1	S01	78	2
2	S02	84	3
3	S03	45	5
4			6
5	S04	98	7
6			9
7	S05	55	8
8	S06	34	10
9			14
10	S07	90	11
11	S08	87	12
12	S09	86	13
13	S10	67	15
14			-1
15	S11	56	-1

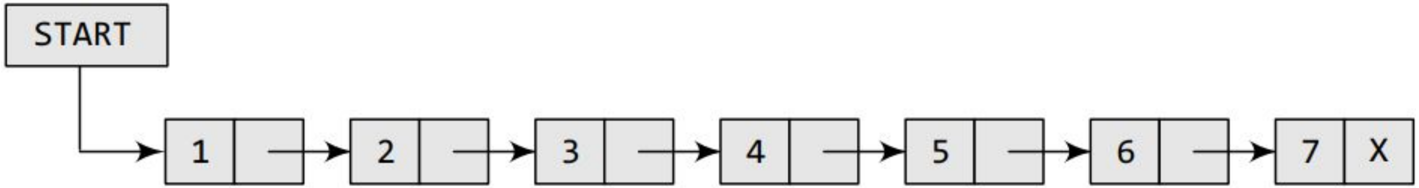
Singly Linked Lists

Singly Linked List

- A singly linked list is the **simplest type of linked list**.
- Here, **every node** contains **some data** and **a pointer to the next node** of the same data type
- A singly linked list allows **traversal of data only in one way**

Singly Linked Lists Allowable Operations:

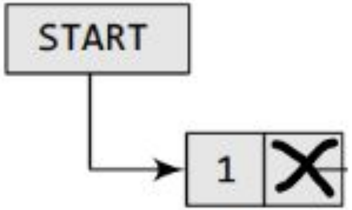
- Traversing a Linked List
- Searching for a Value in a Linked List
- Inserting a New Node in a Linked List (at the beginning, at the end, after or before a given node)
- Deleting a Node from a Linked List (first node, last node, node after a given node)



Create a node

RCOEM
Shri Ramdeobaba College of Engineering and Management, Nagpur

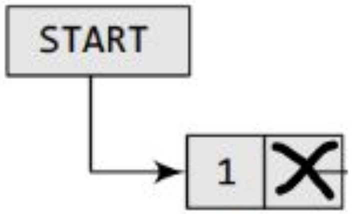
Case 0: When the linked list is newly created



- Declare the structure of node and initialize the start pointer as NULL
- Create a node by:
  - Allocate the memory of a node, i.e. DATA + next address
  - Take the data from user and set to the newly created node’s data.
  - Set the next pointer of the newly created node as NULL to mark the end of the node
  - Point the start pointer to the newly created node

Create a node

Case 0: When the linked list is newly created



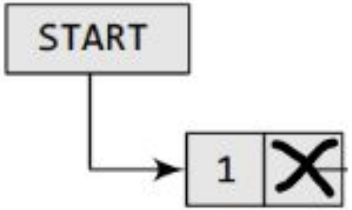
- Declare the structure of node and initialize the start pointer as NULL

```
struct node{
        int data;
        struct node *next;
}
struct node *start = NULL;
struct node *create(struct node *);
```

- Create a node by:
    - Allocate the memory of a node, i.e. DATA + next address
    - Take the data from user and set to the newly created node's data.
    - Set the next pointer of the newly created node as NULL to mark the end of the node
    - Point the start pointer to the newly created node

Create a node

Case 0: When the linked list is newly created (start=NULL)



- Declare the structure of node and initialize the start pointer as NULL
- Create a node by:
  - Allocate the memory of a node, i.e. DATA + next address
    new_node=(struct node*)malloc(sizeof(struct node));
  - Take the data from user and set to the newly created node's data.
    (Ex: **int num**)
  - Assign the data to the data part of newly created node
    new_node->data = num;
  - Set the next pointer of the newly created node as NULL to mark the end of the node
    new_node->next = NULL;
  - Point the start pointer to the newly created node
    start=new_node;

Create a node: create() method

Case 0: When the linked list is newly created (start=NULL)

```
struct node *create_ll(struct node *start) {
struct node *new_node;
int num;
printf("\n Enter the data : ");
scanf("%d", &num);
 new_node = (struct node*)malloc(sizeof(struct node));
 new_node -> data=num;
 new_node -> next = NULL;
 start = new_node;
return start;
}
```

Create a node

Case 1: When the linked list is already created and a new node is to be created

- Declare the structure of node
- Create a node by:
  - Allocate the memory of a node, i.e. DATA + next address
  - Take the data from user and set to the newly created node’s data.

Now, the question is,

Where to add this newly created node?

- At the beginning of the list? Case 1
- At the end of the list? Case 2
- Before a node? Case 3
- After a node? Case 4

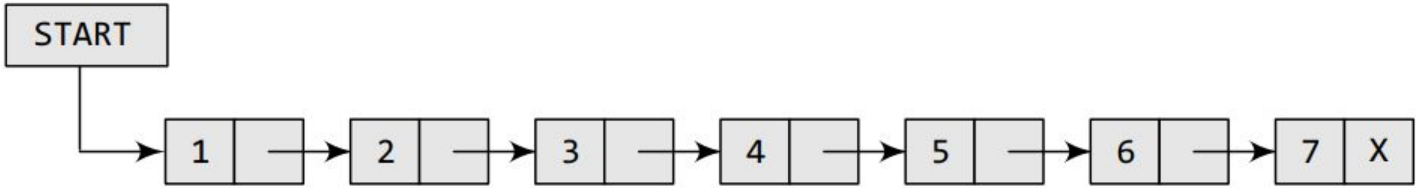
Let us understand, Insertion of a node in a Linked List

Inserting a New Node in a Linked List

- Case 1: The new node is inserted at the beginning.
- Case 2: The new node is inserted at the end.
- Case 3: The new node is inserted after a given node.
- Case 4: The new node is inserted before a given node.

Before Insertion we check, OVERFLOW Condition

- Overflow is a condition that occurs when **AVAIL = NULL** or no free memory cell is present
- When this condition occurs, the program must give an appropriate message



Create a new node and Insert it at the beginning

RCOEM
Shri Ramdeobaba College of Engineering and Management, Nagpur

Case 1: When the linked list is already created and a new node is to be created and to be inserted at the beginning of the linked list

- Create a node:
    - Allocate the memory of a node, i.e. DATA + next address
      new_node=(struct node*)malloc(sizeof(struct node));
    - Take the data from user and set to the newly created node's data.
      (Ex: **int num**)
    - Assign the data to the data part of newly created node
      new_node->data = num;
- Insert the newly created node at the beginning of the linked list.
    - Assign the next pointer of the newly created node to the start of the linked list
      new_node->next=start;
    - Assign the start pointer to the new_node.
      start =new_node;

Inserting a New Node: at the beginning

RCOEM
Shri Ramdeobaba College of Engineering and Management, Nagpur

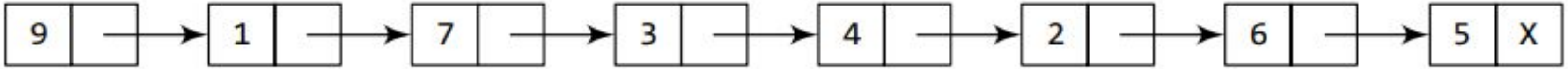
```
Step 1: IF AVAIL = NULL
              Write OVERFLOW
              Go to Step 7
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> NEXT = START
Step 6: SET START = NEW_NODE
Step 7: EXIT
```

Inserting a New Node: at the beginning

START

Allocate memory for the new node and initialize its DATA part to 9.

Add the new node as the first node of the list by making the NEXT part of the new node contain the address of START.



START

Now make START to point to the first node of the list.

START

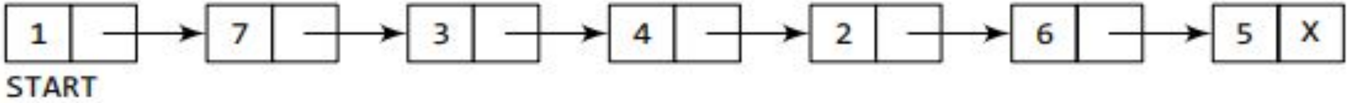
Create a new node and Insert it at the end

Case 2: When the linked list is already created and a new node is to be created and to be inserted at the end of the linked list

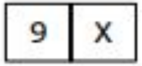
- Create a node:
  - Allocate the memory of a node, i.e. DATA + next address
    new_node=(struct node*)malloc(sizeof(struct node));
  - Assign the data to the data part of newly created node
    new_node->data = num;
  - Assign NULL to the address part of newly created node
    new_node->next = NULL;
- Insert the newly created node at the end of the linked list.
  - Initialize new pointer to start : **ptr = start;**
  - Reach the last node of the linked list : **while(ptr -> next != NULL)**
    ptr = ptr -> next;

    and assign its address to the new_node : **ptr->next = new_node;**

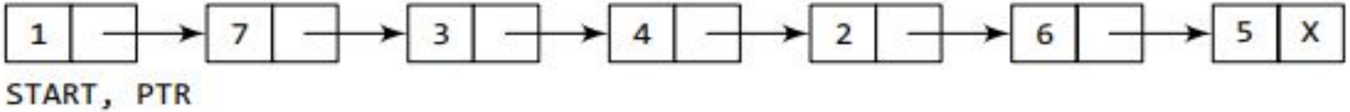
Inserting a New Node: at the end



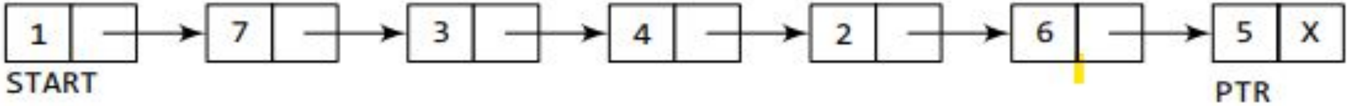
Allocate memory for the new node and initialize its DATA part to 9 and NEXT part to NULL.



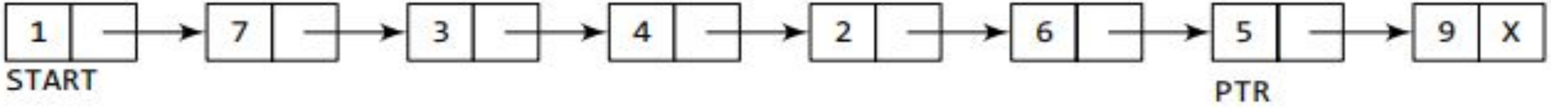
Take a pointer variable PTR which points to START.



Move PTR so that it points to the last node of the list.



Add the new node after the node pointed by PTR. This is done by storing the address of the new node in the NEXT part of PTR.



Inserting a New Node: at the end


```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 10
       [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL->NEXT
Step 4: SET NEW_NODE->DATA = VAL
Step 5: SET NEW_NODE->NEXT = NULL
Step 6: SET PTR = START
Step 7: Repeat Step 8 while PTR->NEXT != NULL
Step 8:     SET PTR = PTR->NEXT
       [END OF LOOP]
Step 9: SET PTR->NEXT = NEW_NODE
Step 10: EXIT
```

Create a new node and Insert it after a node

Case 3: When the linked list is already created and a new node is to be created and to be inserted after a given node

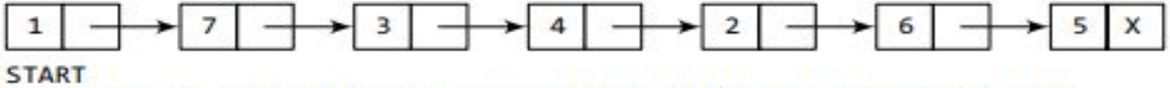
- Create a node:
  - Allocate the memory of a node: **new_node=(struct node*)malloc(sizeof(struct node));**
  - Assign the data to the node: **new_node->data = num;**
- Get the node after which the newly created node is to be inserted: **val**
- Create two pointers ptr and preptr and initialize them as start **ptr=start; preptr=ptr;**
- Until preptr->data does not points to the val:

```
while(preptr->data!=val)
    preptr=ptr;
    ptr=ptr->next;
```

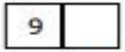
- Insert the new_node using preptr pointer:

```
preptr->next=new_node;
new_node->next=ptr;
```

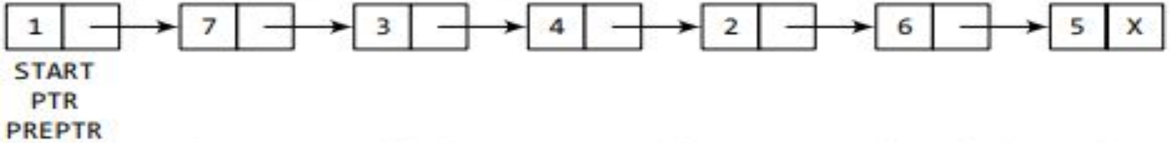
Inserting a New Node after a given node



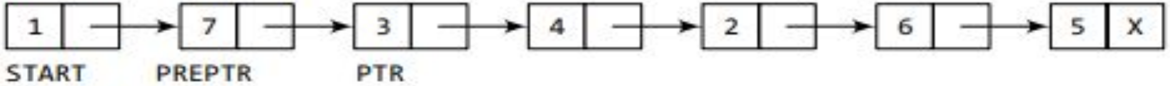
Allocate memory for the new node and initialize its DATA part to 9.

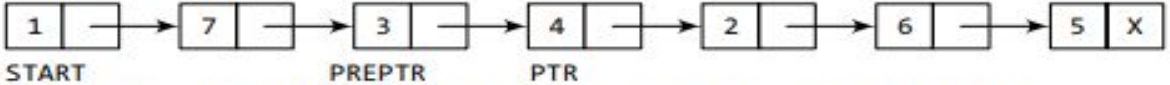


Take two pointer variables PTR and PREPTR and initialize them with START so that START, PTR, and PREPTR point to the first node of the list.

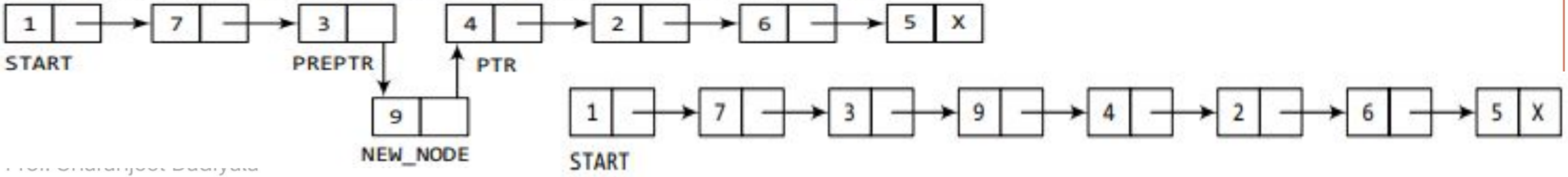


Move PTR and PREPTR until the DATA part of PREPTR = value of the node after which insertion has to be done. PREPTR will always point to the node just before PTR.

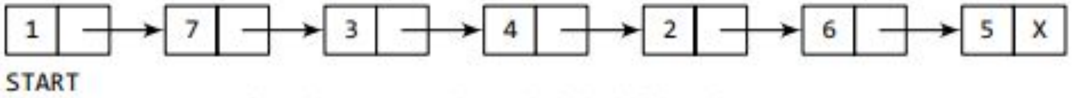




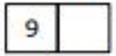
Add the new node in between the nodes pointed by PREPTR and PTR.



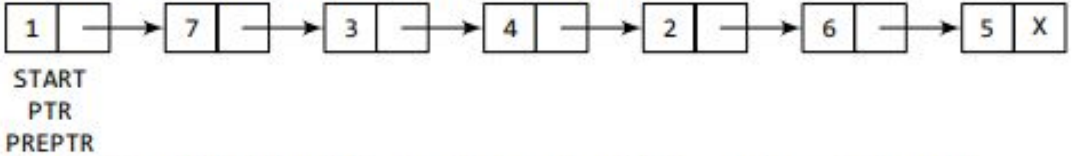
Inserting a New Node before a given node



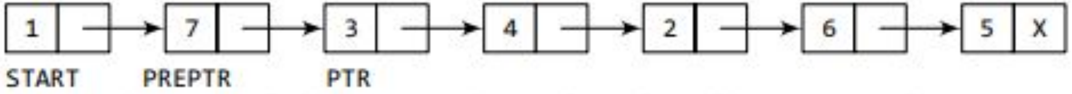
Allocate memory for the new node and initialize its DATA part to 9.



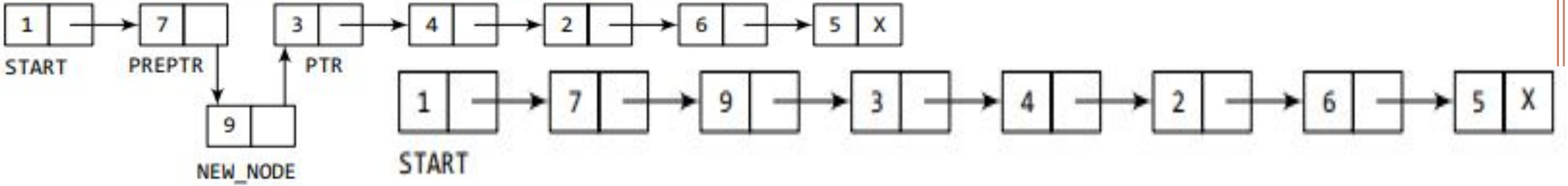
Initialize PREPTR and PTR to the START node.



Move PTR and PREPTR until the DATA part of PTR = value of the node before which insertion has to be done. PREPTR will always point to the node just before PTR.



Insert the new node in between the nodes pointed by PREPTR and PTR.



Create a new node and Insert it before a node

Case 3: When the linked list is already created and a new node is to be created and to be inserted before a given node

- Create a node:
  - Allocate the memory of a node: **new_node=(struct node*)malloc(sizeof(struct node));**
  - Assign the data to the node: **new_node->data = num;**
- Get the node after which the newly created node is to be inserted: **val**
- Create two pointers ptr and preptr and initialize them as start **ptr=start; preptr=ptr;**
- Until ptr->data does not points to the val:

```
while(ptr->data!=val)
    preptr=ptr;
    ptr=ptr->next;
```

- Insert the new_node using preptr pointer:

```
preptr->next=new_node;
new_node->next=ptr;
```

Inserting a New Node before & after a given node

Inserting a New Node before a given node vs Inserting a New Node after a given node

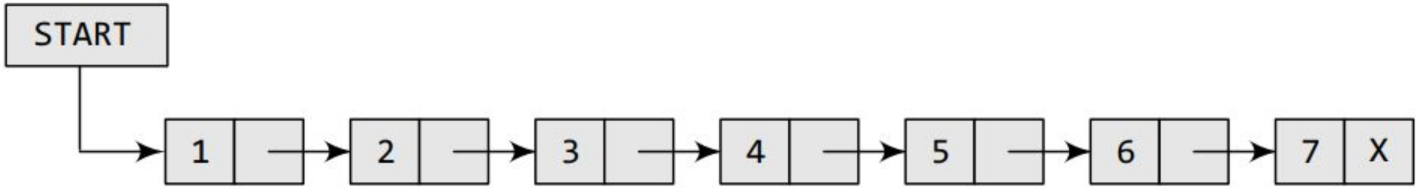
```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 12
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL->NEXT
Step 4: SET NEW_NODE->DATA = VAL
Step 5: SET PTR = START
Step 6: SET PREPTR = PTR
Step 7: Repeat Steps 8 and 9 while PTR->DATA != NUM
Step 8:     SET PREPTR = PTR
Step 9:     SET PTR = PTR->NEXT
        [END OF LOOP]
Step 10: PREPTR->NEXT = NEW_NODE
Step 11: SET NEW_NODE->NEXT = PTR
Step 12: EXIT
```

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 12
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL->NEXT
Step 4: SET NEW_NODE->DATA = VAL
Step 5: SET PTR = START
Step 6: SET PREPTR = PTR
Step 7: Repeat Steps 8 and 9 while PREPTR->DATA
        != NUM
Step 8:     SET PREPTR = PTR
Step 9:     SET PTR = PTR->NEXT
        [END OF LOOP]
Step 10: PREPTR->NEXT = NEW_NODE
Step 11: SET NEW_NODE->NEXT = PTR
Step 12: EXIT
```

Traversing a Linked List

- It is accessing the nodes of the list in order to perform some processing

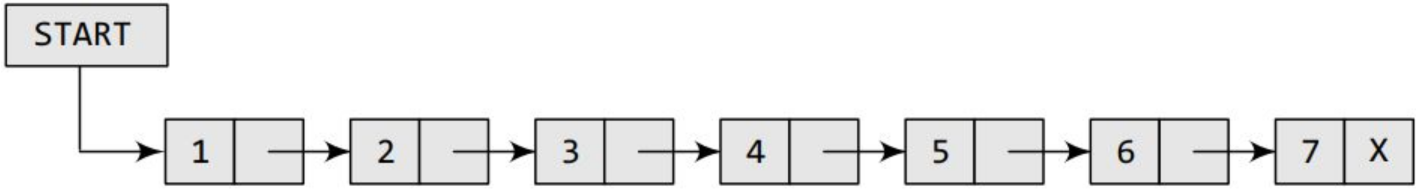
```
Step 1: [INITIALIZE] SET PTR = START
Step 2: Repeat Steps 3 and 4 while PTR != NULL
Step 3:          Apply Process to PTR -> DATA
Step 4:          SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 5: EXIT
```



Example: Algorithm to print the number of nodes in a linked list

- It is accessing the nodes of the list in order to perform some processing

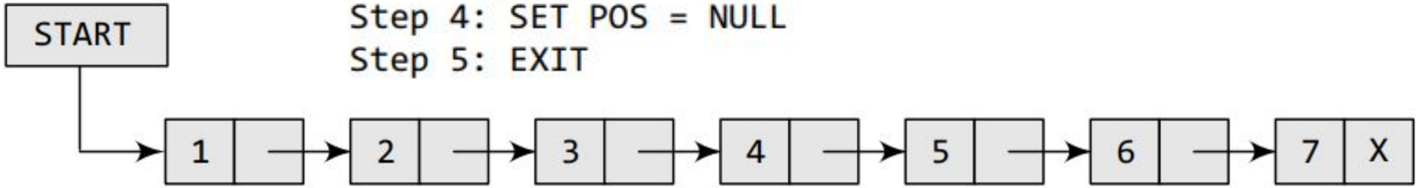
```
Step 1: [INITIALIZE] SET COUNT = 0
Step 2: [INITIALIZE] SET PTR = START
Step 3: Repeat Steps 4 and 5 while PTR != NULL
Step 4:             SET COUNT = COUNT + 1
Step 5:             SET PTR = PTR -> NEXT
         [END OF LOOP]
Step 6: Write COUNT
Step 7: EXIT
```



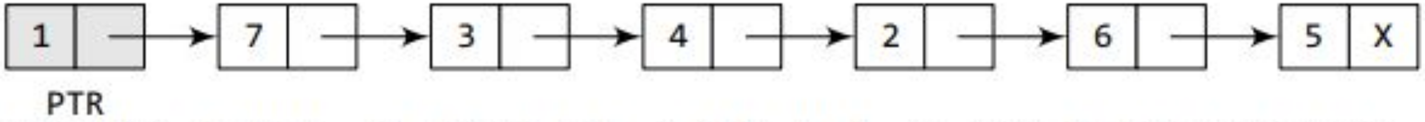
Searching for a Value in a Linked List

- Searching a linked list means to find a particular element in the linked list
- Searching whether a given value is present in the information part of the node or not.
- If it is present, the algorithm returns the address of the node that contains the value.

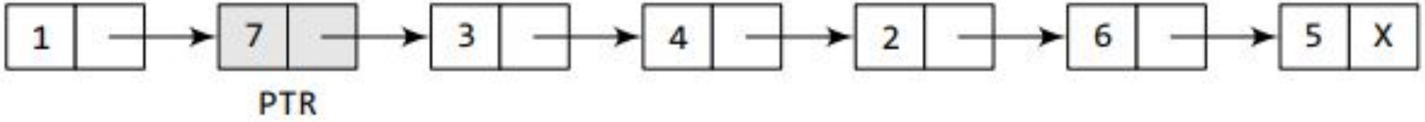
```
Step 1: [INITIALIZE] SET PTR = START
Step 2: Repeat Step 3 while PTR != NULL
Step 3:       IF VAL = PTR -> DATA
                    SET POS = PTR
                    Go To Step 5
              ELSE
                    SET PTR = PTR -> NEXT
              [END OF IF]
        [END OF LOOP]
Step 4: SET POS = NULL
Step 5: EXIT
```



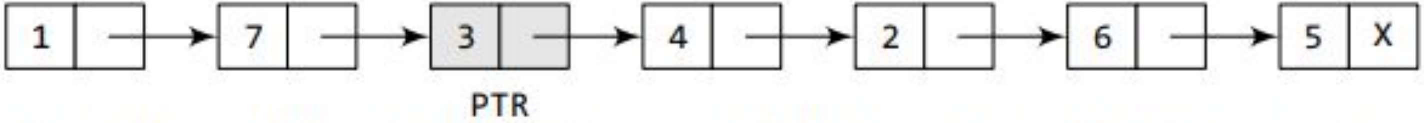
Search for val=4



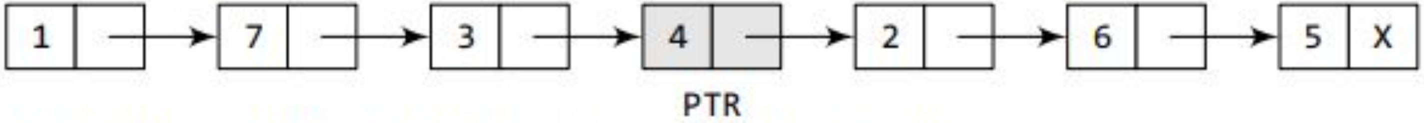
Here PTR -> DATA = 1. Since PTR -> DATA != 4, we move to the next node.



Here PTR -> DATA = 7. Since PTR -> DATA != 4, we move to the next node.



Here PTR -> DATA = 3. Since PTR -> DATA != 4, we move to the next node.



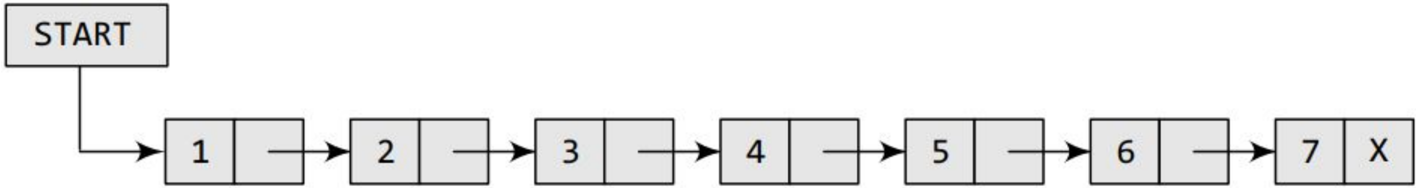
Here PTR -> DATA = 4. Since PTR -> DATA = 4, POS = PTR. POS now stores the address of the node that contains VAL

Deleting a Node from a Linked List

- Case 1: The first node is deleted.
- Case 2: The last node is deleted.
- Case 3: The node after a given node is deleted.

Check Underflow condition, before deletion:

- when START = NULL or when there are no more nodes to delete



Deleting a Node the first node from a Linked List

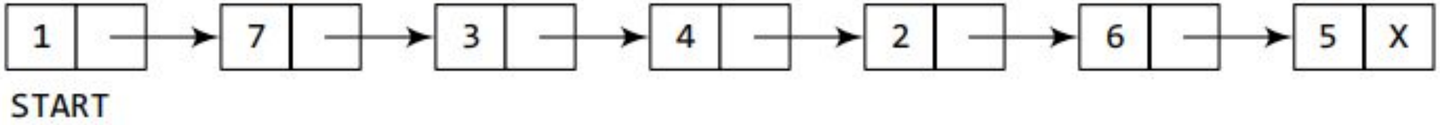
START

Make START to point to the next node in sequence.

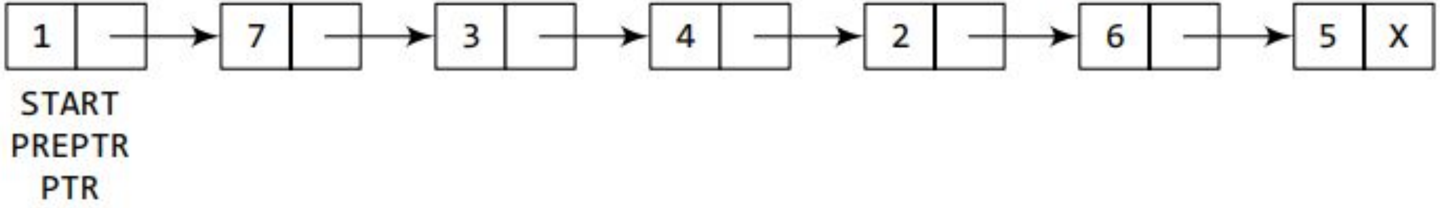
START

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 5
        [END OF IF]
Step 2: SET PTR = START
Step 3: SET START = START -> NEXT
Step 4: FREE PTR
Step 5: EXIT
```

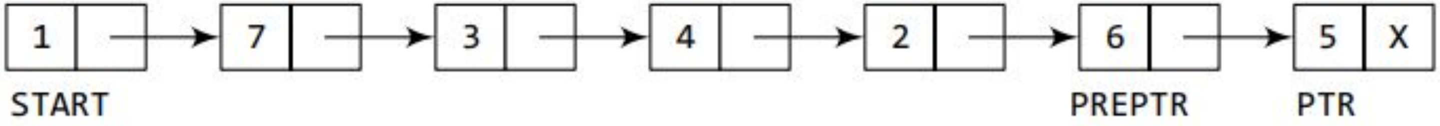
Deleting a Node the last node from a Linked List



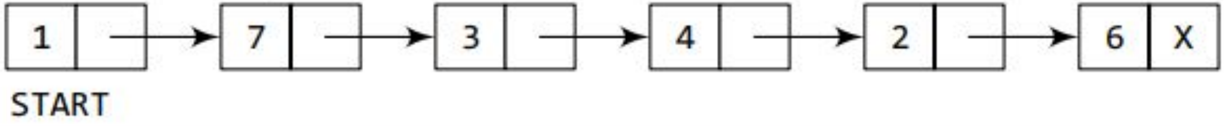
Take pointer variables PTR and PREPTR which initially point to START.



Move PTR and PREPTR such that NEXT part of PTR = NULL. PREPTR always points to the node just before the node pointed by PTR.



Set the NEXT part of PREPTR node to NULL.

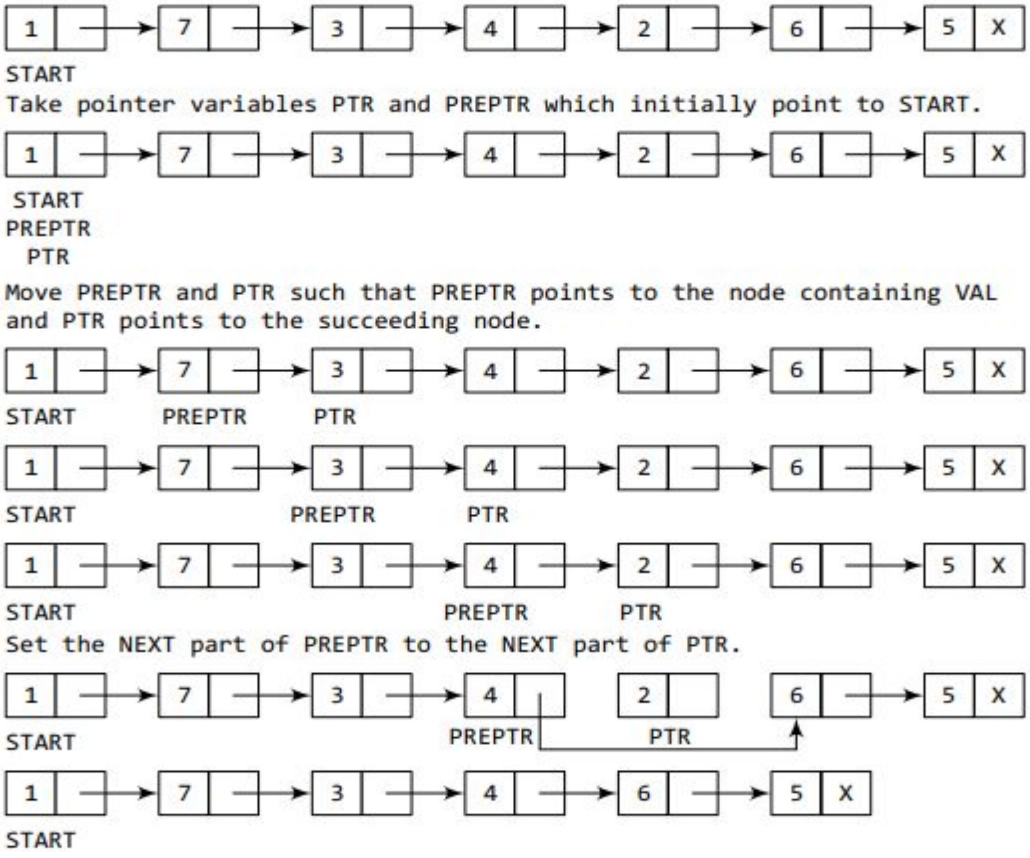


Deleting a Node the last node from a Linked List

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

Deleting a Node after a given node from a Linked List

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 10
        [END OF IF]
Step 2: SET PTR = START
Step 3: SET PREPTR = PTR
Step 4: Repeat Steps 5 and 6 while PREPTR -> DATA != NUM
Step 5:     SET PREPTR = PTR
Step 6:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 7: SET TEMP = PTR
Step 8: SET PREPTR -> NEXT = PTR -> NEXT
Step 9: FREE TEMP
Step 10: EXIT
```



Deleting a Node after a given node from a Linked List


```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 10
        [END OF IF]
Step 2: SET PTR = START
Step 3: SET PREPTR = PTR
Step 4: Repeat Steps 5 and 6 while PREPTR -> DATA != NUM
Step 5:     SET PREPTR = PTR
Step 6:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 7: SET TEMP = PTR
Step 8: SET PREPTR -> NEXT = PTR -> NEXT
Step 9: FREE TEMP
Step 10: EXIT
```

Write a program to create a linked list and perform insertions and deletions of all cases. Write functions to sort and finally delete the entire list at once

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

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

struct node *start = NULL;
struct node *create_ll(struct node *);
struct node *display(struct node *);
struct node *insert_beg(struct node *);
struct node *insert_end(struct node *);
struct node *insert_before(struct node *);
struct node *insert_after(struct node *);
struct node *delete_beg(struct node *);
struct node *delete_end(struct node *);
struct node *delete_node(struct node *);
struct node *delete_after(struct node *);
struct node *delete_list(struct node *);
struct node *sort_list(struct node *);
```

```
int main(int argc, char *argv[]) {
int option;
do{
 printf("\n\n *****MAIN MENU *****");
 printf("\n 1: Create a list");
 printf("\n 2: Display the list");
 printf("\n 3: Add a node at the beginning");
 printf("\n 4: Add a node at the end");
 printf("\n 5: Add a node before a given node");
 printf("\n 6: Add a node after a given node");
 printf("\n 7: Delete a node from the beginning");
printf("\n 8: Delete a node from the end");
 printf("\n 9: Delete a given node");
 printf("\n 10: Delete a node after a given node");
 printf("\n 11: Delete the entire list");
 printf("\n 12: Sort the list");
 printf("\n 13: EXIT");
 printf("\n\n Enter your option : ");
 scanf("%d", &option);
switch(option)  {
 case 1: start = create_ll(start);
 printf("\n LINKED LIST CREATED");  break;
 case 2: start = display(start);  break;
 case 3: start = insert_beg(start);  break;
 case 4: start = insert_end(start);  break;
 case 5: start = insert_before(start);  break;
 case 6: start = insert_after(start);  break;
 case 7: start = delete_beg(start);  break;
 case 8: start = delete_end(start);  break;
 case 9: start = delete_node(start);  break;
 case 10: start = delete_after(start);  break;
 case 11:
 start = delete_list(start);
 printf("\n LINKED LIST DELETED");
 break;
 case 12: start = sort_list(start); break;
 } } while(option !=13);
getch();
return 0;
}
```

```
struct node *create_ll (struct node *start){
struct node *new_node, *ptr;
int num;
printf(“\n Enter -1 to end”);
printf(“\n Enter the data : “);
scanf(“%d”, &num);
while(num!=-1){
 new_node = (struct node*)malloc(sizeof(struct node));
 new_node -> data=num;
 if(start==NULL) {
 new_node -> next = NULL;
 start = new_node;
 }
else {
 ptr=start;
while(ptr->next!=NULL)
 ptr=ptr->next;
 ptr->next = new_node;
 new_node->next=NULL;
 }
 printf(“\n Enter the data : “);
 scanf(“%d”, &num);
}
return start;
}
```

```
struct node *display(struct node *start)
{
struct node *ptr;
ptr = start;
while(ptr != NULL)
{
 printf("\t %d", ptr -> data);
 ptr = ptr -> next;
}
return start;
}
```

```
struct node *insert_beg(struct node *start)
{
struct node *new_node;
int num;
printf("\n Enter the data : ");
scanf("%d", &num);
new_node = (struct node *)malloc(sizeof(struct node));
new_node -> data = num;
new_node -> next = start;
start = new_node;
return start;
}
```

```c
struct node *insert_end(struct node *start)
{
struct node *ptr, *new_node;
int num;
printf(“\n Enter the data : “);
scanf(“%d”, &num);
new_node = (struct node *)malloc(sizeof(struct node));
new_node -> data = num;
new_node -> next = NULL;
ptr = start;
while(ptr -> next != NULL)
ptr = ptr -> next;
ptr -> next = new_node;
return start;
}
```

```c
struct node *insert_before(struct node *start){
struct node *new_node, *ptr, *preptr;
int num, val;
printf(“\n Enter the data : “);
scanf(“%d”, &num);
printf(“\n Enter the value before which the data has to be
inserted : “);
scanf(“%d”, &val);
new_node = (struct node *)malloc(sizeof(struct node));
new_node -> data = num;
ptr = start;
while(ptr -> data != val) {
preptr = ptr;
 ptr = ptr -> next;
}
preptr -> next = new_node;
new_node -> next = ptr;
return start;
}
```

```c
struct node *insert_after(struct node *start){
struct node *new_node, *ptr, *preptr;
int num, val;
printf("\n Enter the data : ");
scanf("%d", &num);
printf("\n Enter the value after which the data has to be
inserted : ");
scanf("%d", &val);
new_node = (struct node *)malloc(sizeof(struct node));
new_node -> data = num;
ptr = start;
preptr = ptr;
while(preptr -> data != val){
 preptr = ptr;
 ptr = ptr -> next;
}
preptr -> next=new_node;
new_node -> next = ptr;
return start;
}
```

```c
struct node *delete_beg(struct node *start){
struct node *ptr;
ptr = start;
start = start -> next;
free(ptr);
return start;
}

struct node *delete_end(struct node *start){
struct node *ptr, *preptr;
ptr = start;
while(ptr -> next != NULL){
 preptr = ptr;
 ptr = ptr -> next;
}
preptr -> next = NULL;
free(ptr);
return start;
}
```

```
struct node *delete_node(struct node *start){
struct node *ptr, *preptr;
int val;
printf("\n Enter the value of the node which has to be
deleted : ");
scanf("%d", &val);
ptr = start;
if(ptr -> data == val){
 start = delete_beg(start);
 return start;
}
else{
while(ptr -> data != val) {
 preptr = ptr;
 ptr = ptr -> next;
 }
 preptr -> next = ptr -> next;
 free(ptr);
 return start;
} }
```

```
struct node *delete_list(struct node *start)
{
        struct node *ptr;
if(start!=NULL){
 ptr=start;
 while(ptr != NULL)
 {
 printf("\n %d is to be deleted next", ptr -> data);
 start = delete_beg(ptr);
ptr = start;
 }
}
return start;
}
```

```
struct node *sort_list(struct node *start) {
struct node *ptr1, *ptr2;
int temp;
ptr1 = start;
while(ptr1 -> next != NULL) {
 ptr2 = ptr1 -> next;
 while(ptr2 != NULL)  {
 if(ptr1 -> data > ptr2 -> data)  {
 temp = ptr1 -> data;
 ptr1 -> data = ptr2 -> data;
 ptr2 -> data = temp;
 }
 ptr2 = ptr2 -> next;
 }
 ptr1 = ptr1 -> next;
}
return start; // Had to be added
}
```

```
struct node *delete_list(struct node *start)
{
           struct node *ptr; // Lines 252-254 were modified
from original code to fix
unresposiveness in output window
if(start!=NULL){
 ptr=start;
 while(ptr != NULL)
 {
 printf(“\n %d is to be deleted next”, ptr -> data);
 start = delete_beg(ptr);
ptr = start;
 }
}
return start;
}
```

CIRCULAR LINKED LISTs