UNIT- II

Linked Structure: List representation, Polish notations, operations on linked list –get node and free node operation, implementing the list operation. inserting into an ordered linked list, deleting, circular linked list, doubly linked list, implementation of stack and queues using linked list.

UNIT – II

Chapter ▶ 3

Linked List

3.1 Introduction

A List can be defined as an ordered collection of data. An array is a list that can be randomly accessed using an index. Arrays are static data structure and disadvantages of using array to store data is that, arrays can not be extended and reduced to fit the data set. Therefore it is possible that we may allocate too much or too little space demanding on our speculation of the size of data set. It is possible to resize the array when the array is full. But the operation requires, claiming new memory for the array and copying elements from the old array to the new array and destroying the old array. These are all expensive operations based on the size of array. Array are also expensive to maintain to new insertion and deletions from the array

Another data structure called LINKED LIST that addresses some of the limitation of Arrays.The decision to use an array or linked list to store data is depend on the type of application.

A linked list is linear data structure which is collection of objects linked together by references from an object to another object.By convention, objects are called nodes. Basic linked list or singly linked list consists of one or more nodes where each node contain one or more data fields and reference to next nodes. Last node contain a null reference which indicate end of list.Unlike arrays where memory is allocated as continuous block of memory,memory required for each node allocated as per need(memory allotted dynamically).

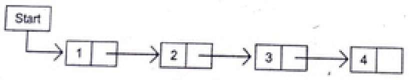


Figure 3.1

The entry node of Linked List is called the head, whereas ,last node termed as tail.If List is empty then head has the value null.Unlike arrays, nodes cannot accessed by index.One must begin from the head and traverse the list to access the node of the List.

3.1 .1 Basic Terminologies

Node: Linked List ,is a linear collection of data elements.The data elements are called as nodes.

DATA: Each node of linked list contains one or more data fields ,called DATA.

NEXT: Each node of linked list contains a pointer to the next node, called NEXT.

PREPTR: In case of Doubly Linked list. A more pointer which points the previous node , called PREPTR.

START: Linked List contain a pointer variable START that store the address of the first node in the list.

NULL: A type of pointer which denotes the end of lists.NULL can be used in another way like "nothing". Circular linked list doesnot have this.

AVAIL: Free pool(which is a linked list of all free memory cells), a pointer variable AVAIL which store the address of the first free space .Before insert a new node in existing linked list , we first check the Avail list , either memory for new node is available or not.

3.2 Advantages of Linked List

(1) They are dynamic in nature which allocates the memory when required.
(2) Insertion of new node and deletion of existing node is easily implemented.
(3) Though Linked List, stacks and queue can be easily implemented.
(4) Linked List reduces the access time.

3.3 Representation of Linked List in Memory

There are two ways to represent a linked list in memory
(1) Static representation using Array.
(2) Dynamic representation using free pool of storage.

Static Representation

A static representation of single linked list maintain two array: one array for data and other for list.

Two parallel array of equal size are allocated which should be sufficient to store the entire linked list. Nevertheless ,this contradicts the idea of linked list (this is non-contiguous location of element).In some programming language like ALGOL,FORTAN,BASIC etc.... such a representation is the only representation to manage linked list.

Dynamic Representation

The efficient way of representing a linked list is using of free pool of storage.In this method, there is a memory bank(a collection of free memory space)and a memory manager(a program,in fact).During the creation of linked list ,when a new node is required , a request placed to memory manager; memory manager will then search memory bank for the block requested and if found grants a desired block to the caller. Again, there is also another program called garbage collector, it active whenever node is no more in use; it return the unused node to memory bank.It is noted that memory bank is also a list of memory space that is available to programmar, Such memory management is called as dynamic memory management.

3.4 Memory Allocation and De-allocation for a Linked List

A free list is a data structure used in dynamic memory allocation. In other words, free list the free pool of memory of unsued space of memory which can be allocated to the new node which has to be insert or created. A pointer variable AVAIL which store the address of the first free space.

De- allocation means deleting a node from the exist linked list. When we delete a particular node from an existing list or delete the entire list , the space occupied by it must be given back to pool so that the memory can be reused by some other program that needs memory space.The whole task is done by operating system. The operating system scans through all the memory cell that are being used by some program. Then it collects the cell 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 as garbage collection.

3.5 Singly Linked List

Singly Linked List is basic linked data structure.Nodes in linked list are linked together using the next field,,which store the address of the next node in the next field of previous node i.e. each node of the lists refers to its successor and the last node contain a null reference.Traversal allow only in one way and there is no going back. Looking at Figure 3.1.

3.5.1 Traversing a Linked List

Traversing of a linked list means accessing the nodes of the lists in order to perform some processing on them.Linked list contain a pointer START which stores the address of first node of the list.End of list has a NULL in the next field of the last node . For traversing the linked list , we have to maintain another pointer variable PTR which points to the node that is currently being accessed.

Algorithm for traversing a linked list

```
Step1:    [INITIALIZE] SET PTR=START
Step2:    Repeat Steps 3 and 4 while PTR !=NULL
Step3:    Apply Process to PTR->DATA
Step4:    SET PTR=PTR->NEXT[END OF LOOP]
Step5:    EXIT
```

In this algorithm, we first initialize PTR with the address of START .so, now ,PTR points to the first node of the linked list .In Step2 , a while loop is executed which is repeated till PTR processes the last node,that is until it encounters NULL. In Step3, we apply the process to the current node ,that is, the node is pointed by PTR.In Step4, we move to the nextnode by making the PTR variable point to the node whose address is stored in the NEXT field.

How you write an algorithm to count the number of nodes in the linked list . To do this, we will traverse each and every node of the list and while traversing every individual node, we will increment the couter by 1. Once we reach NULL, that is, when all nodes of the linked list have been traversed , the final value of the counter will be displayed.So try to write.

3.5.2 Searching for a Value in a Linked List

Searching a linked list means to find a particular element in the linked list.We will check the DATA field of each node that the given value is present in the DATA field or not.If it present, the algorithm return the address of the node that contains the value.

Algorithm to search in a linked list

```
Step1:    [INITIALIZE] SET PTR=START
Step2:    Repeat Step3 while PTR!=NULL
Step3:    IFVAL=PTR->DATASETPOS=PTRGoTOStep5ELSESET PTR=PTR>NEXT[END
          OF IF][END OF LOOP]
Step4:    SET POS=NULL
Step5:    EXIT
```

In step1, we initialize a pointer variable PTR with START that contains the address of the first node .In step2, a while loop is executed which will compare every node'DATA with VAL for which the search is being made.If the search is successful, i.e VAL has been found , then the address of trhat node is store in POS and the control jumps to the last statement of teh algorithm. If search remains unsuccessful, POS is set to NULLwhich indicates the Val is not present in the linked list.

3.5.3 Inserting a New Node in a Linked List

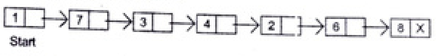
A new node can be inserted in existing linked list in four ways:

(1) The new node is inserted at the beginning of linked list.
(2) The new node is inserted at the end of linked list.
(3) The new node is inserted before a given node.
(4) The new node is inserted after a given node .

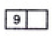
Before going in detail ,lest us first discuss few important terms called OVERFLOW. OVERFLOW is a condition that occurs when AVAIL=NULL or no free memory cell in system.

Inserting a Node at the Begining of a Linked List

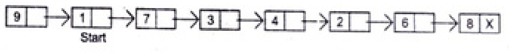
Consider the linked list shown in fig. Suppose we want to add a new node with data 9 and add it the first node of list. Then the following changes will be done.



Allocate memory for the new node and initialise 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



Now make START to paint to the first node of the lest.

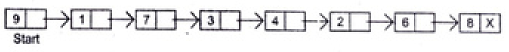


Figure 3.2

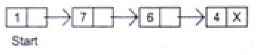
Algorithm to insert a new node at the beginning

```
Step1:    IF AVAIL=NULLWrite"OVERFLOW"GoTO Step7[END OF IF]
Step2:    SET NEW_NODE=AVAIL
Step3:    SET AVAIL=AVAIL->NEXT
Step4:    SET NEW_NODE ->DATA=VAL
Step5:    SET NEW_NODE ->NEXT=START
Step6:    SET START=NEW_NODE
Step7:    EXIT
```

In step 1, we first check ehether the memory is available for the new node. If the free memory has exhausted, then OVERFLOW message is printed. Otherwise, if a free memory cell is available , then we allocate space for the new node . Set its DATA part with the given VAL and the NEXT part is initialise with the address of the first node of the list , which is stored in START. Now, since the new node is added as the first node of the list ,it will now be known as the START node, that is, the start pointer variable will now hold the address of the NEW_NODE . Note the step 2and 3.These steps allocate memory for the new node.

Inserting a Node at the End in a Linked List

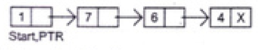
Suppose we want to add a new node with data 9 as the last node . Then following changes will be taken as follow:



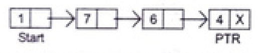
Allocate memory for the new node and initiallise its DATA part to 9 and NEXT part to NULL

Take a pointer variable PTR which points to START.

Take a pointer variable PTR which points to START.



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



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

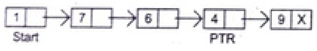


Figure 3.3

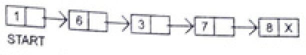
Algorithm to insert a new node at the End

```
Step1:    IF AVAIL=NULLWrite"OVERFLOW"
          GoTO Step10[END OF IF]
Step2:    SET NEW_NODE=AVAIL
Step3:    SET AVAIL=AVAIL->NEXT
Step4:    SET NEW_NODE ->DATA=VAL
Step5:    SET NEW_NODE ->NEXT=NULL
Step6:    SET PTR=START
Step7:          Repeat step8 while PTR->PTr!=NULL
Step8:    SET PTR=PTR->NEXT [END OF LOOP]
Step9:    SET PTR->NEXT=NEW_NODE
Step10:   EXIT
```

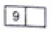
In step 6 , we take pointer variable PTR and initialize it with START .PTR now points to the first node of the *linked list .In while loop , we traverse through the linked list to reach* the last node. Once we reach the last *node , in step 9 we change the NEXT pointer of the last node to store the address of the new node.* Remember *that the NEXT field of tha new node contain NULL, which signifies the end of the linked list.*

Inserting a Node After a Given Node in a Linked List

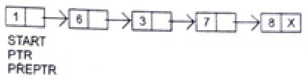
Suppose *we want to add a new node with data 9 after the node containing data 7. Then fol'owing changes will be taken as follow:*



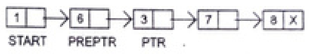
Allocate memory for the new mode and intialise its DATA part to 9

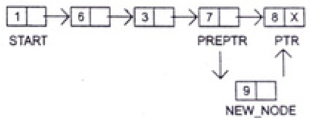


Take two pointer variable PTR and PREPTR and intialise them with START so that START, PTR and PREPTR paint to the first node of list.



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.





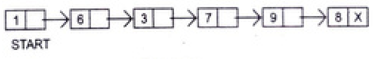


Figure 3.4

Algorithm

```
Step1:    IF AVAIL=NULLWrite"OVERFLOW
          GoTO Step12 [END OF IF]
Step2:    SET NEW_NODE=AVAIL
Step3:    SET AVAIL=AVAIL->NEXT
Step4:    SET NEW_NODE ->DATA=VAL
Step5:    SET PTR=START
Step6:    SET PREPTR=PTr
Step7:          Repeat step8  and 9 while PREPTR->DATA!=NUM
Step8:    SET PREPTR=PTR
Step9:    SET PTR=PTR->NEXT[END OF LOOP]
Step10:   PREPTR->NEXT=NEW_NODE
Step11:   SET NEW_NODE->NEXT=PTR
Step12:   EXIT
```

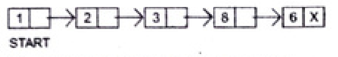
In step 5 , take a pointer PTR and initialise it with START. PTR is now points to the first node of the linked list. Then , we take another pointer PREPTRwhich will be used to store address of the node preceding

PTR. Initially,PREPTR is initialized to PTR. So, now, PTR,PREPTR and START are all pointing to the first node of the linked list.

In while Loop, we traverse through the linked list to reach the node that has its value equal to NUM. We need to reach this node because the new node will be inserted after this node. Once this node , in step 10 and 11 we change the NEXT pointers in such a way that the new node is inserted before the desired node.

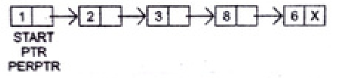
Inserting a Node before a Given node in Linked List

Suppose we want to add a new node with data 9 before the node containing data 8 . Then following changes will be taken as follow:

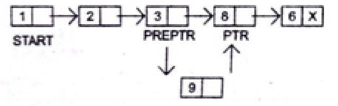


Allocate memory for the new node and initialiase its DATA part to 9

Initialiase 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.



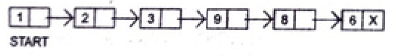


Figure 3.5

Algorithm to insert a new node before a node that has val NUM

```
 Step1:     IF AVAIL=NULLWrite"OVERFLOW"
            GoTO Step12[END OF IF]
 Step2:     SET NEW_NODE=AVAIL
 Step3:     SET AVAIL=AVAIL->NEXT
 Step4:     SET NEW_NODE ->DATA=VAL
 Step5:     SET PTR=START
 Step6:     SET PREPTR=PTR
 Step7:          Repeat step8  and 9 while PREPTR->DATA!=NUM
 Step8:     SET PREPTR=PTR
 Step9:     SET PTR=PTR->NEXT[END OF LOOP]
 Step10:    PREPTR->NEXT=NEW_NODE
 Step11:    SET NEW_NODE->NEXT=PTR
 Step12:    EXIT
```

In step 5 , take a pointer PTR and initialise it with START. PTR is now points to the first node of the linked list. Then , we take another pointer PREPTR and initialize it with PTR So, now, PTR,PREPTR and START are all pointing to the first node of the linked list.

In while Loop, we traverse through the linked list to reach the node that has its value equal to NUM. we need to reach this node because the new node will be inseted before this node. Once this node , in step 10 and 11 we change the NEXT pointers in such a way that the new node is inserted before the desired node.

Deleting a Node from a Linked List

Here we will discussed how node is deleted from existing Linked list .Again we consider three cases:

- The first node is deleted.
- The last node is deleted.
- The node after a given node is deleted.

Underflow is a condition that occurs when we try to delete a node from empty linked list. START=NULL shows this condition.The memory is returned to a free pool so that it can be used to store other programs and data .Whatever the case of deletion , we always change the AVAIL pointer so that it points to the address that has been recently vacated.

Deleting the first Node from a Linked List

Suppose we want to delete the first (starting node) from the linked list . The following changes will be taken as follow;

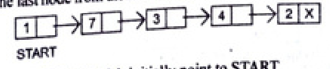
Algorithm to delete the first node

```
Step1:     IF START=NULLWrite UNDERFLOW
           GoTo Step5[END.OF IF]
Step2:     SET PTR=START
Step3:     SET START=START->NEXT
Step4:     FREE PTR
Step5:     EXIT
```

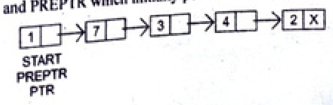
In step1, we check if the linked list is either empty or not. If START=NULL it means UNDERFLOW, and transfer to the EXIT statement. However if there are nodes in linked list , then we use a pointer variable PTR that is set to point to the first node of the list . For this, we initialize PTR with START that stores the address of the first node of the list. In step3 , START is made to point to the next node in sequence and finally the memory occupied by the node in sequence and finally the memory occupied by the node pointer by PTR is freed and returned to the free pool.

Deleting the Last Node from a Linked List

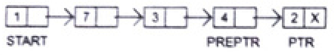
Suppose we want to delete the last node from the linked list , then the following changes will be taken as follow:



Take pointer variable 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.

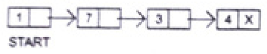


Figure 3.6

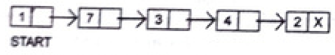
Algorithm to delete the last node

```
Step1:    IF START=NULL Write UNDERFLOW
          GoTo Step8->[END OF IF]
Step2:    SET PTR=START
Step3:          Repeat steps 4 and 5 while PTR->NEXT!=NULL
Step4:    SET PREPTR=PTR
Step5:    SET PTR=PTR->NEXT->[END OF LOOP]
Step6:    SET PREPTR->NEXT=NULL
Step7:    FREE PTR
Step8:    EXIT
```

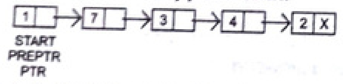
In step 2, we take a pointer variable PTR and initialize it with START . That is, PTR now points to the first node of the linked list. In the while loop, we take another pointer variable PREPTR such that it always points to one node before the PTR. Once we reach the last node and the second last node, we set the NEXT pointer of the second last node to NULL, so that it now become the (new) last node of the linked list.

Deleting the Node After a Given Node in a Linked List

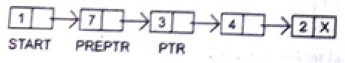
Suppose we want to delete the node that succeeds the node which contains the data 3 . Then the following changes will be taken as following:

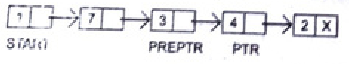


Take pointer variable PTR and PREPTR which initially point to START



Move PREPTR and PTR such that PREPTR points to the node containing VAL and PTR points to the succeding node.





Set the NEXT part of PREPTR to the NEXT part of PTR

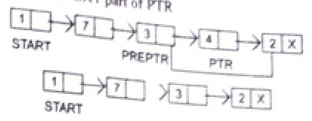


Figure 3.7

Algorithm to delete the node after a given node

```
Step1:    IF START=NULL Write UNDERFLOW
          GoTo Step10->[END OF IF]
Step2:    SET PTR=START
Step3:    SET PREPT=PTR
Step4:          Repeat steps5 and 6 while PREPTR->DATA!=NUM
Step5:    SET PREPTR=PTR
Step6:    SET PTR=PTR->NEXT->[END OF LOOP]
Step7:    SET TEMP=PTR
Step8:    SET PREPTR->NEXT=PTR->NEXT
Step9;    FREE TEMP
Step10:   EXIT
```

In step 2, we take a pointer variable PTR and initialize it with START . That is, PTR now points to the first node of the linked list. In the while loop, we take another pointer variable PREPTR such that it always points to one node before the PTR.Once we reach the node containing VAL and the node succeeding it, we set the next pointer of the node containing VAL to the address contained in next field of the node succeeding it.

3.6 Circular Linked List

A little bit modification in linked list, the last node contains a pointer to the first node of the list. While traversing a circular linked list, we begin at any node and traverse the list in any direction ,forward or backward ,until we reach the same node where we started. Thus, a circular linked list has no begining and no ending .

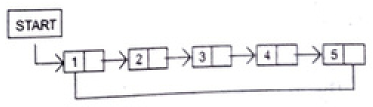


Figure 3.8

The main drawback of circular linked list is its complexity of iteration.Note that there is no NULL value in the NEXT part of any nodes of list.

Inserting a New Node in a circular linked List

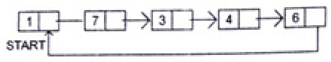
A new node can be inserted in existing linked list in two ways:

(1) The new node is inserted at the beginning of the circular linked list.

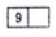
(2) The new node is inserted at the end of the circular linked list.

Inserting a Node at beginning of a Circular Linked List

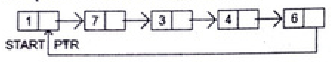
Consider the linked list showin figure. Suppose we want to add a new node with data 9 as the first node of the list



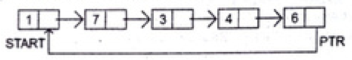
Allocate memory for the new node and initialize its DATA paut to 9.



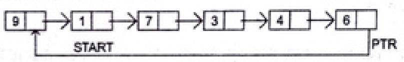
Take a pointer variable PTR that points to the START node of the list



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



Add the new node in between PTR and START



Make START point to the new node.

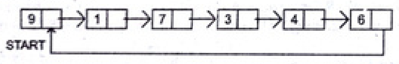


Figure 3.9

Algorithm to insert a new node at the begining

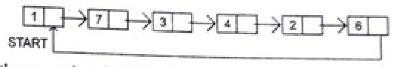
```
Step1:    IF AVAIL=NULL Write "OVERFLOW"
          Go To Step11[END OF IF]
Step2:    SET NEW_NODE =AVAIL
Step3:    SET AVAIL=AVAIL->NEXT
Step4:    SET NEW_NODE->DATA =VAL
Step5:    SET PTR =START
Step6:         Repeat Step7 while PTR->NEXT!=START
Step7:    PTR=PTR->NEXT[END OF IF]
Step8:    SET NEW_NODE ->NEXT=START
Step9:    SET PTR->NEXT=NEW_NODE
Step10:   SET START=NEW_NODE
Step11:   EXIT
```

In step 1, we first check whether memory is available for the new node . If the free memory has exhausted, then OVERFLOW message is printed . Otherwise, if free memory cell is available , then we allocate space for the new node. Set its DATA part with the given VAL and the NEXT part is initialized with the address of the first node of the list, which is stored in START . Now, since the new node is added as the first node of the list, it will now be known a he START node that is, the ATART pointer variable will now hold the address of the NEW_NODE.

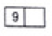
While inserting a node in a circular list , we have use a while loop, to traverse the last node of the list. Because the last node contains a pointer to START, its NEXT field is updated so that after insertion it points to the new node which will be now known as a START.

Inserting a Node at the End of a Circular Linked List

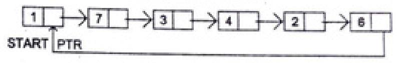
Consider the linked list shown in fig. Suppose we want to add a new node with data 9 as the last node of the list. The following changes will be done.



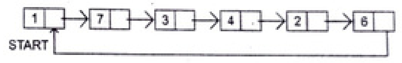
Allocate memory for the new node and initialise its DATA part to 9.



Take a pointer variable PTR which will initially point to START.



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



Add the new node after the node pointed by PTR.

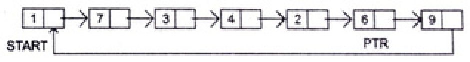


Figure 3.10

Algorithm to insert a new node at the end

```
Step1:    IF AVAIL=NUL Write "OVERFLOW"
          Go To step 10[END OF IF]
Step2:    SET NEW_NODE =AVAIL
Step3:    SET AVAIL=AVAIL->NEXT
Step4:    SET NEW_NODE ->DATA=VAL
Step5:    SET NEW_NODE->NEXT=START
Step6:    SET PTR=START
Step7:         Repeat Step8 while PTR->NEXT!=START
Step8:    SET PTR=PTR->NEXT[END OF LOOP]
Step9:    SET PTR->NEXT=NEW_NODE
Step10:   EXIT
```

In step 6, we take a pointer variable PTR and initialise it with START. That is, PTR now points to the first node of the linked list . In the while loop, we traverse through the linked list to reach the last node . Once we reach the last node , in step 9 we change the NEXT pointer of the last node top store the address of the new node . Remember that the NEXT field of the new node contains the address of the first node which is denoted by START.

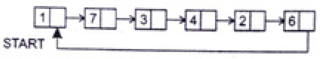
Deleting a Node from a Circular Linked List

A node can be deleted or removed from existing linked list in two ways:

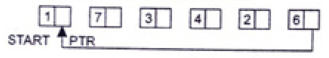
(1) The first node is deleted.
(2) The last node is deleted.

Deleting the First Node from a circular Linked List

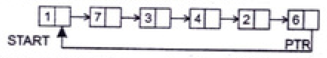
Consider the circular linked list shown in fig . when we want to delete a node from the beginning of the list, then the following changes will be done.



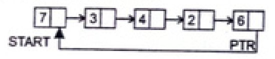
Take avariavle PTR and make it point to the START node of the list



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



The NEXT part of PTR is made to point to the second node of the list and the memory of the first node is freed. The second node becomes the first node of the last.



Figurer 3.11 Deleting the first node from a circular linked list

Algorithm to delete the first node

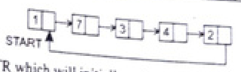
```
Step1:    IF START=NULL Write "Underflow"
          GoTo Step8[END OF IF]
Step2:    SET PTR=START
Step3:        Repeat Step4 while PTR->NEXT!=START
Step4:    SET PTR=PTR->NEXT[END OF LOOP]
Step5:    SET PTR->NEXT=START->NEXT
Step6:    FREE START
Step7:    SET START=PTR->NEXT
Step8:    EXIT
```

In step 1, we check if the linked list exists or not. If START=NULL , then it signifies that there are no *nodes in the list* and the control is transferred to the last statement of the algorithm.

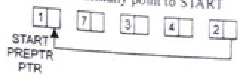
However, if there are nodes in the linked list , then we use pointer variable PTR which will be used to *traversed the list to ultimately reach the last node* . In step 5, we change the next pointer of the last node to point *to the second node of the circular linked list* . In step 6, the memory occupied by the first node is freed. Lastly, *in step 7, the second node now becomes the first node of the list and its address is stored in the variable START.*

Deleting the Last Node from a Circular Linked List

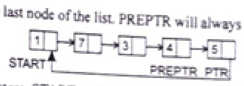
Consider the circular linked list shown in fig . Suppose we want to delete the last node from the linked list, *then the following changes will be done*



Take two pointer PREPTR and PTR which will initially point to START



Move PTR so that it points to the last node of the list. PREPTR will always point to the node preceding PTR.



Make the PREPTR's next part store START node's address and free the space allocated for PTR. Now PREPTR is the last node of the list.

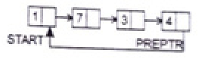


Figure 3.12 Deleting the last node from a circular linked list

Algorithm to delete the Last Node

```
Step1:    IF START=NULLWrite "Underflow"
          GoTo step8[END OF IF]
Step2:    SET PTR=START
Step3:        Repeat step4 and 5 while PTR->NEXT!=START
Step4:    SET PREPTR=PTR
Step5:    SET PTR=PTR->NEXT[END OF LOOP]
Step6:    SET PREPTR->NEXT=START
Step7:    FREE PTR
Step8:    EXIT
```

In step 2 we take a pointer variable PTR and initialize it with START. That is, PTR now points to the first node of the linked list. In the while loop, we take another pointer variable PREPTR such that PREPTR always points to one node before PTR. Once we reach the last node to START, so that it now becomes the (new) last node of the linked list . The memory of the previous last node is freed and returned to the free pool. NOTE: Circular linked list has a wide application in operating system for task maintenance. When we are surfing a Internet , we can use the Back button and the forward button to move the previous pages hat we have already visited.How this is done? The answer is simple .A circular linked list is used to maintain athe sequence of the web pages. Traversing this circular linked list either in forward or backward direction helps to revisit the pages again using Back and Forward buttons. Actually this done using either the circular stack or the circular queue.

Write a program to create a circular linked list. Perform insertion and deletion at the beginning and the end of the list.

```
# include <stdio.h>
# include <conio.h>
# include<malloc.h>
struct node;
```

```
  {
   int data;
   struct node *next;
  };
  struct node *start=NULL;
  struct node *create_cll(struct node);
  struct node *display(struct node *);
  struct node *insert_beg(struct node *);
  struct node *insert_end(struct node *);
  struct node *delete_beg(struct node *);
  struct node *delete_end(struct node *);
  struct node *delete_after(struct node *);
  struct node **delete_list(struct node *);
  int main()
  {
   int option;
   clrscr();
  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 : Delete a node from  the beginning ");
  printf("\n 6 : Delete a node from the beginning ");
  printf("\n 7 : Delete a node after a given node ");
  printf("\n 8 : Delete the entire list ");
  printf("\n 9 : EXIT ");
  printf("\n\n Enter your option : ");
  scanf("%d", &option);
  switch(option)
  {
   case 1: start=create_cll(start);
    printf("\n CIRCULAR LINKED LIST");
    break;
   case 2:
    start=display(start);
    break;
  case 3 :
    start=insert_beg(start)
    break;
  case 4 :
   start=insert_end(start);
   break;
  case 5 :
   start=delete_beg(start);
   break;
 case 6 :
  start= delete_end(start);
  break;
case 7 :
```

```
      start=delete_after(start)
      break;
    case 8 :
      start= delete_list(start);
      printf("\n CIRCULAR LINKED LIST DELETED");
      break;

  }

  }while (option!=9)
  getch();
  return 0;
  }
  struct node *create_cll(struct node *start)
  {

  printf(" \n Enter -1 in end");
  printf(" \n Enter the data");
  scanf(" %d", &num);
  while(num!=1)
  {
   new_node = (struct node*)malloc(size of(struct node));
   new_node-->data=num;
   if(start==NULL)
   {
    new_node->next=new_node;
    start=new_node;
   }
   else
  {
    ptr=start;
     while(ptr=>next != start);
       ptr=ptr->next;
     ptr->next = new_node;
     new_node-> next = start;
   }
   printf(" \n Enter the data");
   scanf("%d", &num);
  }
  return start;
}
struct node *display(struct node *start)
{
  struct node *ptr;
  ptr=start;
  while(ptr-> next!= start)
  {
   printf("\t %d", ptr ->data);
   ptr= ptr->next;
  }
  Printf(" \t %d", ptr-> data);
  return start;
```

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

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

3.7 Doubly Linked List

A complex type of Linked list which contain a pointer to the next as well as the previous node in sequence,called Double Linked List or two-way linked list.Therefore it consists of parts- data , next node pointer, previous node pointer as shown in fig.

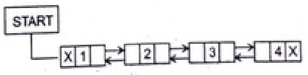


Figure 3.13 Doubly linked list

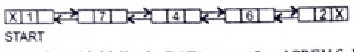
In C, the structure of double linked list can be given as,Struct node
{struct node *prev; int data; struct node *next;};

The Prev pointer of first node and Next pointer of last node contain null.The Prev pointer holds the address of preceding node through which we traverse the list in backward direction.So it is clear that doubly linked list require more spaceas per node than single linked list and more expensive basic operations.The main advantage of using doubly linked list is that it makes searching twice as efficient.

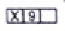
Inserting a New node in Doubly Linked List

In this section we discussed how new node is inserting in an existing doubly linked list.We will take four cases and see how insertion take place.

Inserting a Node in Beginining of Doubly Linked List



Allocate memory for the new node and initialice its DATA part to 9 and PREV field to NULL.



Add new node before the START node. Now the new node becomes the first node of the list.

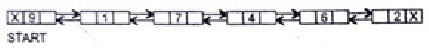


Figure 3.14 Inserting a new node of the begining of Doubly linked list

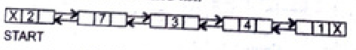
Algorithm to insert a new node at the begining

```
Step1:    IF AVAIL = NULL Write OVERFLOW
          Go to Step 9[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->PREV=NULL
Step 6:   SET NEW_NODE->NEXT=START
Step 7:   SET START->PREV=NEW_NODE
Step 8:   SET START=NEW_NODE
Step 9:   EXIT
```

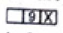
In step1, we check whether the memory is available for new node or not . If the free node is exhausted, then OVERFLOW message is displayed. Otherwise if free node is available, then allocate the memory space to new node. Set the DATA part with the given VAL and the next part is initialised with the address of the first node of the list, which is stored in START. Now , since the new node is added as the first node of the list, it will be known as the START node, that is, the START pointer variable now holds the address of NEW_NODE.

Inserting a Node at the End of a Doubly Linked List

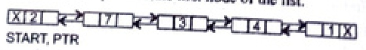
Consider the doubly linked list . Suppose we want to add a new node with data 9 as the last node of the list. Then the following changes will be done in the linked list:



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



Take a pointer variable PTR and make it point to the first node of the list.



Move PTR so that it points to the last node of the list. Add the new node after the node pointed by PTR.

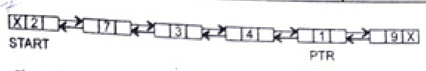


Figure 3.15 Inserting a new node at the end of Doubly linked list

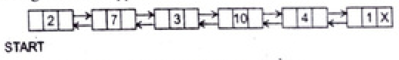
Algorithm

```
Step1:    IF AVAIL=NULL Write OVERFLOW
          Go to Step 11[END OF IF]
Step2:    SET NEW_NODE=AVAIL
Step3:    SET AVAIL=AVAIL->NEXT
Step4:    SET NEW_NODE->DATA=VAL
Step5:    SET NEW_NODE ->NEXT =NULL
Step6:    SET PTR=START
Step7:         Repeat Step 8 while PTR->NEXT !=NULL
Step8:    SET PTR=PTR->NEXT[END OF LOOP]
Step9:    SET PTR->NEXT=NEW_NODE
Step10:   SET NEW_NODE->PREV=PTR
Step11:   EXIT
```

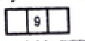
In step 6, we take a pointer variable PTR and initialize it with START. In the while loop, we traverse through the linked list to reach the last node . Once we reach the last node in step 9 , we change the NEXT pointer of the last node to store the address of the new node . Remember that the NEXT field of the new node contains NULL which signifies the end of the linked list . The PREV field of the NEW_NODE will be set so that it points to the node pointed by PTR(now the second last node of the list).

Inserting a Node After a Given Node in a doubly Linked List

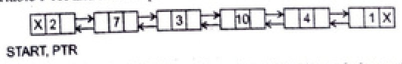
Consider the doubly linked list in fig. Suppose we want to add a new node with value 9 after the node containing 3. Following changes will be happen:



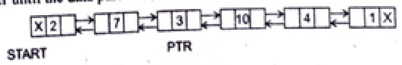
Allocate memory for the new node and intialise its DATA part to 9



Take a pointer variable PTR and make it point to the first node of the list.



Move PTR further until the data part of PTR = value after which the node has to be inserted.



Insert the new node between ptr and node succeding it .

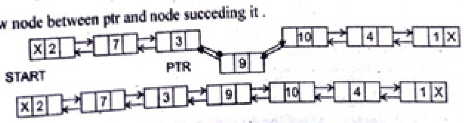
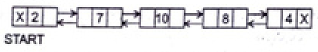


Figure 3.16

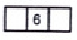
Algorithm to insert a new node after a given node

```
Step1:    IF AVAIL=NULL Write"OVERFL
          Go To Step12 [END OF IF]
Step2:    SET NEW_NODE=AVAIL
Step3:    SET AVAIL=AVAIL->NEXT
Step4:    SET NEW_NODE->DATA=VAL
Step5:    SET PTR=START
Step6:           Repeat step7 while PTR->DATA!=NUM
Step7:    SET PTR=PTR->NEXT[END OF IF]
Step8:    SET NEW_NODE->NEXT->NEXT=PTR->NEXT
Step9:    SET NEW_NODE->PREV=PTR
Step10:   SET PTR->NEXT=NEW_NODE
Step11:   SET PTR->NEXT->PREV=NEW_NODE
Step12:   EXIT
```

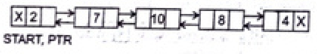
Inserting a Node before a Given Node



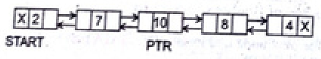
Allocate memory for the new node and initialize its DTA part to 6.



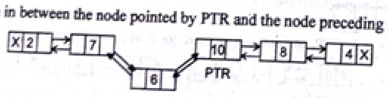
Take a pointer variable PTR and make it point to the first node of the list.



Move PTR further so that it now points to the node whose data is equal to the value before which the node has to be inserted.



Add the new node in between the node pointed by PTR and the node preceding it.



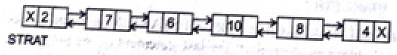


Figure 3.17

```
Step1:    IF AVAIL=NULL Write"OVERFLOW"
          GoTO Step12 [END OF IF]
Step2:    SET NEW_NODE=AVAIL
Step3:    SET AVAIL=AVAIL->NEXT
Step4:    SET NEW_NODE ->DATA=VAL
Step5:    SET PTR=START
Step6:           Repeat step7 while PTR->DATA!=NUM
```

```
step7:    SET PTR=PTR->NEXT[END OF LOOP]
step8:    SET NEW_NODE ->NEXT=PTR
step9:    SET NEW_NODE ->PREV=PTR->PREV
step10:   SET PTR->PREV=NEW _NODE
step11:   SET PTR->PREV->NEXT=NEW_NODE
step12:   EXIT
```

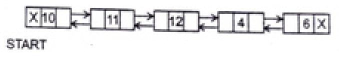
Deleting a Node from a Doubly Linked List

A node can be deleted or removed from existing linked list in four ways:

(1) The first node is deleted.
(2) The last node is deleted.
(3) The node after a given node is deleted .
(4) The node before a given node is deleted.

Deleting the First Node from a Doubly Linked List

Consider the doubly linked list shown in fig. When we want to delete a node from the beginning of the list, then the following changes will be done.



Free the memory occupied by the fist node of the list and make the second node of the list at the START node

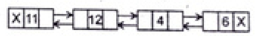


Figure 3.18

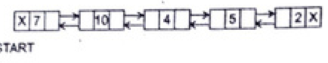
Algorithm to delete the first node of Doubly Linked List

```
Step1:    IF AVAIL=NULL Write"UNDERFLOW"
          Go To Step6[END OF IF]
Step2:    SET PTR=START
Step3:    SET START=START->NEXT
Step4:    SET START->PREV=NULL
Step5:    FREE PTR
Step6:    EXIT
```

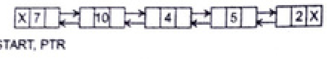
In step 1 we check if the linked list exists or not. If START=NULL then it signifies that there are no nodes in the list and the control is transferred to the last statement of the algorithm. However if there are nodes in linked list , then we use a temporary pointer variable PTR that is set to point to the first node of the list . For this, we initialize PTr with START that stores the address of the first node of the list. In step 3, START is made to point to the next node in sequence and finally the memory occupied by PTr (initially the first node of the list) is freed and returned to free pool.

Deleting the Last Node from a Doubly Linked List

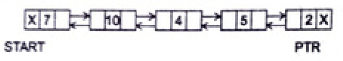
Consider the doubly Linked List shown in fig. Suppose we want to delete the last node from given doubly linked list, then the following changes will be done.



Take a pointer variable PTR that point to the just node of list.



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



Free the space occupied by the node pointed by PTR and store NULL in NEXT field of its preceding node.

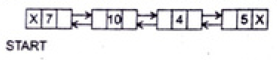


Figure 3.19

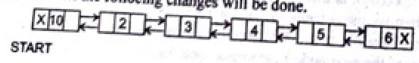
Algorithm to Delete the Last Node

```
Step1:    IF START=NULL Write"UNDERFLOW"
          GoTO Step7[END OF IF]
Step2:    SET PTR=START
Step3:          Repeat Step4 while PTR->NEXT!=NULL
Step4:    SET PTR=PTR->NEXT[END OF IF].
Step5:    SET PTR->PREV->NEXT=NULL
Step6:    FREE PTR
Step7:    EXIT
```

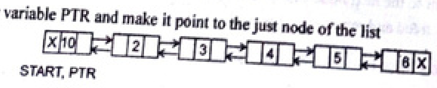
In step 2, we take a pointer variable PTR and initialize it with START. That is PTR now points to the first node of the linked list . The while loop, traverses through the linked list to reach the desired node. Once we reach the node, we can also acess the second laqst node by taking its address from the PREV field of the Last node. To delete the last node, we simply have to set the NEXT field of the second last node to NULL, so that it now becomes the (new) last node of the linked list. The memory of the previous last node is freed and returned to the free pool.

Deleting the Node After a Given Node in Doubly Linked List

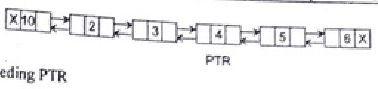
Consider the doubly Linked List shown in fig. Suppose we want to delete the node that succeeds the node which contain data value 4.Then the folloeing changes will be done.



Take a pointer variable PTR and make it point to the just node of the list



Move PTR to further so that its data part is equal to the value after which the node has to be deleted.



Delete the node succeding PTR

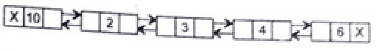


Figure 3.20

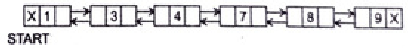
Algorithm to delete a node after a given node

```
Step1:    IF START=NULL Write "UNDERFLOW"
          GoTo Step 9[END OF IF]
Step2:    SET PTR=START
Step3:          Repeat Step4 while PTR->DATA!=NUM
Step4:    SET PTR=PTR->NEXT[END OF IF]
Step5:    SET TEMP=PTR->NEXT
Step6:    SET PTR->NEXT=TEMP->NEXT
Step7:    SET TEMP->NEXT->PREV=PTR
Step8:    FREE TEMP
Step9:    EXIT
```

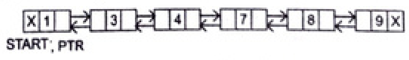
In step 2, we take a pointer variable PTR and initialize it with START. That is PTR now points to the first node of the linked list . The while loop,traverses through the linked list to reach the desired node. Once we reach the node containing VAL, the node succeeding it can be easily accessed by using the address storeed in its NEXT field .The NEXT field of the given node is set to contain the contents in the NEXT field of the succeeding node .Lastly the memory of the node succeeding the given node is freed and returned to the free pool.

Deleting the Node Before a Given Node in a doubly Linked List

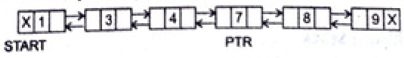
Consider the doubly linked list shown in fig. Suppose we want to delete the node preceding the node with the value 7. Then the following changes will be done as follow:



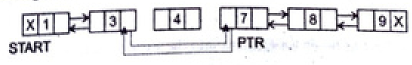
Take a pointer variable PTR that points to the first node of the list.



Move PTR further till its data part is equal to the value before which the node has to be deleted.



Delete the node preceding PTR.



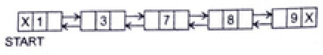


Figure 3.21

Algorithm to delete a node before a given node

```
Step1 :     IF  START=NULL Write "UNDERFLOW"
            Go To Step 9[END OF IF]
Step2:      SET PTR=START
Step3:            Repeat step4 while PTR->DATA
Step4:      SET PTR=PTR->NEXT[END OF LOOP]
Step5:      SET TEMP=PTR->PREV
Step6:      SET TEMP->PREV->NEXT=PTR
Step7:      SET PTR->PREV=TEMP->PREV
Step8:      FREE TEMP
Step9:      EXIT
```

In step 2, we take a pointer variable PTR and initialize it with START. That is PTR now points to the first node of the linked list . The while loop,traverses through the linked list to reach the desired node. Once we reach the node containing VAL, the PREV field of PTR is set to contain the address of the node preceding the node which comes before PTR . The memory of the node preceding PTR is freed and returned to the free pool.

NOTE: We see that we can insert or delete a node in a constant number of operations given only that node's address. This is not possible in Singly Linked List which requires the previous node 's address also to perform the same operation.

3.8 Linked Representation of Stack

We have seen array representation of stack. This technique of creating a stack is easy but have a drawback is that the array must be declared to have some fixed size. In case the stack is a very small one or its maximum size is known in advance , then the array implementation of the stack gives an efficient implementation. But what if the array size cannot be determined in advance, in such situation we use other alternative , i.e. linked representation.

The storage requirement of linked representation of the stack with n elements is O (n), time requirement for the operations is O (1).

As linked list in linked stack, every node has two parts-one that stores data and another that stores the address of the next node. The START pointer of the linked stack is used as TOP. All insertions and deletions are done at the node pointed by TOP.If TOP=NULL , then it indicates that the stack is empty.

The Linked representation of Stack

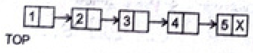


Figure 3.22

Operations on A Linked Stack

Similar to stack , linked stack supports all stack operations, that is, push and pop

Push Operation

The push operation is used to insert an element into the stack .The new element is inserted at the topmost position of the stack.

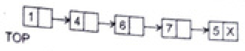


Figure 3.23

To insert an element ,we first check if TOP=NULL.If this is the case then we allocate memory for a new node , store the value in its DATA part and NULL in its next part. Now this new node becomes TOP of stack. However, if TOP!=NULL , then we insert the new node at thew beginning of the linked stack and name this new node as TOP.

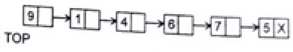


Figure 3.24

Algorithm to insert an element in a linked stack

```
Step1     Allocate memory for the new node and name it as NEW_NODE
Step2     SET NEW_NODE ->DATA=VAL
Step3     IF TOP=NULL
                SET NEW_NODE ->NEXT =TOP
                SET TOP=NEW_NODE
          ELSE
                SET NEW_NODE ->NEXT =TOP
                SET TOP=NEW_NODE
          [END OF IF]
Step4     END
```

Pop Operation

The pop operation is used to delete the topmost element from a stack.Befor deleting the value , we must first check if TOP=NULL, becuse if this is the case , then its means that hte stack is empty and no more deletions can be done. If an attempt is made to delete a value from a stack that is already empty, an UNDERFLOW message is printed.

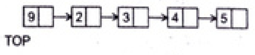


Figure 3.25

In case TOP!=NULL, the we will delete the node pointed by TOP , and make TOP point to the second element of the linked stack.Thus, updated linked stack as follow

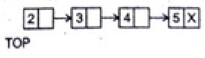
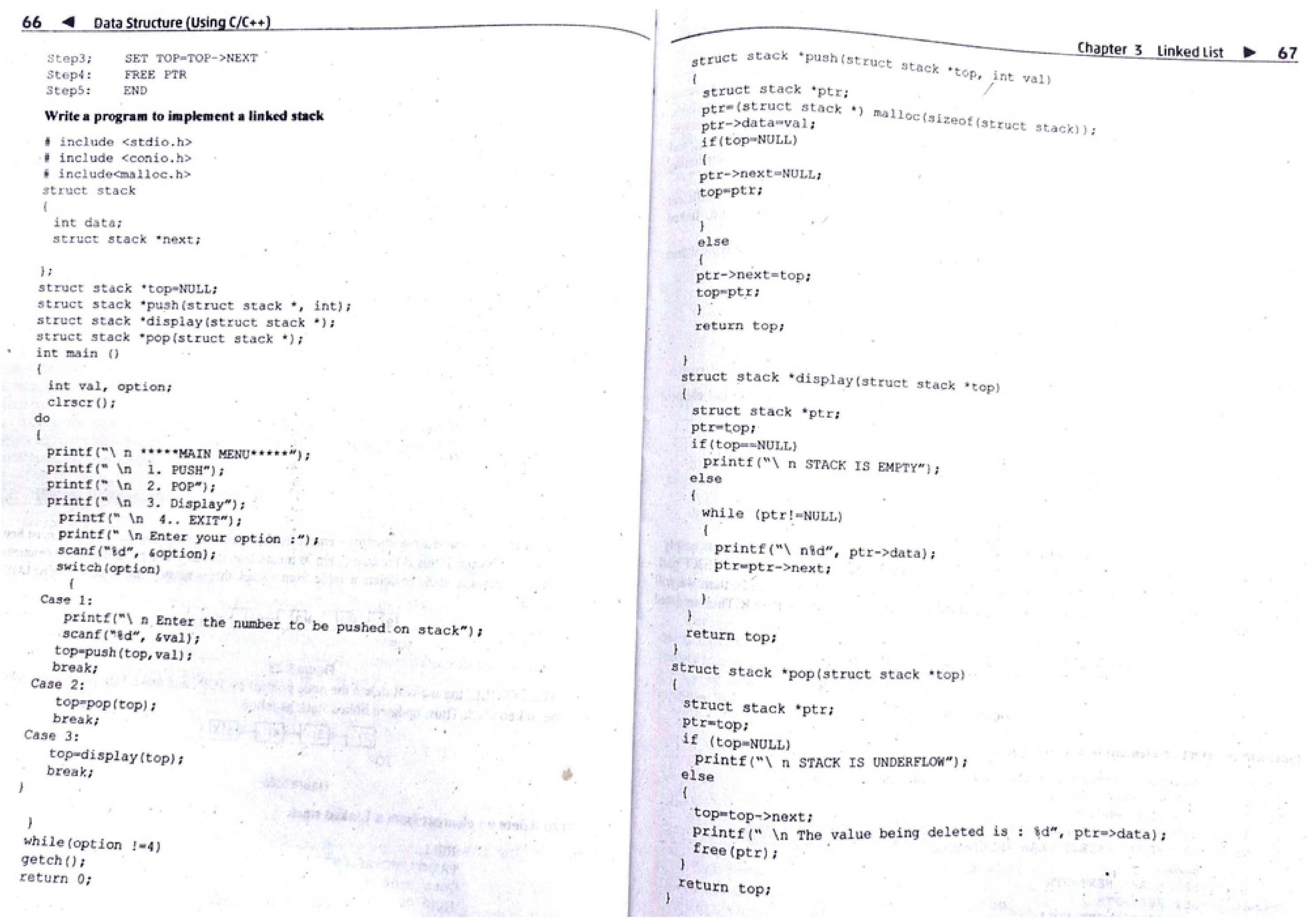


Figure 3.26

Algorithm to delete an element from a Linked stack

```
Step1:    IF TOP=NULL
                PRINT'UNDERFLOW"
                Goto Step 5
                [END OF IF]
Step2:    SET PTR=TOP
```



```
Step3;     SET TOP=TOP->NEXT
Step4:     FREE PTR
Step5:     END
```

Write a program to implement a linked stack

```
 # include <stdio.h>
 # include <conio.h>
 # include<malloc.h>
 struct stack
 {
  int data;
  struct stack *next;

};
struct stack *top=NULL;
struct stack *push(struct stack *, int);
struct stack *display(struct stack *);
struct stack *pop(struct stack *);
int main ()
{
 int val, option;
 clrscr();
do
{
 printf("\ n *****MAIN MENU*****");
 printf(" \n  1. PUSH");
 printf(" \n  2. POP");
 printf(" \n  3. Display");
  printf(" \n  4.. EXIT");
  printf(" \n Enter your option :");
  scanf("%d", &option);
  switch(option)
   {
 Case 1:
   printf("\ n Enter the number to be pushed on stack");
   scanf("%d", &val);
  top=push(top,val);
  break;
Case 2:
  top=pop(top);
  break;
Case 3:
  top=display(top);
  break;
}

 }
while(option !=4)
getch();
return 0;
```

```
 struct stack *push(struct stack *top, int val)
 {
  struct stack *ptr;
  ptr=(struct stack *) malloc(sizeof(struct stack));
  ptr->data=val;
  if(top=NULL)
  {
  ptr->next=NULL;
  top=ptr;

  }
  else
  {
  ptr->next=top;
  top=ptr;
  }
  return top;

}
struct stack *display(struct stack *top)
{
 struct stack *ptr;
 ptr=top;
 if(top==NULL)
  printf("\ n STACK IS EMPTY");
 else
 {
  while (ptr!=NULL)
  {
   printf("\ n%d", ptr->data);
   ptr=ptr->next;

  }
 }
 return top;
}
struct stack *pop(struct stack *top)
{
 struct stack *ptr;
 ptr=top;
 if (top=NULL)
  printf("\ n STACK IS UNDERFLOW");
 else
 {
  top=top->next;
  printf(" \n The value being deleted is : %d", ptr=>data);
  free(ptr);
 }
 return top;
}
```

3.9 Linked Representation of Queue

We have already seen how Queue is created using array. Although this technique of creating a queue is easy but have a drawback of limited size or can say array must to have some fixed size(static nature).If we allocate space for 40 elements in the queue and it hardly use 20-25 locations, then half of the space will be wasted.And in case we allocate less memory locations for a queue that might end up growing largr and large , then a lot of re-allocations will have to be done, thereby creating a lot of overhead and consuming a lot of time.

If array size is known in advance ,then array implementation of the queue gives an efficient implementation.But if the array size cannot be known in advance, the other alternative i.e, linked representation is used.

The storage requirement of linked representation of a queue with n elements is O(n) and the typical time requirement for operations is O(1).

Operations On Linked Queue

Similiar to Queue, Linked queue has two basic operations: Insetion and Deletion.

Insert Operation

The insert operation is used to insert an element into a queue.The new element is added as the last element of the queue.

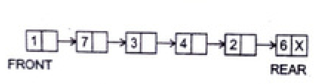


Figure 3.27

To insert an element we first check if FRONT=NULL. If the condition holds , then the queue is empty . So, we allocate memeory for a new node , store the value in its DATA part and NULL in its NEXT part. The New node will then be called both FRONT and REAR. However, if FRONT !=NULL, then we will insert the new node at the rear end of the linked queueand name this new node as REAR.Thus updated queue as follow

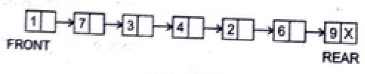


Figure 3.28

Algorithm to insert an element in a linked queue

```
Step1:    Allocate memory for the new node and name it as PTR
Step2:    SET PTR->DATA=NULL
Step3:    IF FRONT =NULL
Step4:    SET FRONT=REAR=PTR
Step5:    SET FRONT -<NEXT=REAR->NEXT=NULL
          ELSE
          SET REAR->NEXT=PTR
Step6:    SET REAR=PTR
Step7:    SET REAR->NEXT=NULL
```

```
          [END OF IF]
Step8:    END
```

Delete Operation

The delete operation is used to delete the element that is first inserted in a queue,i.e,the element whode address is stored in FRONT. However, before deleting the value we must check if, FRONT=NULL because if this case ,then the queue is empty and no more deletions can be done .If the attempt is made to delete a value from a queue that is already empty, an underflow message is printed.

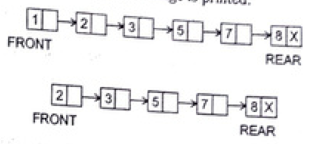


Figure 3.29

Algorithm to delete an element from a linked queue

```
Step1:    IF FRONT=NULL
              Write"UNDERFLOW"
              Goto Step5
              [END OF IF]
Step2:    SET PTR=FRONT
Step3:    SET FRONT=FRONT->NEXT
Step4:    FREE PTR
Step5:    END
```

3.10 Applications of Linked List

Linked list can be used to represent polynomials and the different operations that can be performed on them.
Polynomial Representation

Consider a polynomial $7x^3+6x^2+7x+1$. Every individual term in a polynomial consists of two parts , a coefficient and a power. Here 7, 6, 7 and 1 are the coefficients of the terms that have 3,2,1 and 0 as theiively. rpower respectively.

Every term of a polynomial can be represented as a node of the linked list . Fig. Shows the linked representation of the terms of the above polynomial.

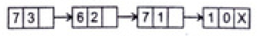


Figure 3.30

■ ■ ■

Very Short Questions

1. Which data structure overcomes the capability of array?
2. Define Linked list.
3. State the list of operations performed over Linked list.
4. What is garbage collection?

5. What is the running time complexity of operation performed over linked list?
6. Write difference between Linear array and linked list
7. Define Multi-linked list.
8. Give the linked representation of the following polynomial:

$$7x^3+6x^2+8x+1$$

Short Questions

1. Write short notes on
   (a) AVAIL (b) Free Pool
   (b) Null Pointer (d) PREV pointer
2. Explain the concept of circular linked list.
3. Explain the concept of Double Linked List.
4. How could you traverse a linked list ? state with an example.
5. Write an algorithm on-
   a). Inserting a new node at beginning of Linked list
   b). Deleting the last node of singly linked list.
   c). Inserting a new node before given node in circular linked list.

Long Questions

1. Make a comparison between Linked list and a linear array. Which one will you prefer to use and when?
2. Why a doubly linked list more useful than a singly linked list?
3. Explain the representing way of linked list in memory.
4. Explain the operations performed over circular linked list.
5. Explain the memory allocation and deallocation concept.