5 Linked List

If the memory is allocated for the variable during the compilation (*i.e.*; *before execution)* of a program, then it is fixed and cannot be changed. For example, an array A[100] is declared with 100 elements, then the allocated memory is fixed and cannot decrease or increase the SIZE of the array if required. So we have to adopt an alternative strategy to allocate memory only when it is required. There is a special data structure called linked list that provides a more flexible storage system and it does not require the use of arrays.

5.1. LINKED LISTS

A linked list is a linear collection of specially designed data elements, called nodes, linked to one another by means of pointers. Each node is divided into two parts: the first part contains the information of the element, and the second part contains the address of the next node in the linked list. Address part of the node is also called linked or next field. Following Fig 5:1 shows a typical example of node.

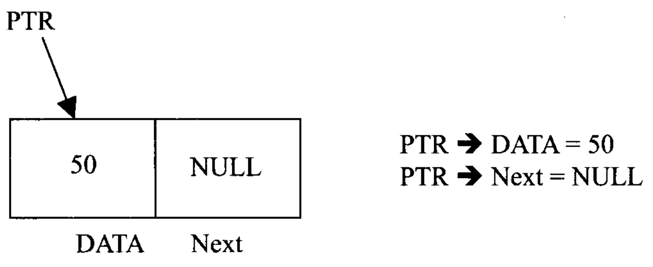

Fig. 5.1. Nodes.

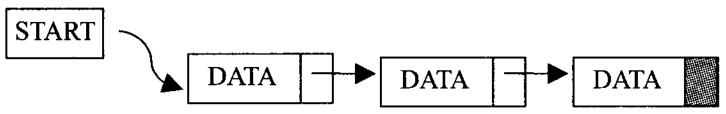

Fig. 5.2. Linked List.

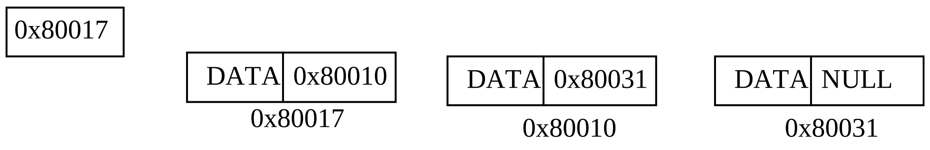

Fig. 5.3. Linked List representation in memory.

Fig. 5.2 shows a schematic diagram of a linked list with 3 nodes. Each node is pictured with two parts. The left part of each node contains the data items and the right part represents the address of the next node; there is an arrow drawn from it to the next node. The next pointer of the last node contains a special value, called the NULL pointer, which does not point to any address of the node. That is NULL pointer indicates the end of the linked list. START pointer will hold the address of the 1st node in the list START = NULL if there is no list (*i.e.*; NULL list or empty list).

5.2. REPRESENTATION OF LINKED LIST

Suppose we want to store a list of integer numbers using linked list. Then it can be schematically represented as

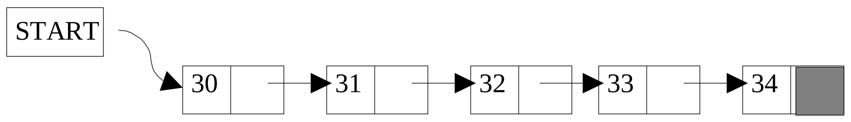

Fig. 5.4. Linked list representation of integers

The linear linked list can be represented in memory with the following declaration.

```
        struct Node
        {
                int DATA; //Instead of 'DATA' we also use 'Info'
                struct Node *Next; //Instead of 'Next' we also use 'Link'
        };
typedef struct Node *NODE;
```

5.3. ADVANTAGES AND DISADVANTAGES

Linked list have many advantages and some of them are:

1. Linked list are dynamic data structure. That is, they can grow or shrink during the execution of a program.
2. Efficient memory utilization: In linked list (or dynamic) representation, memory is not pre-allocated. Memory is allocated whenever it is required. And it is deallocated (or removed) when it is not needed.

3. Insertion and deletion are easier and efficient. Linked list provides flexibility in inserting a data item at a specified position and deletion of a data item from the given position.
4. Many complex applications can be easily carried out with linked list.

Linked list has following disadvantages

1. More memory: to store an integer number, a node with integer data and address field is allocated. That is more memory space is needed.
2. Access to an arbitrary data item is little bit cumbersome and also time consuming.

5.4. OPERATION ON LINKED LIST

The primitive operations performed on the linked list are as follows

1. Creation
2. Insertion
3. Deletion
4. Traversing
5. Searching
6. Concatenation

Creation operation is used to create a linked list. Once a linked list is created with one node, insertion operation can be used to add more elements in a node.

Insertion operation is used to insert a new node at any specified location in the linked list. A new node may be inserted.

(*a*) At the beginning of the linked list

(*b*) At the end of the linked list

(*c*) At any specified position in between in a linked list

Deletion operation is used to delete an item (or node) from the linked list. A node may be deleted from the

(*a*) Beginning of a linked list

(*b*) End of a linked list

(*c*) Specified location of the linked list

Traversing is the process of going through all the nodes from one end to another end of a linked list. In a singly linked list we can visit from left to right, forward traversing, nodes only. But in doubly linked list forward and backward traversing is possible.

Concatenation is the process of appending the second list to the end of the first list. Consider a list A having n nodes and B with m nodes. Then the operation concatenation will place the 1st node of B in the (n+1)th node in A. After concatenation A will contain (n+m) nodes

5.5. TYPES OF LINKED LIST

Basically we can divide the linked list into the following three types in the order in which they (or node) are arranged.

1. Singly linked list
2. Doubly linked list
3. Circular linked list

5.6. SINGLY LINKED LIST

All the nodes in a singly linked list are arranged sequentially by linking with a pointer. A singly linked list can grow or shrink, because it is a dynamic data structure. Following figure explains the different operations on a singly linked list.

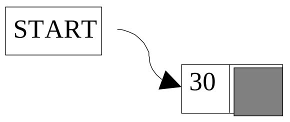

Fig. 5.5. Create a node with DATA(30)

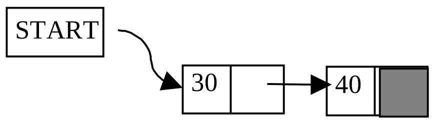

Fig. 5.6. Insert a node with DATA(40) at the end

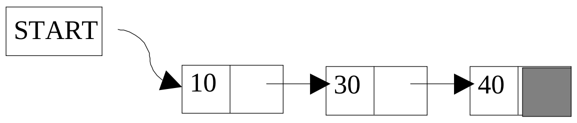

Fig. 5.7. Insert a node with DATA(10) at the beginning

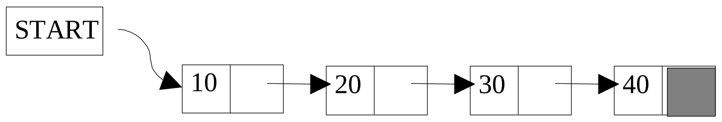

Fig. 5.8. Insert a node with DATA(20) at the 2nd position

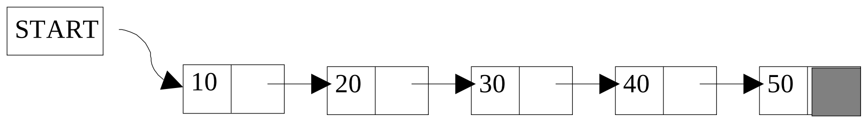

Fig. 5.9. Insert a node with DATA(50) at the end

Output → 10, 20, 30, 40, 50

Fig. 5.10. Traversing the nodes from left to right

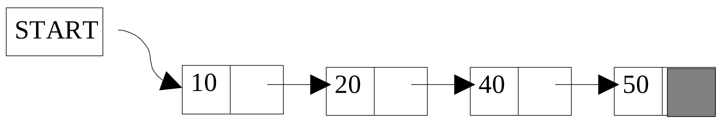

Fig. 5.11. Delete the 3rd node from the list

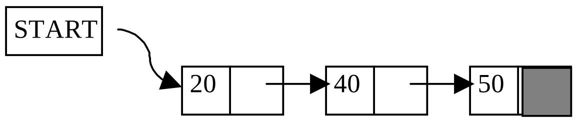

Fig. 5.12. Delete the 1st node

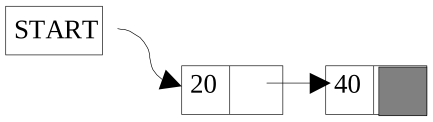

Fig. 5.13. Delete the last node

5.6.1. ALGORITHM FOR INSERTING A NODE

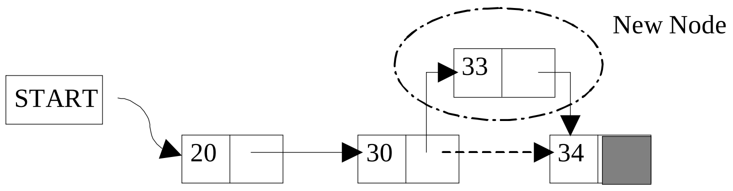

Fig. 5.14. Insertion of New Node

Suppose START is the first position in linked list. Let DATA be the element to be inserted in the new node. POS is the position where the new node is to be inserted. TEMP is a temporary pointer to hold the node address.

Insert a Node at the beginning

```
1. Input DATA to be inserted
2. Create a NewNode
3. NewNode → DATA = DATA
4. If (SATRT equal to NULL)
   (a) NewNode → Link = NULL
5. Else
   (a) NewNode → Link = START
6. START = NewNode
7. Exit
```

Insert a Node at the end

```
1. Input DATA to be inserted
2. Create a NewNode
3. NewNode → DATA = DATA
4. NewNode → Next = NULL
8. If (SATRT equal to NULL)
   (a) START = NewNode
```

```
9. Else
   (a) TEMP = START
   (b) While (TEMP → Next not equal to NULL)
       (i) TEMP = TEMP → Next
10. TEMP → Next = NewNode
11. Exit
```

Insert a Node at any specified position

```
1. Input DATA and POS to be inserted
2. intialise TEMP = START; and j = 0
3. Repeat the step 3 while( k is less than POS)
   (a) TEMP = TEMP è Next
   (b) If (TEMP is equal to NULL)
       (i) Display "Node in the list less than the position"
       (ii) Exit
   (c) k = k + 1
4. Create a New Node
5. NewNode → DATA = DATA
6. NewNode → Next = TEMP → Next
7. TEMP → Next = NewNode
8. Exit
```

5.6.2. ALGORITHM FOR DELETING A NODE

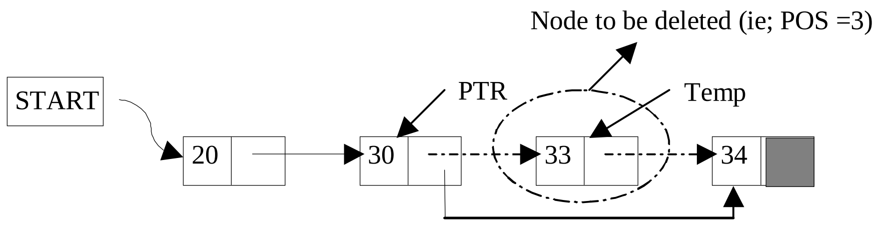

Fig. 5.15. Deletion of a Node.

Suppose START is the first position in linked list. Let DATA be the element to be deleted. TEMP, HOLD is a temporary pointer to hold the node address.

```
1. Input the DATA to be deleted
2. if ((START → DATA) is equal to DATA)
   (a) TEMP = START
   (b) START = START → Next
   (c) Set free the node TEMP, which is deleted
   (d) Exit
```

```
3. HOLD = START
4. while ((HOLD → Next → Next) not equal to NULL))
   (a) if ((HOLD → NEXT → DATA) equal to  DATA)
         (i) TEMP = HOLD → Next
        (ii) HOLD → Next = TEMP → Next
       (iii) Set free the node TEMP, which is deleted
        (iv) Exit
   (b) HOLD = HOLD → Next
5. if ((HOLD → next → DATA) == DATA)
   (a) TEMP = HOLD → Next
   (b) Set free the node TEMP, which is deleted
   (c) HOLD → Next = NULL
   (d) Exit
6. Disply "DATA not found"
7. Exit
```

5.6.3. ALGORITHM FOR SEARCHING A NODE

Suppose START is the address of the first node in the linked list and DATA is the information to be searched. After searching, if the DATA is found, POS will contain the corresponding position in the list.

```
1. Input the DATA to be searched
2. Initialize TEMP = START; POS =1;
3. Repeat the step 4, 5 and 6 until (TEMP is equal to NULL)
4. If (TEMP → DATA is equal to DATA)
   (a) Display "The data is found at POS"
   (b) Exit
5. TEMP = TEMP → Next
6. POS = POS+1
7. If (TEMP is equal to NULL)
   (a) Display "The data is not found in the list"
8. Exit
```

5.6.4. ALGORITHM FOR DISPLAY ALL NODES

Suppose START is the address of the first node in the linked list. Following algorithm will visit all nodes from the START node to the end.

```
1. If (START is equal to NULL)
   (a) Display "The list is Empty"
   (b) Exit
2. Initialize TEMP = START
```

```
3. Repeat the step 4 and 5 until (TEMP == NULL )
4. Display "TEMP → DATA"
5. TEMP = TEMP → Next
6. Exit
```

PROGRAM 5.1

```
//THIS PROGRAM WILL IMPLEMENT ALL THE OPERATIONS
//OF THE SINGLY LINKED LIST
//CODED AND COMPILED IN TURBO C

#include<stdio.h>
#include<conio.h>
#include<malloc.h>
#include<process.h>

//Structure declaration for the node
struct node
{
        int info;
        struct node *link;
    }*start;

    //This function will create a new linked list
    void Create_List(int data)
    {
        struct node *q,*tmp;
        //Dynamic memory is been allocated for a node
        tmp= (struct node*)malloc(sizeof(struct node));
        tmp->info=data;
        tmp->link=NULL;

        if(start==NULL) /*If list is empty*/
            start=tmp;
        else
        {   /*Element inserted at the end*/
            q=start;
            while(q->link!=NULL)
                q=q->link;
            q->link=tmp;
        }
```

```
}/*End of create_list()*/

//This function will add new element at the beginning of the linked list
void AddAtBeg(int data)
{
            struct node *tmp;
            tmp=(struct node*)malloc(sizeof(struct node));
            tmp->info=data;
            tmp->link=start;
            start=tmp;
}/*End of addatbeg()*/

//Following function will add new element at any position
void AddAfter(int data,int pos)
{
            struct node *tmp,*q;
            int i;
            q=start;
            //Finding the position to add new element to the linked list
            for(i=0;i<pos-1;i++)
            {
                  q=q->link;
                  if(q==NULL)
                  {
                        printf ("\n\n There are less than %d elements",pos);
                        getch();
                        return;
                  }
            }/*End of for*/

            tmp=(struct node*)malloc(sizeof (struct node));
            tmp->link=q->link;
            tmp->info=data;
            q->link=tmp;
}/*End of addafter()*/

//Delete any element from the linked list
void Del(int data)
{
            struct node *tmp,*q;
            if (start->info == data)
            {
```

```
                        tmp=start;
                        start=start->link; /*First element deleted*/
                        free(tmp);
                        return;
                }
                q=start;
                while(q->link->link ! = NULL)
                {
                        if(q->link->info == data) /*Element deleted in between*/
                        {
                                tmp=q->link;
                                q->link=tmp->link;
                                free(tmp);
                                return;
                        }
                        q=q->link;
                }/*End of while */
                if(q->link->info==data) /*Last element deleted*/
                {
                        tmp=q->link;
                        free(tmp);
                        q->link=NULL;
                        return;
                }
                printf ("\n\nElement %d not found",data);
                getch();
}/*End of del()*/

//This function will display all the element(s) in the linked list
void Display()
{
                struct node *q;
                if(start == NULL)
                {
                        printf ("\n\nList is empty");
                        return;
                }
                q=start;
                printf("\n\nList is : ");
                while(q!=NULL)
                {
                        printf ("%d ", q->info);
                        q=q->link;
                }
                printf ("\n");
                getch();
}/*End of display() */

//Function to count the number of nodes in the linked list
void Count()
{
                struct node *q=start;
                int cnt=0;
                while(q!=NULL)
                {
                        q=q->link;
                        cnt++;
                }
                printf ("Number of elements are %d\n",cnt);
                getch();
}/*End of count()*/

//This function will reverse the linked list
void Rev()
{
                struct node *p1,*p2,*p3;
                if(start->link==NULL)     /*only one element*/
                        return;
                p1=start;
                p2=p1->link;
                p3=p2->link;
                p1->link=NULL;
                p2->link=p1;
                while(p3!=NULL)
                {
                        p1=p2;
                        p2=p3;
                        p3=p3->link;
                        p2->link=p1;
                }
                start=p2;
}/*End of rev()*/

//Function to search an element from the linked list
void Search(int data)
{
```

```
                struct node *ptr = start;
                int pos = 1;
                //searching for an element in the linked list
                while(ptr!=NULL)
                {
                        if (ptr->info==data)
                        {
                                printf ("\n\nItem %d found at position %d", data, pos);
                                getch();
                                return;
                        }
                        ptr = ptr->link;
                        pos++;
                }
                if (ptr == NULL)
                        printf ("\n\nItem %d not found in list",data);
                getch();
}


void main()
{
                int choice,n,m,position,i;
                start=NULL;
                while(1)
                {
                        clrscr();
                        printf ("1.Create List\n");
                        printf ("2.Add at beginning\n");
                        printf ("3.Add after \n");
                        printf ("4.Delete\n");
                        printf ("5.Display\n");
                        printf ("6.Count\n");
                        printf ("7.Reverse\n");
                        printf ("8.Search\n");
                        printf ("9.Quit\n");
                        printf ("\nEnter your choice:");
                        scanf ("%d",&choice);
                        switch (choice)
                        {
                         case 1:
                                printf ("\n\nHow many nodes you want:");
                                scanf ("%d",&n);
```

```
          for(i = 0;i<n;i++)
          {
                printf ("\nEnter the element:");
                scanf ("%d",&m);
                Create_List(m);
          }
          break;
     case 2:
          printf ("\n\nEnter the element : ");
          scanf ("%d",&m);
          AddAtBeg(m);
          break;
     case 3:
          printf ("\n\nEnter the element:");
          scanf ("%d",&m);
          printf ("\nEnter the position after which this element is inserted:");
          scanf ("%d",&position);
          Add After(m,position);
          break;
     case 4:
          if (start == NULL)
          {
                printf("\n\nList is empty");
                continue;
          }
          printf ("\n\nEnter the element for deletion:");
          scanf ("%d",&m);
          Del(m);
          break;
     case 5:
          Display();
          break;
     case 6:
          Count();
          break;
     case 7:
          Rev();
          break;
     case 8:
          printf("\n\nEnter the element to be searched:");
          scanf ("%d",&m);
```

```
                        Search(m);
                        break;
                    case 9:
                        exit(0);
                    default:
                        printf ("\n\nWrong choice");
                }/*End of switch*/
        }/*End of while*/
}/*End of main()*/
```

PROGRAM 5.2

```
//THIS PROGRAM WILL IMPLEMENT ALL THE OPERATIONS
//OF THE SINGLY LINKED LIST
//CODED AND COMPILED IN TURBO C++


#include<iostream.h>
#include<conio.h>
#include<process.h>


class Linked_List
{
        //Structure declaration for the node
        struct node
        {
                        int info;
                        struct node *link;
        };

                //private structure variable declared
                struct node *start;
            public:
                Linked_List()//Constructor defined
                {
                        start = NULL;
                }
                //public fucntion declared
                void Create_List(int);
```

```
                        void AddAtBeg(int);
                        void AddAfter(int,int);
                        void Delete();
                        void Count();
                        void Search(int);
                        void Display();
                        void Reverse();
};

//This function will create a new linked list of elements
void Linked_List::Create_List(int data)
{
        struct node *q,*tmp;
        //New node is created with new operator
        tmp= (struct node *)new(struct node);
        tmp->info=data;
        tmp->link=NULL;

        if (start==NULL) /*If list is empty */
            start=tmp;
        else
        {       /*Element inserted at the end */
            q=start;
            while(q->link!=NULL)
                q=q->link;
            q-> link=tmp;
        }
}/*End of create_list()*/

//following function will add new element at the beginning
void Linked_List::AddAtBeg(int data)
{
        struct node *tmp;
        tmp=(struct node*)new(struct node);
        tmp->info=data;
        tmp->link=start;
        start=tmp;
}/*End of addatbeg()*/

//This function will add new element at any specified position
void Linked_List::AddAfter(int data,int pos)
{
		struct node *tmp,*q;
		int i;
		q=start;
		//Finding the position in the linked list to insert
		for(i=0;i<pos-1;i++)
		{
			q=q->link;
			if(q==NULL)
			{
				cout<<"\n\nThere are less than "<<pos<<" elements";
				getch();
				return;
			}
		}/*End of for*/

		tmp=(struct node*)new(struct node);
		tmp->link=q->link;
		tmp->info=data;
		q->link=tmp;
}/*End of addafter()*/

//Funtion to delete an element from the list
void Linked_List::Delete()
{
		struct node *tmp,*q;
		int data;
		if(start==NULL)
		{
			cout<<"\n\nList is empty";
			getch();
			return;
		}
		cout<<"\n\nEnter the element for deletion : ";
		cin>>data;

		if(start->info == data)
		{
			tmp=start;
			start=start->link; //First element deleted
			delete(tmp);
			return;
		}
```

```
                q=start;
                while(q->link->link != NULL)
                {
                        if(q->link->info==data) //Element deleted in between
                        {
                                tmp=q->link;
                                q->link=tmp->link;
                                delete(tmp);
                                return;
                        }
                        q=q->link;
                }/*End of while */
                if(q->link->info==data) //Last element deleted
                {
                        tmp=q->link;
                        delete(tmp);
                        q->link=NULL;
                        return;
                }
                cout<<"\n\nElement "<<data<<" not found";
                getch();
}/*End of del()*/

void Linked_List::Display()
{
                struct node *q;
                if(start == NULL)
                {
                        cout<<"\n\nList is empty";
                        return;
                }
                q=start;
                cout<<"\n\nList is : ";
                while(q!=NULL)
                {
                        cout<<q->info;
                        q=q->link;
                }
                cout<<"\n";
                getch();
}/*End of display() */
```

```
void Linked_List::Count()
{
                struct node *q=start;
                int cnt=0;
                while(q!=NULL)
                {
                        q=q->link;
                        cnt++;
                }
                cout<<"Number of elements are \n"<<cnt;
                getch();
}/*End of count() */

void Linked_List::Reverse()
{
                struct node *p1,*p2,*p3;
                if(start->link==NULL)     /*only one element*/
                        return;
                p1=start;
                p2=p1->link;
                p3=p2->link;
                p1->link=NULL;
                p2->link=p1;
                while(p3!=NULL)
                {
                        p1=p2;
                        p2=p3;
                        p3=p3->link;
                        p2->link=p1;
                }
                start=p2;
}/*End of rev()*/

void Linked_List::Search(int data)
{
                struct node *ptr = start;
                int pos = 1;
                while(ptr!=NULL)
                {
                        if(ptr->info==data)
                        {
                                cout<<"\n\nItem "<<data<<" found at position "<<pos;
                                getch();
                                return;
                        }
                        ptr = ptr->link;
                        pos++;
                }
                if(ptr == NULL)
                        cout<<"\n\nItem "<<data<<" not found in list";
                getch();
}


void main()
{
                int choice,n,m,position,i;
                Linked_List po;
                while(1)
                {
                        clrscr();
                        cout<<"1.Create List\n";
                        cout<<"2.Add at begining\n";
                        cout<<"3.Add after \n";
                        cout<<"4.Delete\n";
                        cout<<"5.Display\n";
                        cout<<"6.Count\n";
                        cout<<"7.Reverse\n";
                        cout<<"8.Search\n";
                        cout<<"9.Quit\n";
                        cout<<"\nEnter your choice:";
                        cin>>choice;
                        switch(choice)
                        {
                         case 1:
                                cout<<"\n\nHow many nodes you want:";
                                cin>>n;
                                for(i=0;i<n;i++)
                                {
                                        cout<<"\nEnter the element:";
                                        cin>>m;
                                        po.Create_List(m);
                                }
                                break;
                         case 2:
                                cout<<"\n\nEnter the element:";
                                cin>>m;
                                po.AddAtBeg(m);
                                break;
                         case 3:
                                cout<<"\n\nEnter the element:";
                                cin>>m;
                                cout<<"\nEnter the position after which this element is inserted:";
                                cin>>position;
                                po.AddAfter(m,position);
                                break;
                         case 4:
                                po.Delete();
                                break;
                         case 5:
                                po.Display();
                                break;
                         case 6:
                                po.Count();
                                break;
                         case 7:
                                po.Reverse();
                                break;
                         case 8:
                                cout<<"\n\nEnter the element to be searched:";
                                cin>>m;
                                po.Search(m);
                                break;
                         case 9:
                                exit(0);
                         default:
                                cout<<"\n\nWrong choice";
                        }/*End of switch */
                }/*End of while */
}/*End of main()*/
```

5.7. STACK USING LINKED LIST

In chapter 3, we have discussed what a stack means and its different operations. And we have also discussed the implementation of stack using array, *i.e.*, static memory allocation. Implementation issues of the stack (Last In First Out - LIFO) using linked list is illustrated in following figures.

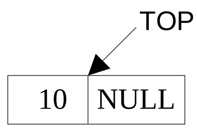

Fig. 5.11. push (10)

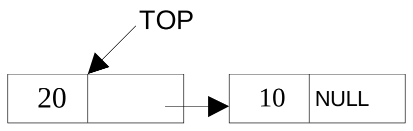

Fig. 5.12. push (20)

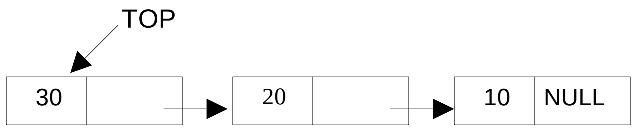

Fig. 5.13. push (30)

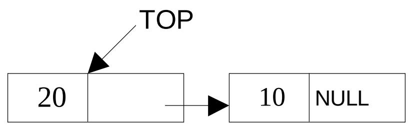

Fig. 5.14. X = pop() (ie; X = 30)

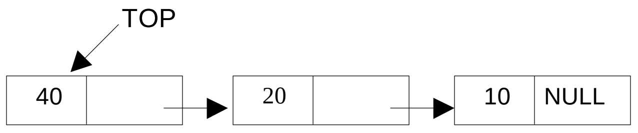

Fig. 5.15. push (40)

5.7.1. ALGORITHM FOR PUSH OPERATION

Suppose TOP is a pointer, which is pointing towards the topmost element of the stack. TOP is NULL when the stack is empty. DATA is the data item to be pushed.

```
1. Input the DATA to be pushed
2. Creat a New Node
3. NewNode → DATA = DATA
4. NewNode → Next = TOP
5. TOP = NewNode
6. Exit
```

5.7.2. ALGORITHM FOR POP OPERATION

Suppose TOP is a pointer, which is pointing towards the topmost element of the stack. TOP is NULL when the stack is empty. TEMP is pointer variable to hold any nodes address. DATA is the information on the node which is just deleted.

```
1. if (TOP is equal to NULL)
   (a) Display "The stack is empty"
2. Else
   (a) TEMP = TOP
   (b) Display "The popped element TOP → DATA"
   (c) TOP = TOP → Next
   (d) TEMP → Next = NULL
   (e) Free the TEMP node
3. Exit
```

PROGRAM 5.3

```
//THIS PROGRAM IS TO DEMONSTRATE THE OPERATIONS
//PERFORMED ON THE STACK INPLEMENTED USING LINKED LIST
//CODED AND COMPILED IN TURBO C
#include<conio.h>
#include<stdio.h>
#include<malloc.h>
#include<process.h>

//Structure is created a node
struct node
{
		int info;
		struct node *link;//A link to the next node
};

//A variable named NODE is been defined for the structure
typedef struct node *NODE;

//This function is to perform the push operation
NODE push(NODE top)
{
		NODE NewNode;
		int pushed_item;
		//A new node is created dynamically
		NewNode = (NODE)malloc(sizeof(struct node));
		printf("\nInput the new value to be pushed on the stack:");
		scanf("%d",&pushed_item);
		NewNode->info=pushed_item;//Data is pushed to the stack
		NewNode->link=top;//Link pointer is set to the next node
		top=NewNode;//Top pointer is set
		return(top);
}/*End of push()*/

//Following function will implement the pop operation
NODE pop(NODE top)
{
		NODE tmp;
		if(top == NULL)//checking whether the stack is empty or not
			printf ("\nStack is empty\n");
		else
		{
	tmp=top;//popping the element
			printf("\nPopped item is %d\n",tmp->info);
			top=top->link;//resetting the top pointer
			tmp->link=NULL
			free(tmp);//freeing the popped node
		}
		return(top);

}/*End of pop()*/

//This is to display the entire element in the stack
void display(NODE top)
{
		if(top==NULL)
			printf("\nStack is empty\n");
		else
		{
			printf("\nStack elements:\n");
			while(top != NULL)
			{
				printf("%d\n",top->info);
				top = top->link;
			}/*End of while */
		}/*End of else*/
}/*End of display()*/
```

```
void main()
{
        char opt;
        int choice;
        NODE Top=NULL;
        do
        {
            clrscr();
            printf("\n1.PUSH\n");
            printf("2.POP\n");
            printf("3.DISPLAY\n");
            printf("4.EXIT\n");
            printf("\nEnter your choice:");
            scanf("%d", &choice);
            switch(choice)
            {
            case 1:
                Top=push(Top);
                break;
            case 2:
                Top=pop(Top);
                break;
            case 3:
                display(Top);
                break;
            case 4:
                exit(1);
            default:
                printf("\nWrong choice\n");
            }/*End of switch*/

            printf ("\n\nDo you want to continue (Y/y) = ");
            fflush(stdin);
            scanf("%c",&opt);
        }while((opt == 'Y') || (opt == 'y'));
}/*End of main() */
```

PROGRAM 5.4

```
//THIS PROGRAM IS TO DEMONSTRATE THE OPERATIONS
//PERFORMED ON THE STACK IMPLEMENTATION
```

```
//USING LINKED LIST
//CODED AND COMPILED IN TURBO C++

#include<conio.h>
#include<iostream.h>
#include<process.h>

//Class is created for the linked list
class Stack_Linked
{
        //Structure is created for the node
        struct node
        {
            int info;
            struct node *link;//A link to the next node
        };

        //A variable top is been declared for the structure
        struct node *top;
        //NODE is defined as the data type of the structure node
        typedef struct node *NODE;

        public:
            //Constructer is defined for the class
            Stack_Linked()
            {
                //top pointer is initialized
                top=NULL;
            }
            //function declarations
            void push();
            void pop();
            void display();
};

//This function is to perform the push operation
void Stack_Linked::push()
{
        NODE NewNode;
        int pushed_item;
        //A new node is created dynamically
        NewNode=(NODE)new(struct node);
        cout<<"\nInput the new value to be pushed on the stack:";
```

```
            cin>>pushed_item;
            NewNode->info=pushed_item;//Data is pushed to the stack
            NewNode->link=top;//Link pointer is set to the next node
            top=NewNode;//Top pointer is set
}/*End of push()*/

//Following function will implement the pop operation
void Stack_Linked::pop()
{
            NODE tmp;
            if(top == NULL)//checking whether the stack is empty or not
                  cout<<"\nStack is empty\n";
            else
            {     tmp=top;//popping the element
                  cout<<"\nPopped item is:"<<tmp->info;
                  top=top->link;//resetting the top pointer
                  tmp->link=NULL;
                  delete(tmp);//freeing the popped node
            }
}/*End of pop()*/

//This is to display all the element in the stack
void Stack_Linked::display()
{
            if(top==NULL)//Checking whether the stack is empty or not
                  cout<<"\nStack is empty\n";
            else
            {
                  NODE ptr=top;
                  cout<<"\nStack elements:\n";
                  while(ptr != NULL)
                  {
                        cout<<"\n"<<ptr->info;
                        ptr = ptr->link;
                  }/*End of while */
            }/*End of else*/
}/*End of display()*/

void main()
{
            char opt;
            int choice;
```

```
        Stack_Linked So;
        do
        {
            clrscr();
            //The menu options are listed below
            cout<<"\n1.PUSH\n";
            cout<<"2.POP\n";
            cout<<"3.DISPLAY\n";
            cout<<"4.EXIT\n";
            cout<<"\nEnter your choice : ";
            cin>>choice;

            switch(choice)
            {
            case 1:
                So.push();//push function is called
                break;
            case 2:
                So.pop();//pop function is called
                break;
            case 3:
                So.display();//display function is called
                break;
            case 4:
                exit(1);
            default:
                cout<<"\nWrong choice\n";
            }/*End of switch */

            cout<<"\n\nDo you want to continue (Y/y) = ";
            cin>>opt;
        }while((opt == 'Y') || (opt == 'y'));
}/*End of main() */
```

5.8. QUEUE USING LINKED LIST

Queue is a First In First Out [FIFO] data structure. In chapter 4, we have discussed about stacks and its different operations. And we have also discussed the implementation of stack using array, ie; static memory allocation. Implementation issues of the stack (Last In First Out - LIFO) using linked list is illustrated in the following figures.

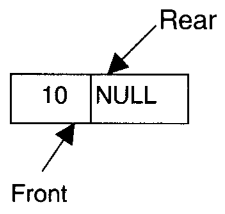

Fig. 5.16. push (10)

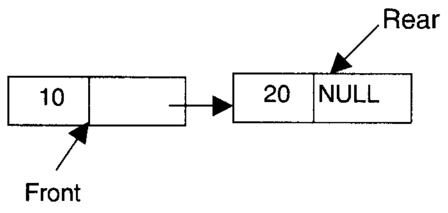

Fig. 5.17. push (20)

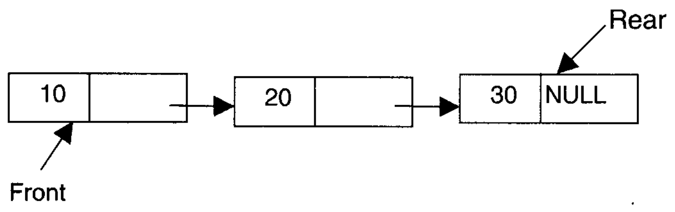

Fig. 5.18. push (30)

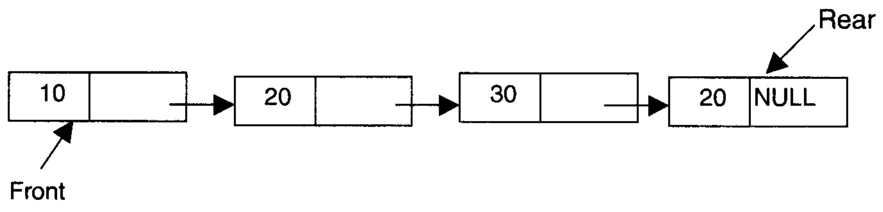

Fig. 5.19. push (40)

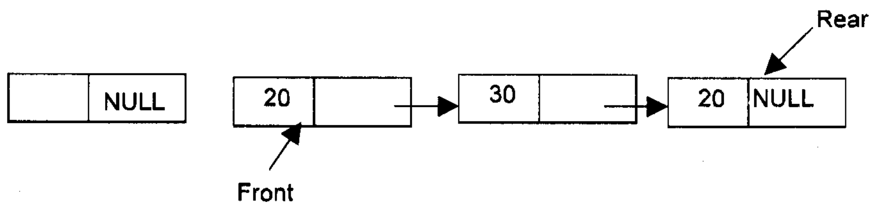

Fig. 5.20. X = pop() (*i.e.*; X = 10)

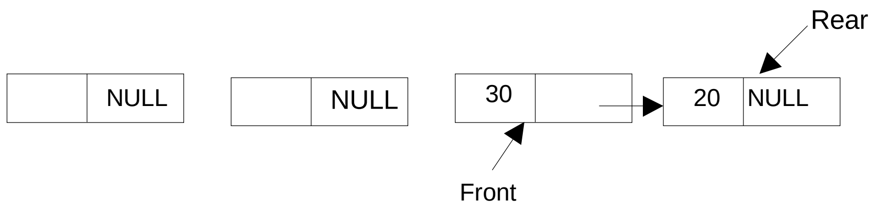

Fig. 5.21. X = pop() (*i.e.*; X = 20)

5.8.1. ALGORITHM FOR PUSHING AN ELEMENT TO A QUEUE

REAR is a pointer in queue where the new elements are added. FRONT is a pointer, which is pointing to the queue where the elements are popped. DATA is an element to be pushed.

```
1. Input the DATA element to be pushed
2. Create a New Node
3. NewNode → DATA = DATA
4. NewNode → Next = NULL
5. If(REAR not equal to NULL)
   (a) REAR → next = NewNode;
6. REAR =NewNode;
7. Exit
```

5.8.2. ALGORITHM FOR POPPING AN ELEMENT FROM A QUEUE

REAR is a pointer in queue where the new elements are added. FRONT is a pointer, which is pointing to the queue where the elements are popped. DATA is an element popped from the queue.

```
1. If (FRONT is equal to NULL)
   (a) Display "The Queue is empty"
2. Else
   (a) Display "The popped element is FRONT → DATA"
   (b) If(FRONT is not equal to REAR)
       (i) FRONT = FRONT → Next
   (c) Else
   (d) FRONT = NULL;
3. Exit
```

PROGRAM 5.5

```
//THIS PROGRAM WILL IMPLEMENT ALL THE OPERATIONS
//OF THE QUEUE, IMPLEMENTED USING LINKED LIST
```

```
//CODED AND COMPILED IN TURBO C

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

//A structure is created for the node in queue
struct queu
{
        int info;
        struct queu *next;//Next node address
};

typedef struct queu *NODE;

//This function will push an element into the queue
NODE push(NODE rear)
{
        NODE NewNode;
        //New node is created to push the data
        NewNode=(NODE)malloc(sizeof(struct queu));
        printf ("\nEnter the no to be pushed = ");
        scanf ("%d",&NewNode->info);
        NewNode->next=NULL;
        //setting the rear pointer
        if (rear != NULL)
            rear->next=NewNode;
        rear=NewNode;
        return(rear);
}

//This function will pop the element from the queue
NODE pop(NODE f,NODE r)
{
        //The Queue is empty when the front pointer is NULL
        if(f==NULL)
            printf ("\nThe Queue is empty");
        else
        {
            printf ("\nThe poped element is = %d",f->info);
            if(f ! = r)
                f=f->next;
                else
                        f=NULL;
        }
        return(f);
}

//Function to display the element of the queue
void traverse(NODE fr,NODE re)
{
        //The queue is empty when the front pointer is NULL
        if (fr==NULL)
                printf (“\nThe Queue is empty”);
        else
        {
                printf (“\nThe element(s) is/are = ”);
                while(fr != re)
                {
                        printf(“%d ”,fr->info);
                        fr=fr->next;
                };
                printf (“%d”,fr->info);
        }
}

void main()
{
        int choice;
        char option;
        //declaring the front and rear pointer
        NODE front, rear;
        //Initializing the front and rear pointer to NULL
        front = rear = NULL;
        dos
        {
                clrscr();
                printf (“1. Push\n”);
                printf (“2. Pop\n”);
                printf (“3. Traverse\n”);
                printf (“\n\nEnter your choice = ”);
                scanf (“%d”,&choice);
                switch(choice)
                {
                    case 1:
                            //calling the push function
                            rear = push(rear);
                            if (front==NULL)
                            {
                                    front=rear;
                            }
                            break;
                    case 2:
                            //calling the pop function by passing
                            //front and rear pointers
                            front = pop(front,rear);
                            if (front == NULL)
                                    rear = NULL;
                            break;
                    case 3:
                            traverse(front,rear);
                            break;
            }
            printf ("\n\nPress (Y/y) to continue = ");
            fflush(stdin);
            scanf ("%c",&option);
    }while(option == 'Y' || option == 'y');
}
```

PROGRAM 5.6

```
//THIS PROGRAM WILL IMPLEMENT ALL THE OPERATIONS
//OF THE QUEUE, IMPLEMENTED USING LINKED LIST
//CODED AND COMPILED IN TURBO C++

#include<iostream.h>
#include<conio.h>
#include<malloc.h>

//class is created for the queue
class Queue_Linked
{
        //A structure is created for the node in queue
        struct queu
```

```
		{
			int info;
			struct queu *next;//Next node address
		};

		struct queu *front;
		struct queu *rear;
		typedef struct queu *NODE;

		public:
			//Constructor is created
			Queue_Linked()
			{
				front = NULL;
				rear = NULL;
			}
			void push();
			void pop();
			void traverse();
};
//This function will push an element into the queue
void Queue_Linked::push()
{
		NODE NewNode;
		//New node is created to push the data
		NewNode=(NODE)malloc(sizeof(struct queu));
		cout<<"\nEnter the no to be pushed = ";
		cin>>NewNode->info;
		NewNode->next=NULL;
		//setting the rear pointer
		if (rear != NULL)
			rear->next = NewNode;
		rear=NewNode;
		if (front == NULL)
			front = rear;
}

//This function will pop the element from the queue
void Queue_Linked::pop()
{
		//The Queue is empty when the front pointer is NULL
		if (front == NULL)
```

```
                {
                        cout<<"\nThe Queue is empty";
                        rear = NULL;
                }
                else
                {
                        //Front element in the queue is popped
                        cout<<"\nThe popped element is = "<<front->info;
                        if (front != rear)
                                front=front->next;
                        else
                                front = NULL;
                }
}

//Function to display the element of the queue
void Queue_Linked::traverse()
{
                //The queue is empty when the front pointer is NULL
                if (front==NULL)
                        cout<<"\nThe Queue is empty";
                else
                {
                        NODE Temp_Front=front;
                        cout<<"\nThe element(s) is/are = ";
                        while(Temp_Front ! = rear)
                        {
                                cout<<Temp_Front->info;
                                Temp_Front=Temp_Front->next;
                        };
                        cout<<Temp_Front->info;
                }
}

void main()
{
                int choice;
                char option;
                Queue_Linked Qo;
                do
                {
                        clrscr();
```

```
                    cout<<"\n1. PUSH\n";
                    cout<<"2. POP\n";
                    cout<<"3. DISPLAY\n";
                    cout<<"\n\nEnter your choice = ";
                    cin>>choice;
                    switch(choice)
                    {
                            case 1:
                                    //calling the push function
                                    Qo.push();
                                    break;
                            case 2:
                                    //calling the pop function by passing
                                    //front and rear pointers
                                    Qo.pop();
                                    break;
                            case 3:
                                    Qo.traverse();
                                    break;
                    }
                    cout<<"\n\nPress (Y/y) to continue = ";
                    cin>>option;
            }while(option == 'Y' || option == 'y');
}
```

5.9. QUEUE USING TWO STACKS

A queue can be implemented using two stacks. Suppose STACK1 and STACK2 are the two stacks. When an element is pushed on to the queue, push the same on STACK1. When an element is popped from the queue, pop all elements of STACK1 and push the same on STACK2. Then pop the topmost element of STACK2; which is the first (front) element to be popped from the queue. Then pop all elements of STACK2 and push the same on STACK1 for next operation (*i.e., push or pop*). PROGRAM 5:5 gives the program to implement the queue using two stacks by singly linked list, coded in C language.

PROGRAM 5.7

```
//IMPLEMENTING THE QUEUE USING TWO STACKS
//BY SINGLY LINK LIST
//CODED AND COMPILED USING TURBO C
```

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

//Stack node is created with structure
struct stack
{
		int info;
		struct stack *next;
};

typedef struct stack *NODE;

//A new element is pushed to the stack
NODE push(NODE top)
{
		NODE NewNode;
		//New node is created
		NewNode=(NODE)malloc(sizeof(struct stack));
		NewNode->next=top;

		printf("\nEnter the no: to be pushed = ");
		scanf("%d",&NewNode->info);

		top=NewNode;
		return(top);
}

NODE pop(NODE top1)
{
		//checking for whether the queue is empty or not
		if(top1 == NULL)
		{
			printf("\nThe Queu is empty");
			return(top1);
		}

		//when top1->next == NULL the queue contains only one element
		if(top1->next == NULL)
		{
			//popping the only one element present in the queue
			printf("\nThe popped element is = %d",top1->info);
```

```
                free(top1);
                top1=NULL;
                return(top1);
        }

        NODE NewNode,top2,TEMP;

        //popping the elements from the first stack and pushing the
        //same element to the second stack
        top2=NULL;
        while(top1 != NULL)
        {

                TEMP=top1;
                //Creating the new node for the second stack
                NewNode=(NODE)malloc(sizeof(struct stack));
                NewNode->next=top2;
                NewNode->info=top1->info;
                top2=NewNode;
                top1=top1->next;
                free(TEMP);
        };

        //popping the top most element from the stack so as to
        //pop an element from the queue
        printf("\nThe popped element is = %d",top2->info);
        top2=top2->next;

        //popping rest of the element from the second stack and
        //pushing the same elements to the first stack
        top1=NULL;
        while(top2 != NULL)
        {
                TEMP=top2;
                //creating new nodes for the first stack
                NewNode=(NODE)malloc(sizeof(struct stack));
                NewNode->next=top1;
                NewNode->info=top2->info;
                top1=NewNode;
                top2=top2->next;
                free(TEMP);//freeing the nodes
};
return(top1);
```

```
}

//this function is to display all the elements in the queue
void traverse(NODE top)
{
        if(top == NULL)
        {
            printf("\nThe Queue is empty");
            return;
        }

        printf ("\nThe element(s) in the Queue is/are =");
        do
        {
            printf (" %d",top->info);
            top=top->next;
        }while(top != NULL);

        return;
    }

    void main()
    {
        int choice;
        char ch;
        NODE top;
        top=NULL;
        do
        {
            clrscr();
            //A menu for the stack operations
            printf("\n1. PUSH");
            printf("\n2. POP");
            printf("\n3. TRAVERSE");
            printf("\nEnter Your Choice = ");
            scanf ("%d", &choice);

            switch(choice)
            {
                case 1://Calling push() function by passing
                    //the structure pointer to the function
                    top=push(top);
```

```
                        break;

                case 2://calling pop() function
                        top=pop(top);
                        break;

                case 3://calling traverse() function
                        traverse(top);
                        break;

                default:
                        printf("\nYou Entered Wrong Choice");
                        break;
            }

            printf("\n\nPress (Y/y) To Continue = ");
            //Removing all characters in the input buffer
            //for fresh input(s), especially <<Enter>> key
            fflush(stdin);
            scanf("%c",&ch);
        }while(ch == 'Y' || ch == 'y');
}
```

5.10. POLYNOMIALS USING LINKED LIST

Different operations, such as addition, subtraction, division and multiplication of polynomials can be performed using linked list. In this section, we discuss about polynomial addition using linked list. Consider two polynomials $f(x)$ and $g(x)$; it can be represented using linked list as follows in Fig. 5.22.

$f(x) = ax^3 + bx + c$

$g(x) = mx^4 + nx^3 + ox^2 + px + q$

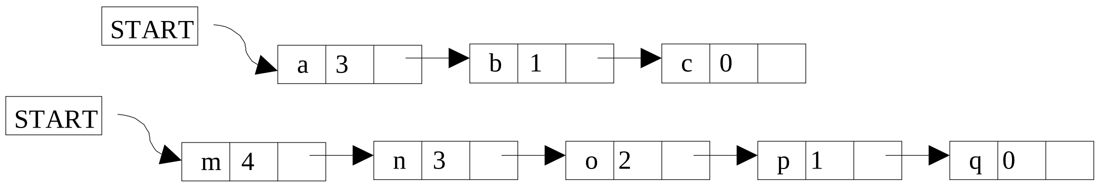

Fig. 5.22. Polynomial Representation

These two polynomials can be added by

$h(x) = f(x) + g(x) = mx^4 + (a + n)\, x^3 + ox^2 + (b + p)x + (c + q)$

i.e.; adding the constants of the corresponding polynomials of the same exponentials. $h(x)$ can be represented as in Fig. 5.23.

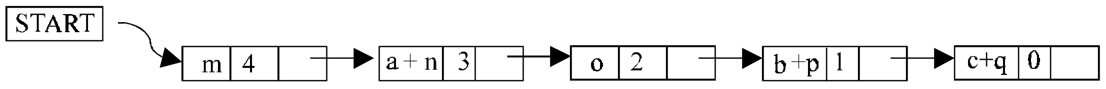

Fig. 5.23

PROGRAM 5.8

```
//Program of polynomial addition using linked list
//CODED AND COMPILED IN TURBO C

#include<stdio.h>
#include<malloc.h>

//structure is created for the node
struct node
{
        float coef;
        int expo;
        struct node *link;
};

typedef struct node *NODE;
//Function to add any node to the linked list
NODE insert(NODE start,float co,int ex)
{
        NODE ptr,tmp;
        //a new node is created
        tmp= (NODE)malloc(sizeof(struct node));
        tmp->coef=co;
        tmp->expo=ex;

        /*list empty or exp greater than first one */
        if(start==NULL || ex>start->expo)
        {
            tmp->link=start;//setting the start
            start=tmp;
        }
        else
```

```
                {
                        ptr=start;
                        while(ptr->link!=NULL && ptr->link->expo>ex)
                                ptr=ptr->link;
                        tmp->link=ptr->link;
                        ptr->link=tmp;
                        if(ptr->link==NULL)  /*item to be added in the end */
                                tmp->link=NULL;
                }
                return start;
}/*End of insert()*/

//This function is to add two polynomials
NODE poly_add(NODE p1,NODE p2)
{
                NODE p3_start,p3,tmp;
                p3_start=NULL;
                if(p1==NULL && p2==NULL)
                        return p3_start;

                while(p1!=NULL && p2!=NULL )
                {
                        //New node is created
                        tmp=(NODE)malloc(sizeof(struct node));
                        if(p3_start==NULL)
                        {
                                p3_start=tmp;
                                p3=p3_start;
                        }
                        else
                        {
                                p3->link=tmp;
                                p3=p3->link;
                        }
                        if(p1->expo > p2->expo)
                        {
                                tmp->coef=p1->coef;
                                tmp->expo=p1->expo;
                                p1=p1->link;
                        }
                        else
                                if(p2->expo > p1->expo)
                        {
                                tmp->coef=p2->coef;
                                tmp->expo=p2->expo;
                                p2=p2->link;
                        }
                        else
                                if(p1->expo == p2->expo)
                                {
                                        tmp->coef=p1->coef + p2->coef;
                                        tmp->expo=p1->expo;
                                        p1=p1->link;
                                        p2=p2->link;
                                }
}/*End of while*/
while(p1!=NULL)
{
                tmp=(NODE)malloc(sizeof(struct node));
                tmp->coef=p1->coef;
                tmp->expo=p1->expo;
                if (p3_start==NULL) /*poly 2 is empty*/
                {
                        p3_start=tmp;
                        p3=p3_start;
                }
                else
                {
                        p3->link=tmp;
                        p3=p3->link;
                }
                p1=p1->link;
        }/*End of while */
        while(p2!=NULL)
        {
                tmp=(NODE)malloc(sizeof(struct node));
                tmp->coef=p2->coef;
                tmp->expo=p2->expo;
                if (p3_start==NULL) /*poly 1 is empty*/
                {
                        p3_start=tmp;
                        p3=p3_start;
                }
                else
```

```
                    {
                            p3->link=tmp;
                            p3=p3->link;
                    }
                    p2=p2->link;
            }/*End of while*/
            p3->link=NULL;
            return p3_start;
}/*End of poly_add() */

//Inputting the two polynomials
NODE enter(NODE start)
{
            int i,n,ex;
            float co;
            printf("\nHow many terms u want to enter:");
            scanf("%d",&n);
            for(i=1;i<=n;i++)
            {
                    printf("\nEnter coeficient for term %d:",i);
                    scanf("%f",&co);
                    printf("Enter exponent for term %d:",i);
                    scanf("%d",&ex);
                    start=insert(start,co,ex);
            }
            return start;
}/*End of enter()*/

//This function will display the two polynomials and its
//added polynomials
void display(NODE ptr)
{
            if (ptr==NULL)
            {
                    printf ("\nEmpty\n");
                    return;
            }
            while(ptr!=NULL)
            {
                    printf ("(%.1fx^%d) + ", ptr->coef,ptr->expo);
                    ptr=ptr->link;
            }
```

```
        printf ("\b\b \n"); /* \b\b to erase the last + sign*/
}/*End of display()*/

void main()
{
        NODE p1_start,p2_start,p3_start;

        p1_start=NULL;
        p2_start=NULL;
        p3_start=NULL;

        printf("\nPolynomial 1 :\n");
        p1_start=enter(p1_start);

        printf("\nPolynomial 2 :\n");
        p2_start=enter(p2_start);
        //polynomial addition function is called
        p3_start=poly_add(p1_start,p2_start);

        clrscr();
        printf("\nPolynomial 1 is: ");
        display(p1_start);
        printf ("\nPolynomial 2 is: ");
        display(p2_start);
        printf ("\nAdded polynomial is: ");
        display(p3_start);
        getch();
}/*End of main()*/
```

5.11. DOUBLY LINKED LIST

A doubly linked list is one in which all nodes are linked together by multiple links which help in accessing both the successor (next) and predecessor (previous) node for any arbitrary node within the list. Every nodes in the doubly linked list has three fields: LeftPointer, RightPointer and DATA. Fig. 5.22 shows a typical doubly linked list.

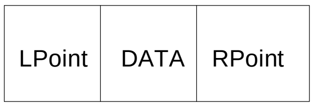

Fig. 5.24. A typical doubly linked list node

LPoint will point to the node in the left side (or previous node) that is LPoint will hold the address of the previous node. RPoint will point to the node in the right side (or next node) that is RPoint will hold the address of the next node. DATA will store the information of the node.

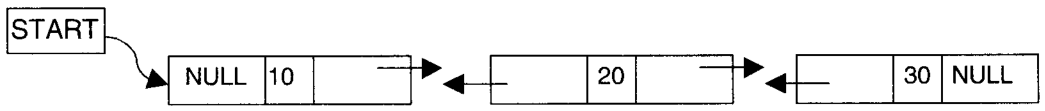

Fig. 5.25. Doubly Linked List

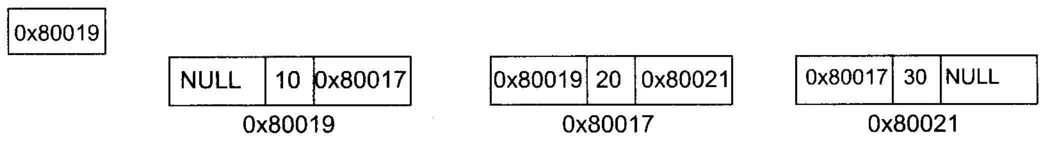

Fig. 5.26. Memory Representation of Doubly Linked List

5.11.1. REPRESENTATION OF DOUBLY LINKED LIST

A node in the doubly linked list can be represented in memory with the following declarations.

```
struct Node
{
        int DATA;
        struct Node *RChild;
        struct Node *LChild;
};
        typedef struct Node *NODE;
```

All the operations performed on singly linked list can also be performed on doubly linked list. Following figure will illustrate the insertion and deletion of nodes.

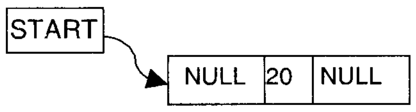

Fig. 5.27. Add(20)

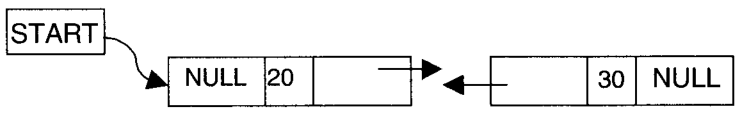

Fig 5.28. Insert (30) at the end

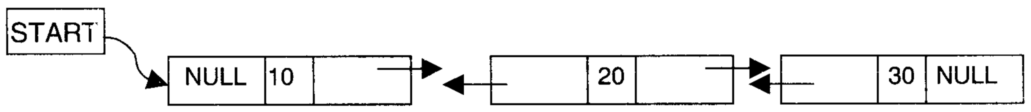

Fig 5.29. Insert (10) at the beginning

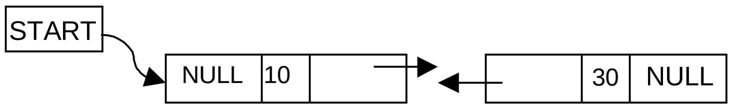

Fig 5.30. Delete a node at the 2nd position

5.11.2. ALGORITHM FOR INSERTING A NODE

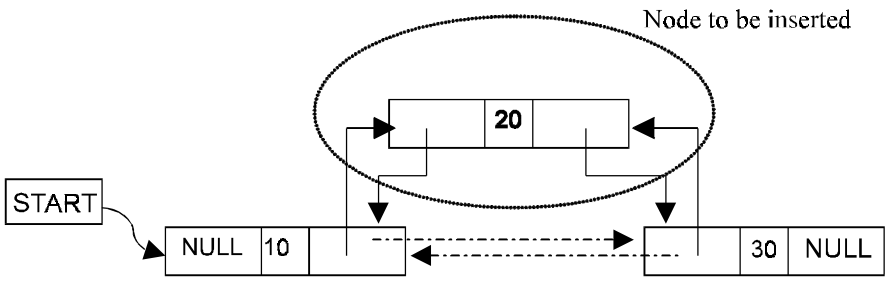

Fig. 5.31. Insert a node at the 2nd position

Suppose START is the first position in linked list. Let DATA be the element to be inserted in the new node. POS is the position where the NewNode is to be inserted. TEMP is a temporary pointer to hold the node address.

```
1. Input the DATA and POS
2. Initialize TEMP = START; i = 0
3. Repeat the step 4 if (i less than POS) and (TEMP is not equal to NULL)
4. TEMP = TEMP → RPoint; i = i +1
5. If (TEMP not equal to NULL) and (i equal to POS)
   (a) Create a New Node
   (b) NewNode → DATA = DATA
   (c) NewNode → RPoint = TEMP → RPoint
   (d) NewNode → LPoint = TEMP
   (e) (TEMP → RPoint) → LPoint = NewNode
   (f ) TEMP → RPoint = New Node
6. Else
   (a) Display "Position NOT found"
7. Exit
```

5.11.3. ALGORITHM FOR DELETING A NODE

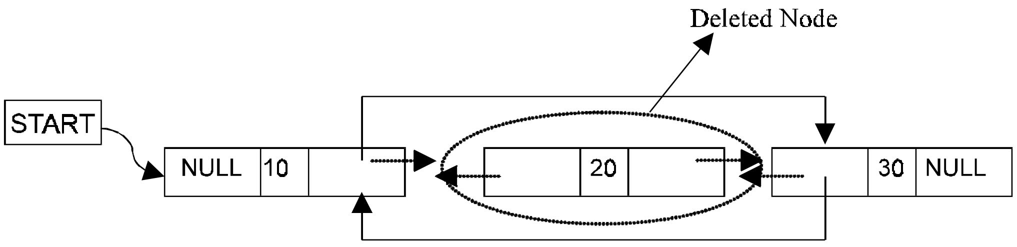

Fig. 5.32. Delete a node at the 2nd position

Suppose START is the address of the first node in the linked list. Let POS is the position of the node to be deleted. TEMP is the temporary pointer to hold the address of the node. After deletion, DATA will contain the information on the deleted node.

```
1. Input the POS
2. Initialize TEMP = START; i = 0
3. Repeat the step 4 if (i less than POS) and (TEMP is not equal to NULL)
4. TEMP = TEMP → RPoint; i = i +1
5. If (TEMP not equal to NULL) and (i equal to POS)
   (a) Create a New Node
   (b) NewNode → DATA = DATA
   (c) NewNode → RPoint = TEMP → RPoint
   (d) NewNode → LPoint = TEMP
   (e) (TEMP → RPoint) → LPoint = NewNode
   (f ) TEMP → RPoint = New Node
6. Else
   (a) Display "Position NOT found"
7. Exit
```

PROGRAM 5.9

```
// PROGRAM TO IMPLEMENT ALL THE OPERATIONS IN THE
//DOUBLY LINKED LIST
//CODED AND COMPILED IN TURBO C

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

```
//Structure is created for the node
struct node
{
            struct node *prev;
            int info;
            struct node *next;
      }*start;

      typedef struct node *NODE;

      //fucntion to create a doubly linked list
      void create_list(int num)
      {
            NODE q,tmp;
            //a new node is created
            tmp=(NODE)malloc(sizeof(struct node));
            tmp->info=num;//assigning the data to the new node
            tmp->next=NULL;
            if(start==NULL)
            {
                  tmp->prev=NULL;
                  start->prev=tmp;
                  start=tmp;
            }
            else
            {
                  q=start;
                  while(q->next!=NULL)
                        q=q->next;
                  q->next=tmp;
                  tmp->prev=q;
            }
}/*End of create_list()*/

//Function to add new node at the beginning
void addatbeg(int num)
{
            NODE tmp;
            //a new node is created for inserting the data
            tmp=(NODE)malloc(sizeof(struct node));
            tmp->prev=NULL;
            tmp->info=num;
            tmp->next=start;
            start->prev=tmp;
            start=tmp;
}/*End of addatbeg()*/

//This fucntion will insert a node in any specific position
void addafter(int num,int pos)
{
            NODE tmp,q;
            int i;
            q=start;
            //Finding the position to be inserted
            for(i=0;i<pos-1;i++)
            {
                  q=q->next;
                  if(q==NULL)
                  {
                        printf ("\nThere are less than %d elements\n",pos);
                        return;
                  }
            }
            //a new node is created
            tmp=(NODE)malloc(sizeof(struct node) );
            tmp->info=num;
            q->next->prev=tmp;
            tmp->next=q->next;
            tmp->prev=q;
            q->next=tmp;
}/*End of addafter() */

//Function to delete a node
void del(int num)
{
            NODE tmp,q;
            if(start->info==num)
            {
                  tmp=start;
                  start=start->next;  /*first element deleted*/
                  start->prev = NULL;
                  free(tmp);//Freeing the deleted node
                  return;
            }
            q=start;
            while(q->next->next!=NULL)
            {
                  if(q->next->info==num)      /*Element deleted in between*/
                  {
                        tmp=q->next;
                        q->next=tmp->next;
                        tmp->next->prev=q;
                        free(tmp);
                        return;
                  }
                  q=q->next;
            }
            if (q->next->info==num)     /*last element deleted*/
            {     tmp=q->next;
                  free(tmp);
                  q->next=NULL;
                  return;
            }
            printf("\nElement %d not found\n",num);
}/*End of del()*/

//Displaying all data(s) in the node
void display()
{
            NODE q;
            if(start==NULL)
            {
                  printf("\nList is empty\n");
                  return;
            }
            q=start;
            printf("\nList is :\n");
            while(q!=NULL)
            {
                  printf("%d ", q->info);
                  q=q->next;
            }
            printf("\n");
}/*End of display() */

//Function to count the number of nodes in the linked list
```

```
void count()
{
            NODE q=start;
            int cnt=0;
            while(q!=NULL)
            {
                  q=q->next;
                  cnt++;
            }
            printf("\nNumber of elements are %d\n",cnt);
}/*End of count()*/

//Reversing the linked list
void rev()
{
            NODE p1,p2;
            p1=start;
            p2=p1->next;
            p1->next=NULL;
            p1->prev=p2;
            while(p2!=NULL)
            {
                  p2->prev=p2->next;
                  p2->next=p1;
                  p1=p2;
                  p2=p2->prev; /*next of p2 changed to prev */
            }
            start=p1;
}/*End of rev()*/

void main()
{
            int choice,n,m,po,i;
            start=NULL;
            while(1)
            {
                  //Menu options for the doubly linked list operation
                  clrscr();
                  printf("\n1.Create List\n");
                  printf("2.Add at begining\n");
                  printf("3.Add after\n");
                  printf("4.Delete\n");
```

```
                printf("5.Display\n");
                printf("6.Count\n");
                printf("7.Reverse\n");
                printf("8.exit\n");
                printf("\nEnter your choice:");
                scanf("%d",&choice);
                //switch instruction is called to execute
                //correspoding function
                switch(choice)
                {
                 case 1:
                        printf("\nHow many nodes you want:");
                        scanf("%d",&n);
                        for(i=0;i<n;i++)
                        {
                               printf("\nEnter the element:");
                               scanf("%d",&m);
                               //create linked list function is called
                               create_list(m);
                        }
                        break;

                 case 2:
                        printf("\nEnter the element:");
                        scanf("%d",&m);
                        addatbeg(m);
                        break;
                 case 3:
                        printf("\nEnter the element:");
                        scanf("%d",&m);
printf("\nEnter the position after which this element is inserted:");
                        scanf("%d",&po);
                        addafter(m,po);
                        break;
                 case 4:
                        printf("\nEnter the element for deletion:");
                        scanf("%d",&m);
                        //Delete a node fucntion is called
                        del(m);
                        break;
                 case 5:
                        display();
                        getch();
```

```
                    break;
              case 6:
                    count();
                    getch();
                    break;
              case 7:
                    rev();
                    break;
              case 8:
                    exit(0);
              default:
                    printf("\nWrong choice\n");
                    getch();
        }/*End of switch*/
    }/*End of while*/
}/*End of main()*/
```

5.12. CIRCULAR LINKED LIST

A circular linked list is one, which has no beginning and no end. A singly linked list can be made a circular linked list by simply storing the address of the very first node in the linked field of the last node. A circular linked list is shown in Fig. 5.33. Implementation of circular linked list is in PROGRAM 5:8.

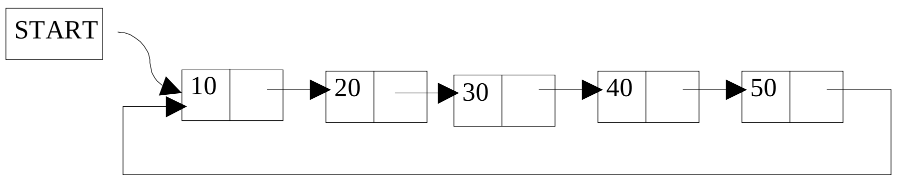

Fig. 5.33. Circular Linked list

A circular doubly linked list has both the successor pointer and predecessor pointer in circular manner as shown in the Fig. 5.34. Implementation of circular doubly linked list is left to the readers.

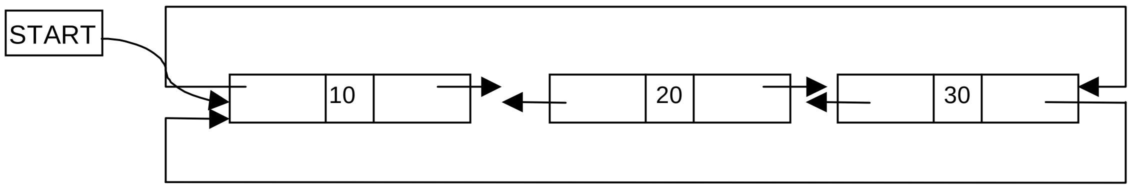

Fig. 5.34. Circular Doubly Linked list

PROGRAM 5.10

```
// PROGRAM TO IMPLEMENT CIRCULAR SINGLY LINKED LIST
//CODED AND COMPILED INTO TURBO C++
#include<iostream.h>
#include<process.h>

//class is created for the circular linked list
class Circular_Linked
{
        //structure node is created
        struct node
        {
            int info;
            struct node *link;
        };
        struct node *last;
        typedef struct node *NODE;

        public:
            //Constructor is defined
            Circular_Linked()
            {
                last=NULL;
            }

            void create_list(int);
            void addatbeg(int);
            void addafter(int,int);
            void del();
            void display();
};

//A circular list created in this function
void Circular_Linked::create_list(int num)
{
        NODE q,tmp;
        //New node is created
        tmp = (NODE)new(struct node);
        tmp->info = num;
        if (last == NULL)
        {
            last = tmp;
            tmp->link = last;
        }
        else
        {
            tmp->link = last->link; /*added at the end of list*/
            last->link = tmp;
            last = tmp;
        }
}/*End of create_list()*/

//This function will add new node at the beginning
void Circular_Linked::addatbeg(int num)
{
        NODE tmp;
        tmp = (NODE)new(struct node);
        tmp->info = num;
        tmp->link = last->link;
        last->link = tmp;
}/*End of addatbeg()*/

//Function to add new node at any position of the circular list
void Circular_Linked::addafter(int num,int pos)
{
        NODE tmp,q;
        int i;
        q = last->link;
        //finding the position to insert a new node
        for(i=0; i < pos-1; i++)
        {
            q = q->link;
            if (q == last->link)
            {
                cout<<"There are less than "<<pos<<" elements\n";
                return;
            }
        }/*End of for*/
        //creating the new node
        tmp = (NODE)new(struct node);
        tmp->link = q->link;
```

```
                tmp->info = num;
                q->link = tmp;
                if(q==last)     /*Element inserted at the end*/
                        last=tmp;
}/*End of addafter()*/

//Function to delete a node from the circular linked list
void Circular_Linked::del()
{
                int num;
                if(last == NULL)
                {
                        cout<<"\nList underflow\n";
                        return;
                }
                cout<<"\nEnter the number for deletion:";
                cin>>num;

                NODE tmp,q;
                if( last->link == last && last->info == num)  /*Only one element*/
                {
                        tmp = last;
                        last = NULL;
                        //deleting the node
                        delete(tmp);
                        return;
                }
                q = last->link;
                if(q->info == num)
                {
                        tmp = q;
                        last->link = q->link;
                        //deleting the node
                        delete(tmp);
                        return;
                }
                while(q->link != last)
                {
                        if(q->link->info == num)      /*Element deleted in between*/
                        {
                                tmp = q->link;
                                q->link = tmp->link;
                        delete(tmp);
                        cout<<"\n"<<num<<" deleted\n";
                        return;
                  }
                  q = q->link;
            }/*End of while*/
            if(q->link->info == num)     /*Last element deleted q->link=last*/
            {
                  tmp = q->link;
                  q->link = last->link;
                  delete(tmp);
                  last = q;
                  return;
            }
            cout<<"\nElement "<<num<<" not found\n";
}/*End of del()*/

//Function to display all the nodes in the circular linked list
void Circular_Linked::display()
{
            NODE q;
            if(last == NULL)
            {
                  cout<<"\nList is empty\n";
                  return;
            }
            q = last->link;
            cout<<"\nList is:\n";
            while(q != last)
            {
                  cout<< q->info;
                  q = q->link;
            }
            cout<<"\n"<<last->info;
}/*End of display()*/

void main()
{
            int choice,n,m,po,i;
            Circular_Linked co;//Object is created for the class
            while(1)
            {
                  //Menu options
```

```
cout<<"\n1.Create List\n";
cout<<"2.Add at begining\n";
cout<<"3.Add after \n";
cout<<"4.Delete\n";
cout<<"5.Display\n";
cout<<"6.Quit\n";
cout<<"\nEnter your choice:";
cin>>choice;

switch(choice)
{
 case 1:
        cout<<"\nHow many nodes you want:";
        cin>>n;
        for(i=0; i < n;i++)
        {
               cout<<"\nEnter the element:";
               cin>>m;
               co.create_list(m);
        }
        break;
 case 2:
        cout<<"\nEnter the element:";
        cin>>m;
        co.addatbeg(m);
        break;
 case 3:
        cout<<"\nEnter the element:";
        cin>>m;
        cout<<"\nEnter the position after which this element is inserted:";
        cin>>po;
        co.addafter(m,po);
        break;
 case 4:
        co.del();
        break;
 case 5:
        co.display();
        break;
 case 6:
        exit(0);
 default:
```

```
                              cout<<"\nWrong choice\n";
                    }/*End of switch*/
          }/*End of while*/
}/*End of main()*/
```

5.13. PRIORITY QUEUES

Priority Queue is a queue where each element is assigned a priority. In priority queue, the elements are deleted and processed by following rules.

1. An element of higher priority is processed before any element of lower priority.
2. Two elements with the same priority are processed according to the order in which they were inserted to the queue.

For example, Consider a manager who is in a process of checking and approving files in a first come first serve basis. In between, if any urgent file (with a high priority) comes, he will process the urgent file next and continue with the other low urgent files.

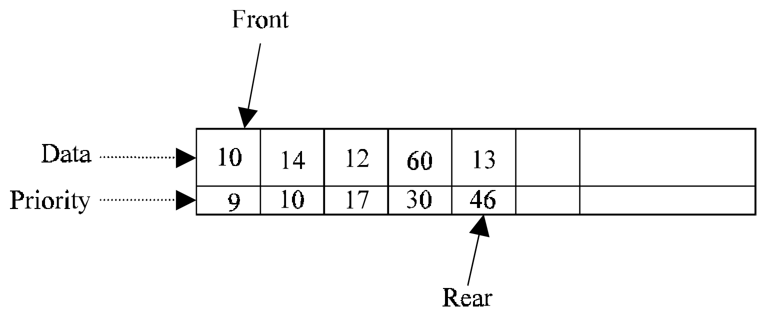

Fig. 5.35. Priority queue representation using arrays

Above Fig. 5.35 gives a pictorial representation of priority queue using arrays after adding 5 elements (10,14,12,60,13) with its corresponding priorities (9,10,17,30,46). Here the priorities of the data(s) are in ascending order. Always we may not be pushing the data in an ascending order. From the mixed priority list it is difficult to find the highest priority element if the priority queue is implemented using arrays. Moreover, the implementation of priority queue using array will yield n comparisons (in liner search), so the time complexity is O(n), which is much higher than the other queue (ie; other queues takes only O(1)) for inserting an element. So it is always better to implement the priority queue using linked list - where a node can be inserted at anywhere in the list - which is discussed in this section.

A node in the priority queue will contain DATA, PRIORITY and NEXT field. DATA field will store the actual information; PRIORITY field will store its corresponding priority of the DATA and NEXT will store the address of the next node. Fig. 5.36 shows the linked list representation of the node when a DATA (*i.e.*, 12) and PRIORITY (*i.e.*, 17) is inserted in a priority queue.

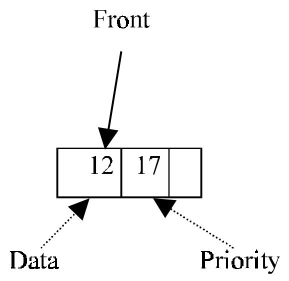

Fig. 5.36. Linked list representation of priority queue

When an element is inserted into the priority queue, it will check the priority of the element with the element(s) present in the linked list to find the suitable position to insert. The node will be inserted in such a way that the data in the priority field(s) is in ascending order. We do not use rear pointer when it is implemented using linked list, because the new nodes are not always inserted at the rear end. Following figures will illustrate the push and pop operation of priority queue using linked list.

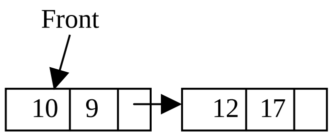

Fig. 5.37. push(DATA =10, PRIORITY = 9)

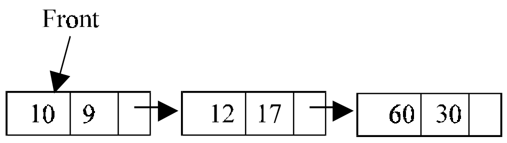

Fig. 5.38. push(DATA = 60, PRIORITY = 30)

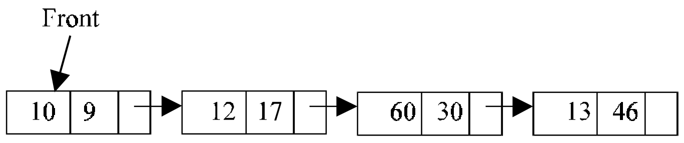

Fig. 5.39. push(DATA = 13, PRIORITY = 46)

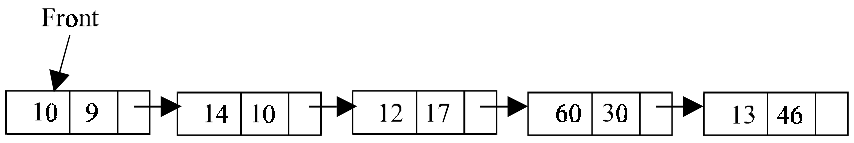

Fig. 5.40. push(DATA = 14, PRIORITY = 10)

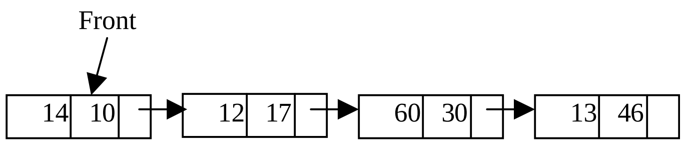

Fig. 5.41. *x* = pop() (*i.e.*, 10)

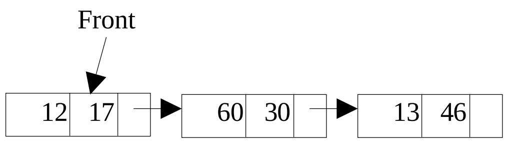

Fig. 5.42. *x* = pop() (*i.e.*, 14)

PROGRAM 5.11

```
//PROGRAM TO IMPLEMENT PRIORITY QUEUE USING LINKED LIST
//CODED AND COMPILED USING TURBO C

#include<conio.h>
#include<stdio.h>
#include<malloc.h>
#include<process.h>

//A structure is created for a node
struct node
{
        int priority;
        int info;
        struct node *link;
};

typedef struct node *NODE;

//This function will insert a data and its priority
NODE insert(NODE front)
{
        NODE tmp,q;
        int added_item,item_priority;
        //New node is created
        tmp = (NODE)malloc(sizeof(struct node));
        printf("\nInput the item value to be added in the queue:");
        scanf("%d",&added_item);
        printf("\nEnter its priority:");
        scanf("%d",&item_priority);
        tmp->info = added_item;
        tmp->priority = item_priority;
        /*Queue is empty or item to be added has priority more than first item*/
        if(front == NULL || item_priority < front->priority)
        {
            tmp->link = front;
            front = tmp;
        }
        else
        {
            q = front;
            while(q->link != NULL && q->link->priority <= item_priority)
                q=q->link;
            tmp->link = q->link;
            q->link = tmp;
        }/*End of else*/
        return(front);
}/*End of insert()*/

//Following function is to delete a node from the priority queue
NODE del(NODE front)
{
        NODE tmp;
        if(front == NULL)
            printf("\nQueue Underflow\n");
        else
        {
            tmp = front;
            printf("\nDeleted item is %d\n",tmp->info);
            front = front->link;
            free(tmp);
        }
        return(front);
}/*End of del()*/

void display(NODE front)
{
		NODE ptr;
		ptr = front;
		if(front == NULL)
			printf("\nQueue is empty\n");
		else
		{	printf("\nQueue is:\n");
			printf("\nPriority Item\n");
			while(ptr != NULL)
			{
				printf("%5d %5d\n",ptr->priority,ptr->info);
				ptr = ptr->link;
			}
		}/*End of else */
}/*End of display() */


void main()
{
		int choice;
		NODE front=NULL;
		while(1)
		{
			clrscr();
			//Menu options
			printf("\n1.Insert\n");
			printf("2.Delete\n");
			printf("3.Display\n");
			printf("4.Quit\n");
			printf("\nEnter your choic");
			scanf("%d", &choice);

			switch(choice)
			{
			 case 1:
				front=insert(front);
				break;
			 case 2:
				front=del(front);
				getch();
				break;
			 case 3:
				display(front);
```

```
                        getch();
                        break;
                  case 4:
                        exit(1);
                  default :
                        printf("\nWrong choice\n");
                  }/*End of switch*/
            }/*End of while*/
}/*End of main()*/
```

SELF REVIEW QUESTIONS

1. Write an algorithm to count the number of nodes in a singly linked list?
[*Calicut - APR 1997* (*BTech*), *MG - MAY 2000* (*BTech*)]
2. Write an algorithm to insert a node in a linked list? [*MG - MAY 2004* (*BTech*)]
3. Describe how a polynomial is represented using singly linked lists. Write an algorithm to add two polynomials represented using linked list?
[*MG - MAY 2004* (*BTech*), *MG - MAY 2003* (*BTech*), *MG - NOV 2003* (*BTech*), *MG - NOV 2002* (*BTech*), *MG - MAY 2002* (*BTech*), *MG - MAY 2000* (*BTech*) *CUSAT - DEC 2003* (*MCA*)]
4. What is doubly linked list? Write an algorithm to add and delete a node from it?
[*MG - NOV 2004* (*BTech*), *MG - MAY 2000* (*BTech*) *ANNA - DEC 2003* (*BE*), *CUSAT - DEC 2003* (*MCA*) *ANNA - MAY 2004* (*MCA*), *ANNA - DEC 2004* (*BE*) *KERALA - MAY 2003* (*BTech*)]
5. Explain the array representation of a priority queue? Write an algorithm for deleting and inserting in a priority queue? Mention its Applications?
[*MG - NOV 2004* (*BTech*), *MG - MAY 2003* (*BTech*), *MG - NOV 2003* (*BTech*), *CUSAT - NOV 2002* (*BTech*), *MG - NOV 2002* (*BTech*)]
6. Discuss the advantages and disadvantages of singly linked list?
[*MG - MAY 2003* (*BTech*)]
7. Explain the structure of a doubly linked list. Write a general algorithm for inserting and deleting nodes in the middle? [*MG - MAY 2003* (*BTech*), *MG - NOV 2002* (*BTech*)]
8. Compare and distinguish between singly linked lists and doubly linked lists?
[*CUSAT - APR 1998* (*BTech*), *MG - NOV 2003* (*BTech*) *KERALA - MAY 2001* (*BTech*), *CUSAT - JUL 2002* (*MCA*)]
9. Discuss the advantages and disadvantages of linked list over arrays?
[*MG - NOV 2002* (*BTech*)]

10. Write down the steps to invert a singly-linked circular linked list?
[*CUSAT - MAY 2000* (*BTech*), *MG - MAY 2002* (*BTech*)
11. How do you use circular queue in programming? [*MG - MAY 2000* (*BTech*)]
12. How doubly linked list can be used for dynamic storage? [*MG - MAY 2000* (*BTech*)]
13. What is a Priority queue? How is it represented in the memory?
[*ANNA - DEC 2004* (*BE*), *CUSAT - APR 1998* (*BTech*)]
14. What is a linked list? What are its advantages over array? How is linked list implemented in the memory? [*CUSAT - APR 1998* (*BTech*)]
15. Develop an algorithm to insert a node to the right of kth node of a singly linked linear list.
[*CUSAT - APR 1998* (*BTech*)]
16. Evaluate the complexity of the algorithm to add two polynomials in the form of linked lists [*CUSAT - MAY 2000* (*BTech*)]
17. Compare linear data structures with linked storage data types.
[*CUSAT - MAY 2000* (*BTech*)]
18. Explain the representation of polynomials using linked lists.
[*CUSAT - NOV 2002* (*BTech*)]
19. Write algorithms for adding and deleting elements from a circular queue implemented as linked list. [*CUSAT - DEC 2003* (*MCA*)]
20. Explain any three operations on a linked list. Write algorithms for these operations.
[*ANNA - DEC 2003* (*BE*)]
21. Discuss the advantage of circular queue with examples. [*ANNA - MAY 2004* (*MCA*)]
22. Explain how pointers are used to implement linked list structures.
[*ANNA - MAY 2004* (*BE*)]
23. Differentiate singly linked list and circularly linked list.
[*KERALA - DEC 2003* (*BTech*), *ANNA - MAY 2003* (*BE*)]
24. Write an algorithm to traverse elements in a singly linked list
[*KERALA - DEC 2004* (*BTech*)]
25. Explain about frequency count and give an example. Write procedures for circular queue operations. [*KERALA - JUN 2004* (*BTech*)]
26. Write procedure for searching an element in a singly linked list.
[*KERALA - JUN 2004* (*BTech*)]
27. What is the advantage of a doubly linked list compared to a singly linked list?
[*KERALA - MAY 2002* (*BTech*)]