Unit - V

5 Stack

Syllabus

Basic concept, stack Abstract Data Type, Representation of Stacks Using Sequential Organization, stack operations, Multiple Stacks, Applications of Stack- Expression Evaluation and Conversion, Polish notation and expression conversion, Need for prefix and postfix expressions, Postfix expression evaluation, Linked Stack and Operations. Recursion- concept, variants of recursion- direct, indirect, tail and tree, Backtracking algorithmic strategy, use of stack in backtracking.

Contents

5.1 Basic Concept

Let us have the formal definition of the stack.

Definition :

A stack is an **ordered list** in which all insertions and deletions are made at **one end**, called the **top**. If we have to make stack of elements 10, 20, 30, 40, 50, 60 then 10 will be the bottommost element and 60 will be the topmost element in the stack. A stack is shown in Fig. 5.1.1.

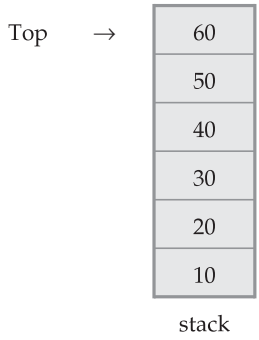


Fig. 5.1.1 Stack

Example

The typical example can be a stack of coins. The coins can be arranged one on another, when we add a new coin it is always placed on the previous coin and while removing the coin the recently placed coin can be removed. The example resembles the concept of stack exactly.

5.2 Stack Abstract Data Type

Stack is a data structure which posses LIFO i.e. Last In First Out property. The abstract data type for stack can be as given below.

Abstract DataType stack
{
Instances : Stack is a collection of elements in which insertion and deletion of elements is done by one end called top.
Preconditions :
1. Stfull () : This condition indicates whether the stack is full or not. If the stack is full then we cannot insert the elements in the stack.
2. Stempty () : This condition indicates whether the stack is empty or not. If the stack is empty then we cannot pop or remove any element from the stack.
Operations :
1. Push : By this operation one can push elements onto the stack. Before performing push we must check stfull () condition.
2. Pop : By this operation one can remove the elements from stack. Before popping the elements from stack We should check stempty () condition.
}

Review Question

1. *What is ADT ? Write ADT for stack operations.*

5.3 Representation of Stacks using Sequential Organization

Declaration 1 :

```
#define size 100
int  stack[size], top = -1;
```

In the above declaration stack is nothing but an array of integers. And most recent index of that array will act as a top.

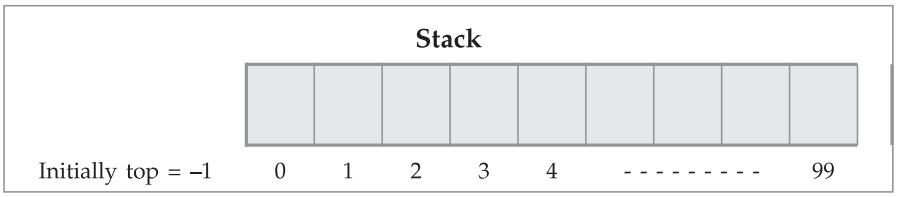


Fig. 5.3.1 Stack using one dimensional array

The stack is of the size 100. As we insert the numbers, the top will get incremented. The elements will be placed from 0^{th} position in the stack. At the most we can store 100 elements in the stack, so at the most last element can be at (size-1) position, i.e., at index 99.

Declaration 2 :

```
#define size 10
struct stack  {
          int s[size];
              int top;
                    }st;
```

In the above declaration stack is declared as a structure.

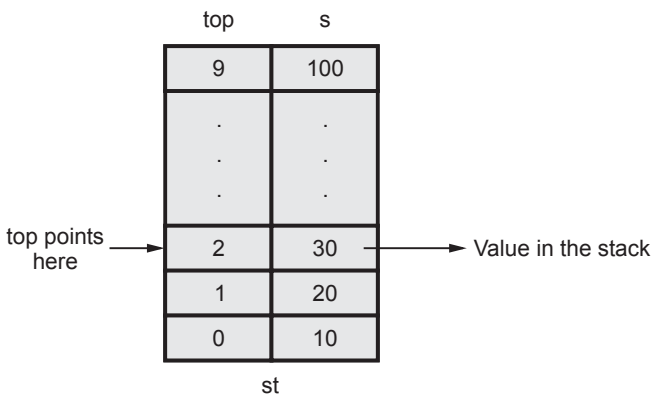


Fig. 5.3.2 Stack using structure

Now compare declaration 1 and 2. Both are for stack declaration only. But the second declaration will always preferred. Why ? Because in the second declaration we have used a structure for stack elements and top. By this we are binding or co-relating top variable with stack elements. Thus top and stack are associated with each other by putting them together in a structure. The stack can be passed to the function by simply passing the structure variable.

We will make use of the second method of representing the stack in our program.

The stack can also be used in the databases. For example if we want to store marks of all the students of forth semester we can declare a structure of stack as follows

```
# define size 60
typedef struct student
      {
            int roll.no;
            char name[30];
            float marks;
      }stud;
      stud S1[size];
      int top = –1;
```

The above structure will look like this

	roll_no	name	marks
59			
⋮			
top = 3 →	40	Mita	66
2	30	Rita	91
1	20	Geeta	88.3
0	10	Seeta	76.5

Thus we can store the data about whole class in our stack. The above declaration means creation of stack. Hence we will write only push and pop function to implement the stack. And before pushing or popping we should check whether stack is empty or full.

5.4 Stack Operations

- Basically there are two important stack operations - **(1) Push** and **(2) Pop.**
- Performing **Push operation** means we are inserting the elements onto the stack. And **Pop operation** means we are removing the element from the stack.
- Before pushing we need to check **stack full condition** and before performing pop operation we need to check **stack empty condition.**

5.4.1 Stack Empty Operation

Initially stack is empty. At that time the top should be initialized to – 1 or 0. If we set top to – 1 initially then the stack will contain the elements from 0^{th} position and if we set top to 0 initially, the elements will be stored from 1^{st} position, in the stack. Elements may be pushed onto the stack and there may be a case that all the elements are removed from the stack. Then the stack becomes empty. Thus whenever top reaches to – 1 we can say the stack is empty.

```
int stempty()
{
 if(st.top==-1)
  return 1;
 else
  return 0;
}
```

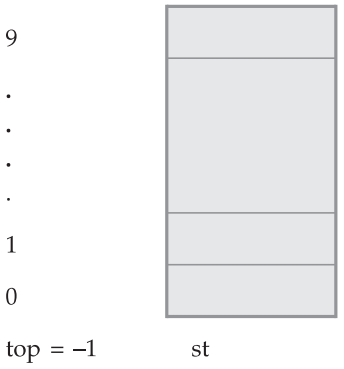


Fig. 5.4.1 Stack empty condition

Key Point *If top = – 1 means stack empty.*

5.4.2 Stack Full Operation

In the representation of stack using arrays, size of array means size of stack. As we go on inserting the elements the stack gets filled with the elements. So it is necessary before inserting the elements to check whether the stack is full or not. Stack full condition is achieved when stack reaches to maximum size of array.

```
int stfull()
{
 if(st.top>=size-1)
  return 1;
 else
  return 0;
}
```

Thus stfull is a Boolean function if stack is full it returns 1 otherwise it returns 0.

Key Point *If top >= size means stack is full.*

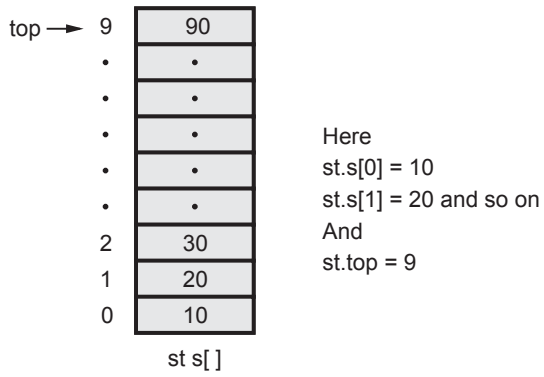


Fig. 5.4.2 Stack full condition

5.4.3 The Push and Pop Operations

We will now discuss the two important functions which are carried out on a stack. push is a function which inserts new element at the top of the stack. The function is as follows.

```
void push(int item)
{
 st.top++; / * top pointer is set to next location */
 st.s[st.top] =item; /* placing the element at that location */
}
```

Note that the push function takes the parameter **item** which is actually the element which we want to insert into the stack - means we are pushing the element onto the stack. In the function we have checked whether the stack is full or not, if the stack is not full then only the insertion of the element can be achieved by means of push operation.

Now let us discuss the operation pop, which deletes the element at the top of the stack. The function pop is as given below -

Note that always top element can be deleted.

```
int pop()
{
 int item;
 item = st.s[st.top];
 st.top – –;
 return(item);
}
```

In the choice of pop- it invokes the function 'stempty' to determine whether the stack is empty or not. If it is empty, then the function generates an error as stack underflow ! If not, then pop function returns the element which is at the top of the stack. The value at the top is stored in some variable as item and it then decrements the value of the top, which now points to the element which is just under the element being

retrieved from the stack. Finally it returns the value of the element stored in the variable item. Note that this is what called as logical deletion and not a physical deletion, i.e. even when we decrement the top, the element just retrieved from the stack remains there itself, but it no longer belongs to the stack. Any subsequent push will overwrite this element.

Push operation can be shown by following Fig. 5.4.3.

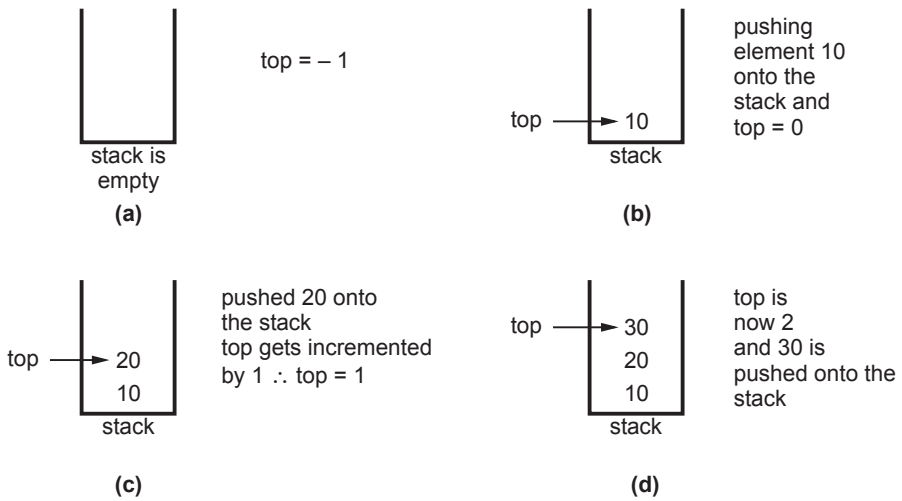


Fig. 5.4.3 Performing push operation

The pop operation can be shown by following Fig. 5.4.4.

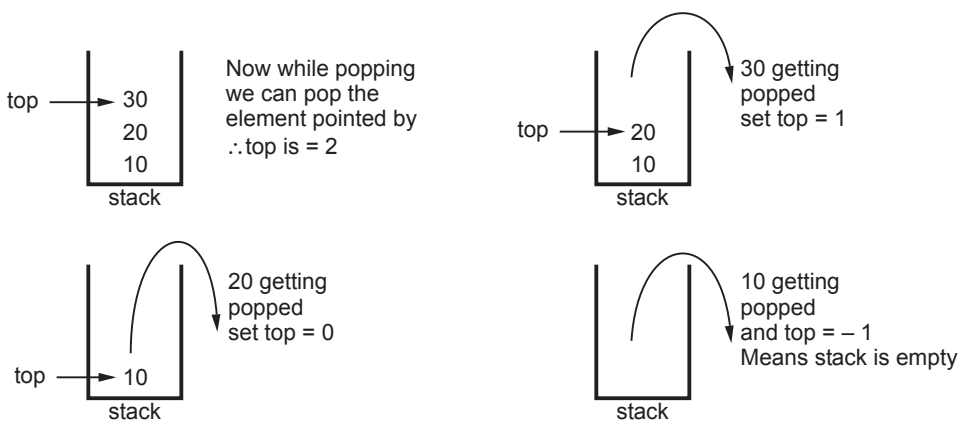


Fig. 5.4.4 Performing pop operation

Key Point *If stack is empty we cannot pop and if stack is full we cannot push any element.*

'C++' Program

```
/*************************************************************
Program for  implementing a stack using arrays.It involves various operations such as
push, pop, stack empty, stack full and display.
*************************************************************/
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
#define size 5
/* stack structure */
class STACK_CLASS
{
 private:
 struct stack
 {
     int s[size];
     int top;
  } st;
 public:
        STACK_CLASS();
        int stfull();
        void push(int item);
        int stempty();
        int pop();
        void display();
};
//constructor is used to initialise stack
STACK_CLASS::STACK_CLASS()
{
        st.top=-1;
        for(int i=0;i<size;i++)
                st.s[i]=0;
}

/*
 The stfull Function
 Input:none
 Output:returns 1 or 0 for stack full or not
 Called By:main
 Calls:none
*/
int STACK_CLASS::stfull()
{
 if(st.top>=size-1)
        return 1;
 else
```

```
        return 0;
}
/*
 The push Function
 Input:item which is to be pushed
 Output:none-simply pushes the item onto the stack
 Called By:main
 Calls:none
*/
void STACK_CLASS::push(int item)
{
 st.top++;
 st.s[st.top] =item;
}
/*
 The stempty Function
 Input:none
 Output:returns 1 or 0 for stack empty or not
 Called By:main
 Calls:none
*/
int STACK_CLASS::stempty()
{
 if(st.top==–1)
  return 1;
 else
  return 0;
}
/*
 The pop Function
 Input:none
 Output:returns the item which is popped from the stack
 Called By:main
 Calls:none
*/
int STACK_CLASS::pop()
{
 int item;
 item=st.s[st.top];
 st.top – – ;
 return(item);
}
/*
 The display Function
 Input:none
 Output:none-displays the contents of the stack
 Called By:main
```

```
 Calls:none
*/
void STACK_CLASS::display()
{
 int i;
 if(stempty())
   cout<<"\n Stack Is Empty!";
 else
 {
  for(i=st.top;i>=0;i--)
  cout<<"\n"<<st.s[i];
 }
}
/*
 The main Function
 Input:none
 Output:none
 Called By:O.S.
 Calls:push,pop,stempty,stfull,display
*/
void main(void)
{
        int item,choice;
        char ans;
        STACK_CLASS obj;
        clrscr();
        cout<<"\n\t\t Implementation Of Stack";
        do
        {
                cout<<"\n Main Menu";
                cout<<"\n1.Push\n2.Pop\n3.Display\n4.exit";
                cout<<"\n Enter Your Choice: ";
                cin>>choice;
                switch(choice)
                {
                  case 1:     cout<<"\n Enter The item to be pushed ";
                              cin>>item;
                              if(obj.stfull())
                              cout<<"\n Stack is Full!";
                        else
                               obj.push(item);
                               break;
                  case 2:if(obj.stempty())
                           cout<<"\n Empty stack!Underflow !!";
                        else
                        {
                          item=obj.pop();
```

```
                    cout<<"\n The popped element is "<<item;
                   }
                   break;
            case 3:obj.display();
                        break;
            case 4:exit(0);
           }
           cout<<"\n Do You want To Continue?";
           ans=getche();
      }while(ans ==Y ||ans ==y);
      getch();
}
/******************** End Of Program ************************/
```

Output

```
          Implementation Of Stack
 Main Menu
1.Push
2.Pop
3.Display
4.exit
 Enter Your Choice: 1

 Enter The item to be pushed 10

 Do You want To Continue?y
 Main Menu
1.Push
2.Pop
3.Display
4.exit
 Enter Your Choice: 1

 Enter The item to be pushed 20

 Do You want To Continue?y
 Main Menu
1.Push
2.Pop
3.Display
4.exit
 Enter Your Choice: 1

 Enter The item to be pushed 30

 Do You want To Continue?y
 Main Menu
```

```
1.Push
2.Pop
3.Display
4.exit
 Enter Your Choice: 2
 The popped element is 30
 Do You want To Continue?y
 Main Menu
1.Push
2.Pop
3.Display
4.exit
 Enter Your Choice: 3
20
10
 Do You want To Continue?y
```

5.5 Multiple Stacks

- In a single array any number of stacks can be adjusted. And push and pop operations on each individual stack can be performed.
- The following Fig. 5.5.1 shows how multiple stacks can be stored in a single dimensional array

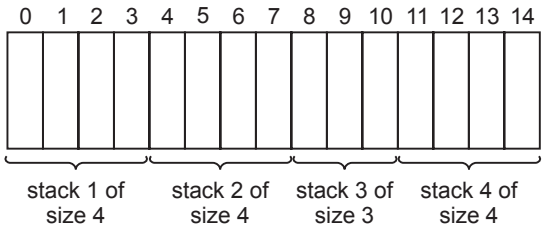


Fig. 5.5.1 Multiple stacks in one dimensional array

- Each stack in one dimensional array can be of any size.
- The only one thing which has to be maintained that total size of all the stacks < = size of single dimensional array.

Example

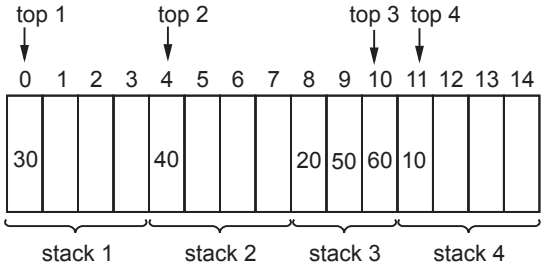
Let us perform following operations on the stacks -

1. push 10 in stack 4
2. push 20 in stack 3
3. push 30 in stack 1
4. push 40 in stack 2

5. push 50 in stack 3

6. push 60 in stack 3

The multiple stack will then be as follows -



- Here stack 3 is now full and we can not insert the elements in stack 3.
- The top 1, top 2, top 3, top 4 indicates the various top pointers for stack 1, stack 2, stack 3 and stack 4 respectively.
  - **stack 1-**array [0] will be lower bound and array [3] will be upper bound. That is we can perform stack 1 operation for array [0] to array [3] only.
  - **stack 2-**The area for stack 2 will be from array [4] to array [7]
  - **stack 3-**From array [8] to array [10]
  - **stack 4**-From array [11] to array [14]

Thus you can declare any number of stacks having different sizes of them.

Let us see the implementation of multiple stacks

C++ Program

```
/************************************************************
Program to implement multiple stacks using single array
************************************************************/
#include<iostream>
using namespace std;
#define MAX 20
class MultipleStack {
        private:
                int stack[MAX];
                int lb,ub;
        public:
                int size[MAX];
        /*Constructor defined*/
                MultipleStack()
                {
                        for(int i=1;i<=MAX;i++)
                                stack[i]=-1;/*for initialization of the entire array of stacks*/
```

```
        }
    /* set_stack Function */
        void set_stack(int index)
        {
            int sum,i;
            if(index==1)
            {
                lb=1;
                ub=size[index];
            }
            else
            {
                sum=0;
                for(i=1;i<index;i++)
                    sum=sum+size[i];
                lb=sum+1;
                ub=sum+size[index];
            }
        }
    /* stfull Function */
        int stfull(int index)
        {
            int top,i,sum;
            set_stack(index);
            for(top=lb;top<=ub;top++)
            {
                if(stack[top]==-1)
                    break;
            }
            if(top-1==ub)
                return 1;
            else
                return 0;
        }
    /*  stempty Function */
        int stempty(int index)
        {
            int top;
            set_stack(index);
            for(top=lb;top<=ub;top++)
            {
                if(stack[top]!=-1)
                    return 0;
                return 1;
            }
        }
    /*  push Function */
```

```
        void push(int item)
        {
                int top;
                for(top=lb;top<=ub;top++)
                        if(stack[top]==-1)
                                break;
                stack[top]=item;
                return;
        }

    /* pop Function */
        int pop()
        {
                int top,item;;
                for(top=lb;top<=ub;top++)
                        if(stack[top]==-1)
                                break;
                top--;
                item=stack[top];
                stack[top]=-1;
                return item;
        }
    /*  display Function */
        void display(int index)
        {
                int top;
                set_stack(index);
                for(top=lb;top<=ub;top++)
                {
                        if(stack[top]!=-1)
                                cout<<"   "<<stack[top];
                }
        }

/*  display_all Function */
        void display_all(int n)
        {
                int top,index;
                for(index=1;index<=n;index++)
                {
                        cout<<"\nstack number "<<index<<" is";
                        set_stack(index);
                        for(top=lb;top<=ub;top++)
                        {
                                if(stack[top]!=-1)
                                        cout<<"  "<<stack[top];
                        }
```

```
                    }
                }
};
/*  main Function */
int main(void)
{
        int n,index,i,item,choice;
        char ans;
        ans='y';
        MultipleStack ms;
        cout<<"\n\t Program For multiple stacks using single array";
        cout<<"\n Enter how many stacks are there";
        cin>>n;
        cout<<"\n Enter the size for each stack";
        for(i=1;i<=n;i++)
        {
                cout<<"\n size for stack"<<i<<" is: ";
                cin>>ms.size[i];
        }
        do
        {
                cout<<"\n\t\t Main Menu";
                cout<<"\n 1.Push \n2.Pop \n3.Display \n4.Dispaly all";
                cout<<"\n Enter Your choice";
                cin>>choice;
                switch(choice)
                {
                        case 1:cout<<"\n Enter the item to be pushed";
                                cin>>item;
                                cout<<"\n In Which stack?";
                          cin>>index;
                          if(ms.stfull(index))
                                cout<<"\n Stack is Full,can not Push";
                        else
                                ms.push(item);
                        break;
                        case 2:cout<<"\n From Which stack?";
                                cin>>index;
                                if(ms.stempty(index))
                                        cout<<"\n Stack number: "<<index<<" is empty!";
                                else
                                {
                                        item=ms.pop();
                                        cout<<"\n"<<item<<" is popped from stack "<<index;
                                }
                        break;
                        case 3:cout<<"\n Which stack has to be displayed?";
```

```
                        cin>>index;
                        ms.display(index);
                    break;
                    case 4:ms.display_all(n);
                    break;
                    default:cout<"\n Exitting";
                }
                cout<<"\n Do you wish to continue?";
                cin>>ans;
        }while(ans=='y'||ans=='Y');
        return 0;
}
```

Output

```
        Program For multiple stacks using single array
Enter how many stacks are there 4
Enter the size for each stack
size for stack1 is: 2
size for stack2 is: 3
size for stack3 is: 4
size for stack4 is: 5
                Main Menu
1.Push
2.Pop
3.Display
4.Dispaly all
Enter Your choice 1
Enter the item to be pushed 10
In Which stack?1
Do you wish to continue?y

                Main Menu
1.Push
2.Pop
3.Display
4.Dispaly all
Enter Your choice1

Enter the item to be pushed 20

In Which stack?3

Do you wish to continue?y

                Main Menu
1.Push
2.Pop
3.Display
```

```
4.Dispaly all
 Enter Your choice1

 Enter the item to be pushed 30

 In Which stack?3

 Do you wish to continue?y

              Main Menu
 1.Push
 2.Pop
 3.Display
 4.Dispaly all
 Enter Your choice1
 Enter the item to be pushed 40
 In Which stack?4
 Do you wish to continue?y
              Main Menu
 1.Push
 2.Pop
 3.Display
 4.Dispaly all
 Enter Your choice1
 Enter the item to be pushed 50
 In Which stack?2
 Do you wish to continue?y
              Main Menu
 1.Push
 2.Pop
 3.Display
 4.Dispaly all
 Enter Your choice 1
 Enter the item to be pushed 60
 In Which stack?2
 Do you wish to continue?y
              Main Menu
 1.Push
 2.Pop
 3.Display
 4.Dispaly all
 Enter Your choice4

stack number 1 is  10
stack number 2 is  50  60
stack number 3 is  20  30
stack number 4 is  40
```

```
Do you wish to continue?y

            Main Menu
1.Push
2.Pop
3.Display
4.Dispaly all
Enter Your choice2

From Which stack?3

30 is popped from stack 3
Do you wish to continue?y

            Main Menu
1.Push
2.Pop
3.Display
4.Dispaly all
Enter Your choice4

stack number 1 is  10
stack number 2 is  50  60
stack number 3 is  20
stack number 4 is s 40
Do you wish to continue?n
```

5.5.1 Two Stacks in Single Array

Two stacks can be adjusted in a single array with n numbers, which is as shown below

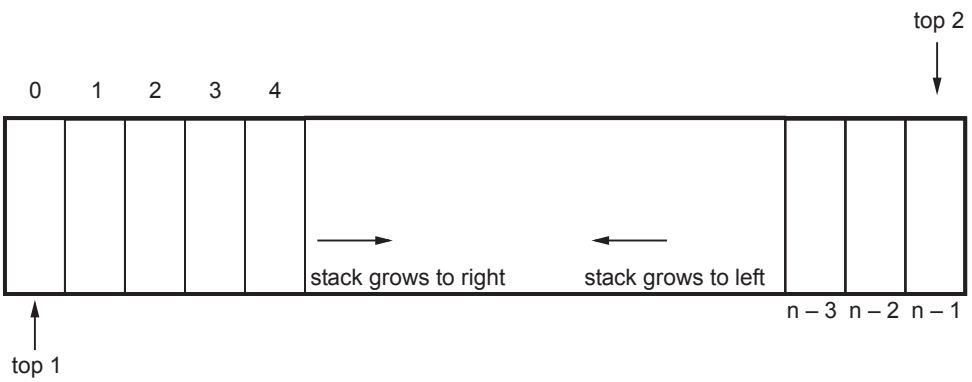


Fig. 5.5.2 Two stacks in single array

- One stack starts at the leftmost end of array and other at the rightmost end of array.
- The insertion in stack1 moves top1 to right while insertion in stack2 moves **top2** to left.
- When stack is full both the **top1** and **top2** positions are adjacent to each other.
- The **advantage** of this arrangement is that every location of the array can be utilized.
- The implementation routines are -

```
# define MAX 80
int stack[MAX];
int top1=–1, top2=n;
void push (int item, int stackno)
{
        if(stackno == 1)
        {
        /* pushing in stack1 */
                if(top1+1 == top2)

                        cout<<"stack1 is full";
                top1++;
                stack [top1]=item;
        }
        else
        {
                if (top2–1 == top1)
                cout<<"stack2 is full";
                top2– –
                stack[top2]=item;
        }
int pop (int stackno)
{
        int item;
        if (stackno == 1)
        {
                if (top1 == - 1)
                        cout<<"stack1 is empty";
                item = stack [top 1];
                top1++
                return(item);
        }
        else
        {
                if (top2 == MAX)
                        cout<<"stack2 is empty";
```

```
            item = stack [top2];
            top2--
            return(item);
        }
}
```

5.6 Applications of Stack

Various applications of stack are -

1. Stack is used for converting one form of expression to another.
2. Stack is also useful for evaluating the expression.
3. Stack is used for parsing the well formed parenthesis.
4. Decimal to binary conversion can be done by using stack.
5. The stack is used for reversing the string.
6. In recursive routines for storing the function calls the stack is used.

5.7 Polish Notation and Expression Conversion

- Expression is a string of operands and operators.
- Operands are some numeric values and operands are of two types - unary operators and binary operators.
- Unary operators are **+ and –**.
- Binary operators are **+, –, /, *, % and exponential.**

There are three types of expressions :

1. Infix Expression :

In this type of expressions the arrangement of operands and operator is as follows :

Infix expression = operand1 operator operand2

For example 1. (a + b) 2. (a + b) * (c – d) 3. (a + b/e) * (d + f)

Parenthesis can be used in these expressions. Infix expression are the most natural way of representing the expressions.

2. Postfix Expression :

In this type of expressions the arrangement of operands and operator is as follows :

Postfix expression = operand1 operand2 operator

For example 1. *a*b+ 2. *a*b + cd – * 3. *a*b + e / df + *

In postfix expression there is no parenthesis used. All the corresponding operands come first and then operator can be placed.

3. Prefix Expression :

In prefix expression the arrangement of operands and operators is as follows

Prefix expression = operator operand1 operand2

For example 1. + *a*b 2. * + *a*b – cd 3. * / + *a*be + df

This notation is also called as **Polish Notation**.

In prefix expression, there is no parenthesis used. All the corresponding operators come first and then operands are arranged.

Conversion of Infix to Postfix

Algorithm

1. Read the infix expression for left to right one character at a time.
2. Initially push $ onto the stack. For $ in stack set priority as – 1.

If input symbol read is '(' then push it on to the stack.

3. If the input symbol read is an operand then place it in postfix expression.
4. If the input symbol read is operator then

   a) Check if priority of operator in stack is greater than the priority of incoming (or input read) operator then pop that operator from stack and place it in the postfix expression. Repeat step 4(a) till we get the operator in the stack which has greater priority than the incoming operator.

   b) Otherwise push the operator being read, onto the stack.

   c) If we read input operator as ')' then pop all the operators until we get "(" and append popped operators to postfix expression. Finally just pop "(".

5. Final pop the remaining contents, from the stack until stack becomes empty append them to postfix expression.
6. Print the postfix expression as a result.

The priorities of different operators when they are in stack will be called as **instack** priorities. And the priorities of different operator when then are from input will be called as **incoming priorities**.

These priorities are as given below

Operator	Instack priority	Incoming priority
+ or –	2	1
* or /	4	3

∧ for any exponential operator	5	6
(	0	9
Operand	8	7
)	NA	0

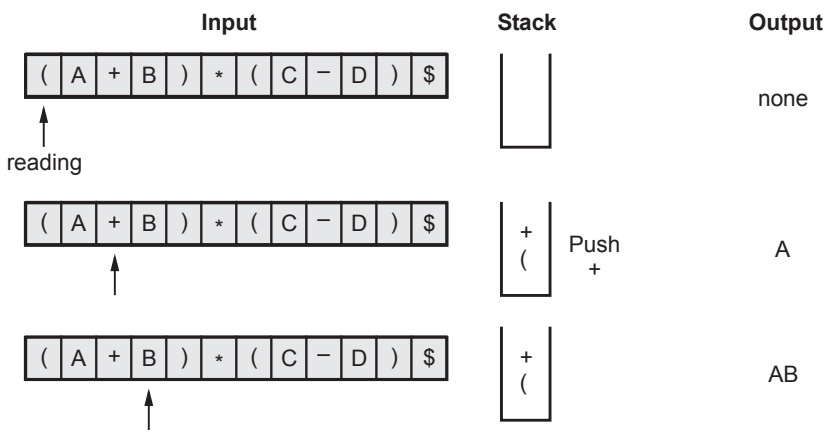
Example 5.7.1 *Convert A*B+C $ postfix form.*

Solution :

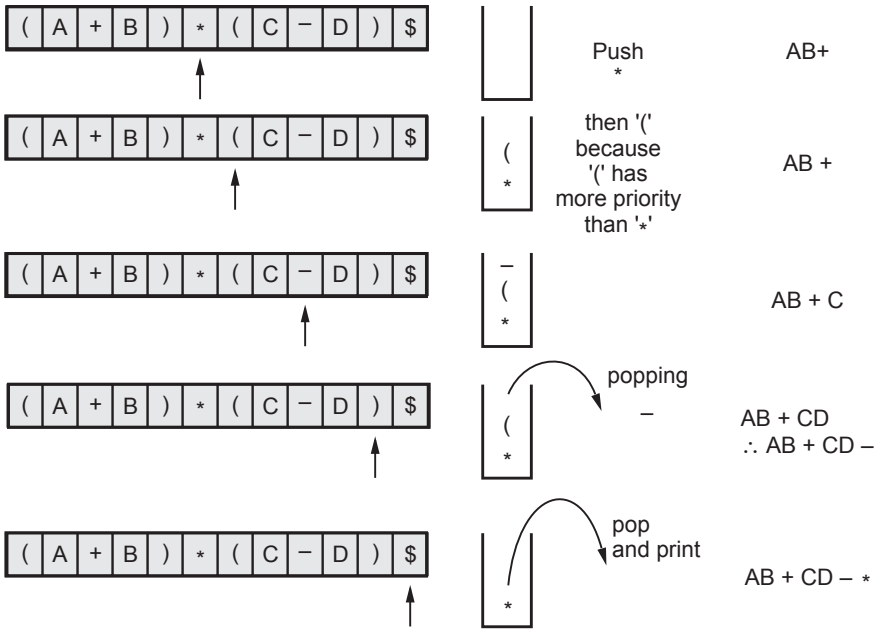
Input	Stack	Output
none	empty	none
A	empty	A
*	*	A
B	*	AB
+	pop * and Push	AB*
C	+	AB*C
$	Empty	AB*C +

Example 5.7.2 *Convert (A+B)* (C–D) to postfix.*

Solution :



When closing parenthesis comes pop everything and print except '('



So finally the converted expression is

AB+CD–*

C++ Program

```
/*************************************************************
 Program for conversion of Infix expression to Postfix form.
*************************************************************/
#include<iostream.h>
#include<conio.h>
class EXP
{
 private:
 char post[40];
 int top,st[20];
 public:
 EXP();
 void postfix(char inf[40]);
 void push(int);
 char pop();
};
EXP::EXP()
```

```
{
 top=0;
}
/*
-------------------------------------------------------------
Postfix function
-------------------------------------------------------------
*/
void EXP::postfix(char inf[40])
 {
        int i,j=0;
        for(i=0;inf[i]!='\0';i++)
        {
        switch(inf[i])
        {
                case '+' :   while(st[top] >= 1)
                             post[j++] = pop();
                             push(1);
                             break;

               case '–' :    while(st[top] >= 1)
                             post[j++] = pop();
                             push(2);
                             break;

                case '*' :   while(st[top] >= 3)
                             post[j++] = pop();
                             push(3);
                             break;

                case '/' :   while(st[top] >= 3)
                             post[j++] = pop();
                             push(4);
                             break;

                case '^' :   while(st[top] >= 4)
                             post[j++] = pop();
                             push(5);
                             break;

                case '(' :   push(0);
                             break;

                case ')' :   while(st[top] != 0)
                             post[j++] = pop();
                             top– –;
                             break;
```

```
                default  : post[j++] = inf[i];
        }
        }
        while(top>0)
        post[j++] = pop();
        cout<<"\n\tPostfix expression is =>\n\n\t\t "<<post;
 }
/*
-------------------------------------------------------------
Push function
-------------------------------------------------------------
*/
void EXP::push(int ele)
 {
        top++;
        st[top] = ele;
 }
/*
-------------------------------------------------------------
pop Function
-------------------------------------------------------------
*/
char EXP::pop()
 {
        int el;
        char e;
        el = st[top];
        top– –;
        switch(el)
        {
                case 1 : e = '+';
                                break;
                case 2 : e = '–';
                                break;
                case 3 : e = '*';
                                break;
                case 4 : e = '/';
                                break;
                case 5 : e = '^';
                                break;
        }
        return(e);
 }

/*
  The main Function
```

```
  Input:none
  Output:none
  Called By:O.S.
  Calls:postfix

*/
void main(void)
{
        EXP obj;
        char inf[40];
        clrscr();
        cout<<"\n\tEnter the infix expression :: \n";
        cin>>inf;
        cout<<"The infix expression entered by you is..."<<inf;
        obj.postfix(inf);
        getch();
 }
```

Output

```
Enter the infix expression ::
(a+b)*(c–d)
The infox expression entered by you is...(a+b)*(c–d)
        Postfix expression is =>

                        ab+cd–*
```

Example 5.7.3 *Convert the following expression into postfix. Show all steps : (a + (b * c/d) – e).*

Solution :

Step	Input	**Action**	Stack	Postfix expression
1.	(	Push (	(	
2.	a	Print a	(	a
3.	+	Push +	(+	a
4.	(	Push (	(+ (	a
5.	b	Print b	(+ (	ab
6.	*	Push *	(+ (*	ab
7.	c	Print c	(+ (*	abc
8.	/	Pop * print it then push / onto the stack	(+ (/	abc*
9.	d	Print d	(+ (/	abc*d

10.	)	Pop all the contents of the stack until (. Print all the popped contents. Then pop (.	(+	abc*d /
11.	–	pop + , print Then push –	(–	abc*d /+
12.	e	Print e		abc*d /+e
13.	)	pop all the contents until (, print the popped contents. Then pop)		abc*d /+e –

The postfix expression is **abc*d/+ e –**

Example 5.7.4 *Identify the expressions and convert them into remaining two forms :*

i) AB + C∗DE−FG+$, where $-exponent

ii) −A/B∗C$DE

Solution : i) AB + C * DE – FG + $ → postfix expression

i) Infix :

AB + C * DE – FG + $

(A + B) C * DE – FG + $

((A + B)* C) DE – RG + $

((A + B)* C) D – E FG + $

((A + B)* C) (D – E) (F + G) $ ⇒ ((A + B) * C) (D – E) $ (F + G)

ii) Prefix :

= AB + C * DE – FG + $

= (+ AB) C * DE – FG + $

= (* + ABC) DE – FG + $

= (* + ABC) (– DE) (+ FG) $

= *** + ABC $ – DE + FG**

ii) – A/B * C $ DE prefix

a) To postfix

Reverse the prefix expression

ED$C* B/A –

DE $ * C /B – A

reverse it

A – B/C *DE $ → postfix

b) To infix

= (A – B) / (C * D) $ E

Example 5.7.5 *Give the postfix and prefix expression - (a+b*c)/(x+y/z)*

Solution : Infix to postfix conversion

Input read	Action	Stack	Output
(	Push (	(	
a	Print a	(	a
+	Push +	(+	a
b	Print b	(+	ab
*	Push	(+ *	ab
c	Print c	(+ *	abc
)	Pop *, Print. Pop +, Print Pop (		abc * +
/	Push /	/	abc * +
(	Push (	/ (	abc * +
x	Print x	/ (	abc * + x
+	Push +	/ (+	abc * + x
y	Print y	/ (+	abc * + xy
/	Push /	/ (+ /	abc * + xy
z	Print z	/ (+ /	abc * + xyz
)	Pop /, Print. Pop +, Print Pop (	/	abc * + xyz / +
end of input	Pop /, Print		**abc * + xyz / + /** is required postfix expression.

Infix to Prefix Conversion

Reverse the input as) z/y + x (/) c * b + a (and then read each character one at a time.

Input read	Action	Stack	Output
)	Push)	)	
z	Print z	)	z
/	Push	) /	z
y	Print y	) /	zy
+	Pop /, Print push +	) +	zy /
x	Print x	) +	zy / x
(	Pop +, Print + Pop)		zy / x +
/	Push /	/	zy / x +
)	Push	/)	zy / x +
c	Print c	/)	zy / x + c
*	Push *	/) *	zy / x + c
b	Print b	/) *	zy / x + c
+	Pop *, Print it push +	/)	zy / x + cb *
a	Print	/) +	zy / x + cb * a
(	Pop +, Print + Pop)	/	zy/ x + cb * a +
end of input	Pop/, Print it	empty	zy / x + cb * a + /
	Reverse the output and print it.		**/ + a * bc + x / yz** is prefix expression.

Example 5.7.6 *Convert the following expression into postfix form. Show all the steps and stack content :*

*4$2*3 – 3+8/4(1+1)*

Solution :

Input symbol	Action	Stack	Postfix expression
4	Print 4	empty	4
$	Push $	$	4
2	Print 2	$	42

*	POP $, Print it. Push *	*	42 $
3	Print 3	*	42 $ 3
–	POP *, Print it Push –	-	42 $ 3*
3	Print it	-	42 $ 3*3
+	POP –, Print it. Then push +	+	42 $ 3*3 –
8	Print 8	+	42 $ 3*3 – 8
/	Push /	+ /	42 $ 3*3 – 8
4	Print 4	+ /	42 $ 3*3 – 84
/	POP/, Print it Then Push/	+ /	42 $ 3*3 – 84 /
(	Push (	+ / (	42 $ 3*3 – 84 /
1	Print 1	+ / (	42 $ 3*3 – 84 / 1
+	Push +	+ / (+	42 $ 3*3 – 84 / 1
1	Print 1	+ / (+	42 $ 3*3 – 84 / 11
)	POP +, Print it POP (	+ /	42 $ 3*3 – 84 / 11 +
end of input	POP/, Print POP +, Print	empty	42 $ 3*3 – 84 / 11 + / +

The postfix expression is **42$3*3 – 84/11+/+**

5.7.1 Need for Prefix and Postfix Expressions

- Prefix expression is a kind of expression in which the operator is placed before the two operands. The postfix expression is kind of expression in which the operator is placed after the two operands. The advantage of this arrangement is that the operators are **no longer ambiguous** with respect to the operands that they work on.
- The **order of operations** within prefix and postfix expressions is completely determined by the position of the operator and nothing else.
- Hence during compilation of an expression, the infix expression is converted to postfix form first and then it is evaluated.

5.8 Postfix Expression Evaluation

Algorithm for evaluation of postfix

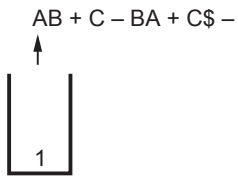
1. Read the postfix expression from left to right.
2. If the input symbol read is an operand then push it on to the stack.
3. If the operator is read POP two operands and perform arithmetic operations if operator is
   - + then result = operand 1 + operand 2
   - – then result = operand 1 – operand 2
   - * then result = operand 1 * operand 2
   - / then result = operand 1 / operand 2
4. Push the result onto the stack.
5. Repeat steps 1-4 till the postfix expression is not over.

For example - Consider postfix expression -

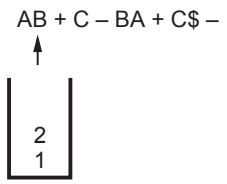
AB + C – BA + C $ – for A = 1, B = 2 and C = 3

Now as per the algorithm of postfix expression evaluation, we scan the input from left to right. If operand comes we push them onto the stack and if we read any operator, we must pop two operands and perform the operation using that operator. Here $ is taken as exponential operator.

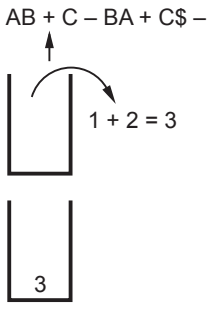
As A = 1, push 1



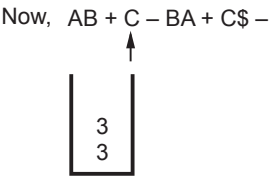
As B = 2, push 2



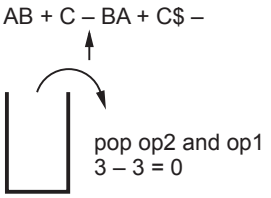
POP two operands and perform addition of 1 and 2. Then push the result onto the stack.



As C = 3, push 3 onto the stack.

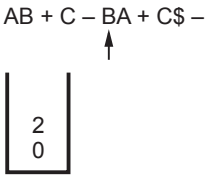


Perform subtraction

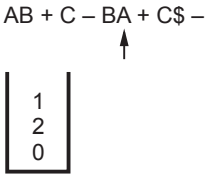


Then push the result onto the stack

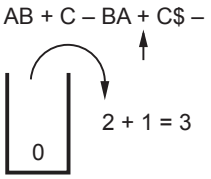
Push B = 2



Push A = 1

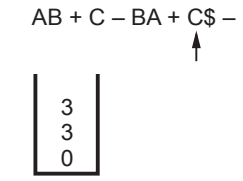


As operator comes perform operation by popping two operands.

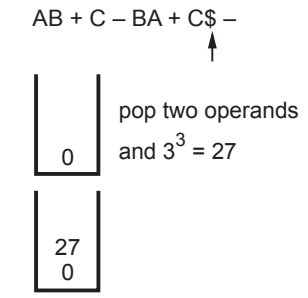


Then push the result onto the stack.

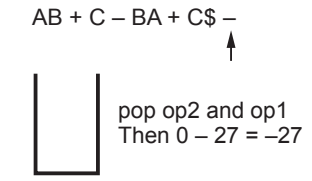
Push C = 3 onto the stack.



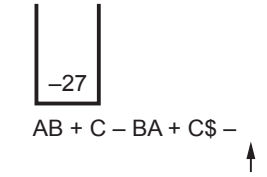
Then push the computed result



Perform the subtraction



Now since there is no further input we should pop the contents of stack and print it as a result of evaluation.



Hence output will be – 27.

C++ Program

```
/*************************************************************
      Program to evaluate a given postfix expression.
*************************************************************/
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#define size 80
```

```
class EVAL
{
/*declaration of stack data structure*/
            private:
            struct stack
            {
                  double s[size];
                  int  top;
            }st;
            public:
                  double post(char post[]);
                  void push(double);
                  double pop();
};
/*-------------------------------------------------------------------------------
 The post function which is for evaluating postfix expression
 Input : A post Expression of single digit operand
 Output: Resultant value after evaluating the expression
 Parameter Passing Method : By reference
 Called By : main()
 Calls : push(), pop()
-------------------------------------------------------------------------------
*/
double EVAL::post(char exp[])
{
      char   ch,*type;
      double  result, val, op1, op2;
      int i;
      st.top = 0;
      i=0;
      ch = exp[i];
      while ( ch !='$' )
      {
            if ( ch >= '0' && ch <= '9')
            type ="operand";

            else if ( ch == '+' || ch == '-'||
             ch == '*'|| ch == '/' ||
             ch == '^' )
            type="operator";
            if( strcmp(type,"operand")==0)/*if the character is operand*/
            {
            val = ch – 48;
            push(val);
            }
            else
```

The characters '0', '1', ... '9' will be converted to their values, so that they will perform arithmetic operation.

```
            if (strcmp(type,"operator")==0)/*if it is operator*/
            {
             op2 = pop();
             op1 = pop(); //popping two operands to perform arithmetic operation
             switch(ch)
            {
            case '+' : result = op1 + op2;
                        break;
            case '–' : result = op1 – op2;
                        break;
            case '*' : result = op1 * op2;
                       break;
            case '/' : result = op1 / op2;
                       break;
            case '^' : result = pow(op1,op2);
                       break;
      }/* switch */
      push(result);
      }
            i++;
            ch=exp[i];
      } /* while */
      result = pop(); /*pop the result*/
      return(result);
}
/*
---------------------------------------------------------------------
      The push function
      Input : A value to be pushed on global stack
      Output: None, modifies global stack and its top
      Parameter Passing Method : By Value
      Called By : post()
      Calls : none
---------------------------------------------------------------------
*/
void EVAL::push(double val)
{
      if ( st.top+1 >= size )
                  cout<<"\nStack is Full\n";
      st.top++;
      st.s[st.top] = val;
}
/*---------------------------------------------------------------------
      The pop function
      Input : None, uses global stack and top
      Output: Returns the value on top of stack
```

Finally result will be pushed onto the stack.

```
        Parameter Passing Method : None
        Called By : post()
        Calls : None
-------------------------------------------------------------------
*/
double EVAL::pop()
{
        double val;
        if ( st.top == –1 )
                cout<<"\nStack is Empty\n";
        val = st.s[st.top];
        st.top – –;
        return(val);
}
/*-------------------------------------------------------------------
        The main function
        Input : None
        Output: None
        Parameter Passing Method :None
        Called By : OS
        Calls : post()
-------------------------------------------------------------------
*/
void main ( )
{
        char exp[size];
        int len;
        double  Result;
        EVAL obj;
        clrscr();
        cout<<"Enter the postfix Expression\n";
        cin>>exp;
        len = strlen(exp);
        exp[len] = '$'; /* Append $ at the end as a endmarker*/
        Result =obj.post(exp);
        cout<<"The Value of the expression is  " <<Result;
        getch();
        exit(0);
}
```

Output

```
Enter the postfix Expression
12+34*+
The Value of the expression is  15
```

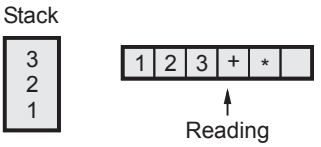
Logic of evaluation of postfix expression [Refer the above program]

Let us take some example of postfix expression and try to evaluate it

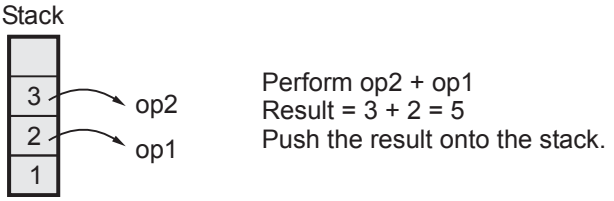
123+*

Step 1 : Assume the array exp [ ] contains the input

The $ symbol is used as an end marker. Read from first element of the array if it is **operand** push it onto the stack.

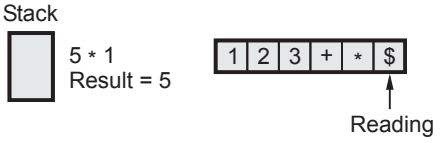


Step 2 : If the operator is read pop two operands



Step 3 : Again pop two operands and perform the operation.

Step 4 : At this point the stack is empty and the input is read as $. So here stop the evaluation procedure and return the result = 5.



5.9 Linked Stack and Operations

- The advantage of implementing stack using linked list is that we need not have to worry about the size of the stack.
- Since we are using linked list as many elements we want to insert those many nodes can be created. And the nodes are dynamically getting created so there won't be any stack full condition.
- The typical structure for linked stack can be

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

- Each node consists of data and the next field. Such a node will be inserted in the stack. Following figure represents stack using linked list.

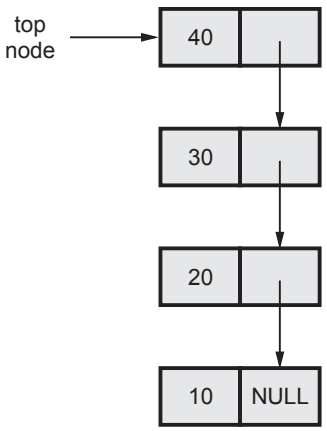


Fig. 5.9.1 Representing the linked stack

Let us now see the 'C++' implementation of it.

C++ Program

```
/*************************************************************
Program for creating the stack using the linked list. Program performs all the operations
such as push,pop and display. It takes care of stack underflow condition. There can not
be stack full condition in stack using linked list.
        *************************************************************/
        #include <iostream.h>
        #include <conio.h>
        #include<stdlib.h>
        //Declaration for data structure of linked stack
        class Lstack
```

```
{
 private:
 typedef struct stack
 {
        int data;
        struct stack *next;
 }node;
 node *top;
 public:
 Lstack();
 ~Lstack();
 void create(),remove(),show();
 void Push(int , node **);
 void Display(node **);
 int Pop(node **);
 int Sempty(node *);
};
/*
-------------------------------------------------------------
The constructor defined
-------------------------------------------------------------
*/
Lstack::Lstack()
{
  top = NULL;
}
/*
-------------------------------------------------------------
The destructor defined
-------------------------------------------------------------
*/
Lstack::~Lstack()
{
 node *temp;
 temp=top;
 if(temp==NULL)
 delete temp;
 else
 {
   while(temp!=NULL)
  {
        temp=temp–>next;
        top=NULL;
        top=temp;
  }
  delete temp;//de allocating memory
 }
```

```
}
/*
-------------------------------------------------------
The create function
-------------------------------------------------------
*/
void Lstack::create()
{
  int data;
  cout<<"\n Enter the data";
  cin>>data;
  Push(data,&top);
}
/*
-------------------------------------------------------
The remove function
-------------------------------------------------------
*/
void Lstack::remove()
{
 int item;
 if(Sempty(top))
        cout<<"\n stack underflow!";
 else
   {
        item = Pop(&top);
        cout<<"\n The popped node is  "<<item;
  }
}
/*
-------------------------------------------------------
The show function
-------------------------------------------------------
*/
void Lstack::show()
{
        Display(&top);
}

/*
-------------------------------------------------------
The Push function
-------------------------------------------------------
*/
void Lstack::Push(int Item, node **top)
{
        node *New;
```

```
        New = new node;
        New–>data=Item;
        New –> next = *top;
     *top = New;
}
/*
-------------------------------------------------------
The Sempty Function
-------------------------------------------------------
*/
int Lstack::Sempty(node *temp)
{
 if(temp==NULL)
  return 1;
 else
  return 0;
}
/*
-------------------------------------------------------
        The Pop function
-------------------------------------------------------
*/
int Lstack::Pop(node **top)
{
        int     item;
        node *temp;
        item = (*top) –>data;
        temp = *top;
        *top = (*top) –> next;
        delete temp;
        return(item);
}
/*
-------------------------------------------------
        The Display function
-------------------------------------------------
*/
void Lstack::Display(node **head )
{
        node *temp ;
        temp = *head;
        if(Sempty(temp))
         cout<<"\n The stack is empty!";
        else
        {
         while ( temp != NULL )
         {
```

```
            cout<<"  "<<temp-> data;
            temp = temp -> next;
           }
          }
          getch();
}
/*
-------------------------------------------------
          The main function
-------------------------------------------------
*/
void main ( )
{
          int choice;
          char ans,ch;
          Lstack st;
          clrscr();
          cout<<"\n\t\t Stack Using Linked List";
          do
          {
            cout<<"\n\n The main menu";
            cout<<"\n1.Push\n2.Pop\n3.Display\n4.Exit";
            cout<<"\n Enter Your Choice";
            cin>>choice;
            switch(choice)
            {
             case 1:st.create();
                              break;
             case 2:st.remove();
                              break;
             case 3:st.show();
                              break;
            case 4:exit(0);
           }
           cout<<"\n Do you want to continue?";
           ans =getche();
          getch();
          clrscr();
   }while(ans =='Y' ||ans =='y');
getch();
}
```

Output

```
 The main menu
1.Push
2.Pop
3.Display
4.Exit
 Enter Your Choice1

 Enter the data10

 Do you want to continue?
 The main menu
1.Push
2.Pop
3.Display


4.Exit
 Enter Your Choice1

 Enter the data20

 Do you want to continue?
 The main menu
1.Push
2.Pop
3.Display
4.Exit
 Enter Your Choice1

 Enter the data30

 Do you want to continue?
 The main menu
1.Push
2.Pop
3.Display
4.Exit
Enter Your Choice1
Enter the data40

 Do you want to continue?
 The main menu
1.Push
2.Pop
3.Display
4.Exit
 Enter Your Choice1
 Enter the data50

 Do you want to continue?
```

```
 The main menu
1.Push
2.Pop
3.Display
4.Exit
 Enter Your Choice3
  50  40  30  20  10
 Do you want to continue?

 The main menu
1.Push
2.Pop
3.Display
4.Exit
 Enter Your Choice2

 The popped node is  50
 Do you want to continue?

 The main menu
1.Push
2.Pop
3.Display
4.Exit
 Enter Your Choice4
```

Program Explanation

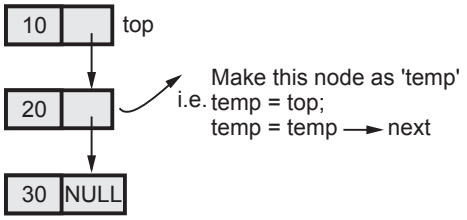
We have defined constructor for stack in which simply top is intialized by NULL value.

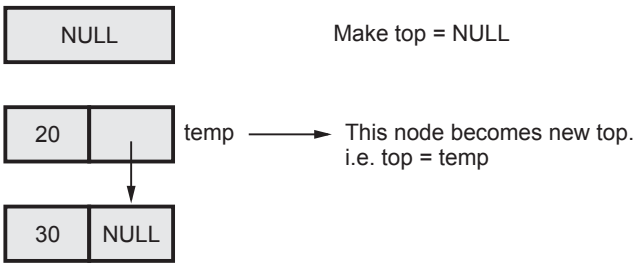
```
LStack :: LStack()
{
top = NULL ;
}
```

And in the destructor we are deallocating the memory which was reserved for each node of stack.

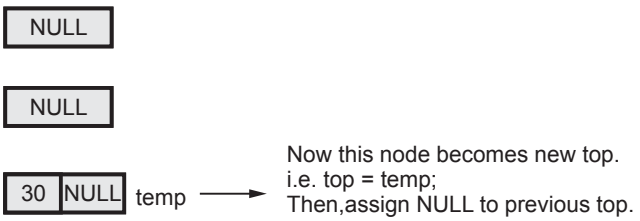
For example : If the linked stack is created as follows -

Continuing in while loop.





Hence,



This shows that all the nodes are assigned NULL. Then the memory for these nodes

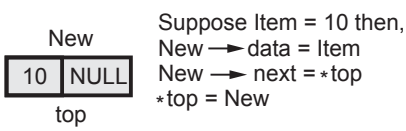
is deallocated by 'delete'.

In push function :

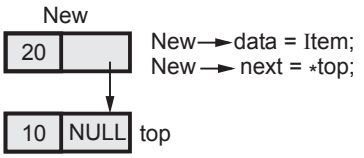
The memory for 'New ' node is allocated using new operator. In 'New' node we will put the data.

Initially top = NULL, hence

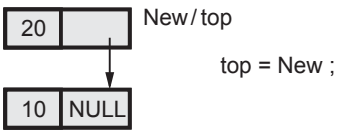
Now make 'New' node as 'top' node. When we push next item as 20 then,



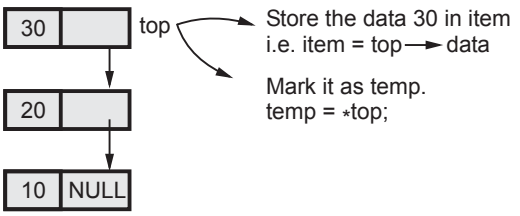
Now make 'New' node as 'top' node.



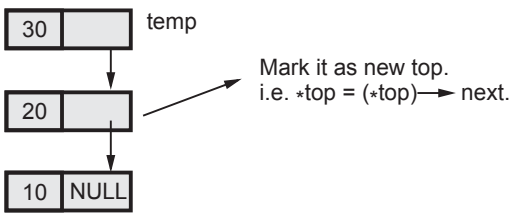
In POP function



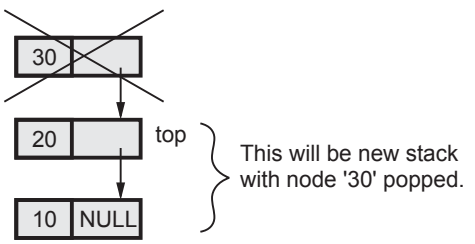
We will POP the element which is at the top. Consider a scenario that we have a stack of



Then



Then delete 'temp' node.



Review Question

1. *Explain the concept of linked stack with suitable example.*

5.10 Recursion- Concept

Definition : Recursion is a programming technique in which the function calls itself repeatedly for some input.

By recursion the same task can be performed repeatedly.

Properties of Recursion

Following are two fundamental principles of recursion

1. There must be at least one condition in recursive function which do not involve the call to recursive routine. This condition is called a "way out" of the sequence of recursive calls. The is called **base case property.**
2. The invoking of **each recursive call must reduce** to some manipulation and must go closer to base case condition.

For example - While computing factorial using recursive method -

```
if(n==0)
  return 1;   //This is a base case
else
  return n*fact(n-1);//on each call the computation will go closer to base case
```

Example

Algorithm for Factorial Function using Iterative Definition

```
1. prod = 1;
2. x=n;
3. while(x>0)
4. {
5. prod = prod*x;
6. x – –;
7. }
8. return(prod);
```

Algorithm for Factorial Function using Recursive Definition

```
1. if(n==0)
2. fact =1;
3. else
   {
4. x=n–1;
5. y= value of x!;
6. fact = n* y ;
7. } /*else ends here */
```

This definition is called the **recursive definition** of the factorial function. This definition is called recursive because again and again the same procedure of multiplication is followed but with the different input and result is again multiplied with the next input. Let us see how the recursive definition of the factorial function is used to evaluate the 5!

Step 1. 5! = 5 * 4!

Step 2. 4! = 4 * 3!

Step 3. 3! = 3 * 2!

Step 4. 1! = 1 * 0!

Step 5. 2! = 2 * 1!

Step 6. 0! = 1.

Actually the step 6 is the only step which is giving the direct result. So to solve 5! we have to backtrack from step 6 to step 1,collecting the result from each step. Let us see how to do this.

Step 6′. 0! =1

Step 5′. 1! = 1*0! = 1 from step 6′

Step 4′. 2! = 2*1! = 2 from step 5′

Step 3′. 3! = 3*2! = 6 from step 4′

Step 2′. 4! = 4*3! = 24 from step 3′

Step 1′. 5! = 5*4! = 120 from step 2′

Example 5.10.1 *Generate n^{th} term Fibonacci sequence with recursion.*

Solution :

```
#include<iostream>
using namespace std;
```

```
int fib(int n)
{
        int x,y;
        if(n<=1)
                return n;
        x=fib(n-1);
        y=fib(n-2);
        return (x+y);
}
int main(void)
{
        int n,num;
        cout<<"\n Enter location in fibonacci series: ";
        cin>>n;
        num=fib(n);
        cout<<"\n The number at "<<n<<" position is "<<num<<" in fibonacci series";
        return 0;
}
```

Output

```
Enter location in fibonacci series: 3
The number at 3 position is 2 in fibonacci series
```

5.10.1 Use of Stack in Recursive Functions

For illustrating the use of stack in recursive functions, we will consider an example of finding factorial. The recursive routine for obtaining factorial is as given below -

```
int fact (int num)
{
        int a, b;
        if (num = = 0)
                return 1 ;
        else
        {
                a = num – 1 ;
                b = fact (a) ;
                f = a * b;
                return f ;
        }
}
/* Call to the function */
ans = fact (4) ;
```

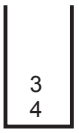
For the above recursive routine we will see the use of stack. For the stack the only principle idea is that – when there is a call to a function the values are pushed onto the stack and at the end of the function or at the return the stack is popped

Suppose we have num = 4. Then in function **fact** as num ! = 0 else part will be executed. We get

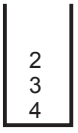
a = num – 1

a = 3

And a recursive call to fact (3) will be given. Thus internal stack stores



Next fact (2) will be invoked. Then

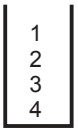


∵ a = num – 1

a = 3 – 1

b = fact (2)

Then,



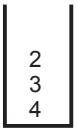
∵ a = num – 1

a = 2 – 1

b = fact (1)

Now next as num = 0, we return the value 1.

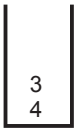
∴ a = 1, b = 1, we return f = a ∗ b = 1 from function **fact**. Then 1 will be popped off.



Now the top of the stack is the value of a

Then we get b = fact (2) = 1

∴ f = a ∗ b = 2 ∗ 1 = 2 will be returned.



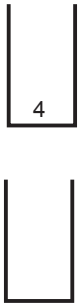
Here a = 3

Then b = fact (3)

b = 2

∴ f = a ∗ b = 6 will be returned.

The stack will be



Here a = 4

Then b = fact (4)

= 6

∴ f = a * b = 24 will be returned.

As now stack is empty. We return f = 24 from the function **fact**.

5.10.2 Variants of Recursion

Variants of recursion tells us different ways by which the recursive calls can be made depending upon the problem.

Let us understand them with the help of suitable examples -

5.10.2.1 Direct Recursion

Definition : A C function is directly recursive if it contains an explicit call to itself.

For example

```
int fun1(int n)
{
      if(n<=0)
            return n;
      else
            return fun1(n-1);
}
```

Call to itself

5.10.2.2 Indirect Recursion

Definition : A C function is indirectly recursive if function1 calls function2 and function2 ultimately calls function1.

For example

```
int fun1(int n)
{
      if(n<=0)
            return n;
      else
            return fun2(n);
}
int fun2(int m)
{
      return fun1(n-1);
}
```

5.10.2.3 Tail Recursion

Tail recursion is a kind of recursion in which there is no pending operation after returning from the recursive function.This concept was introduced by David H. D. Warren. This is a special kind of recursive approach which helps in improving the space and time efficiency of recursive function.

For example : Consider a simple example of computing factorial of number n. Following is a simple recursive function.

C program

```
/*********************************************************************
Implementing the FACTORIAL function without using tail recursion
*********************************************************************/
#include<stdio.h>
#include<conio.h>
void main()
{
     int fact(int n);
     int n;
     printf("\n Enter some number n ");
     scanf("%d",&n);
     printf("\n The factorial = %d",fact(n));
     getch();
}
int fact(int n)
{
       if(n==1)
            return 1;
       return n*fact(n-1);
}
```

Output

```
Enter some number n 4
```

The above given program does not have tail recursion, because it contains a recursive function fact and on return of this recursive call multiplication is performed. The multiplication operation is always a pending operation even after return of recursive call. We can convert the above code into a tail recursive form as follows -

```
/*********************************************************************
Implementation of tail recursion for computing FACTORIAL function
*********************************************************************/
#include<stdio.h>
#include<conio.h>
void main()
```

```
{
        int fact(int n);
        int n;
        printf("\n\t Program for finding factorial(n) using Tail Recursion ");
        printf("\n Enter some number n ");
        scanf("%d",&n);
        printf("\n The factorial = %d",fact(n));
        getch();
}
int fact(int n)
{
        int my_new_fact(int n,int f);
        return my_new_fact(n,1);
}
int my_new_fact(int n,int f)
{
        if(n==1)
                return f;
        my_new_fact(n-1,n*f);
}
```

This is tail recursive function as after returning from this function there is no pending operation.

Output

```
        Program for finding factorial(n) using Tail Recursion
 Enter some number n 4

 The factorial = 24
```

Note that once the control returns from the function my_new_fact, there is no pending operation. The control always returns from this function with the final result in variable f.

Advantages of tail recursion

1. **Optimizes** the task of compiler for executing the recursive function.
2. Improves the **space and time efficiency** of the recursive code.

5.10.2.4 Tree

A tree recursion is a kind of recursion in which the recursive function contains the pending operation that involves another recursive call to the function.

The implementation of **fibonnaci** series is a classic example of tree recursion.

C Function

```
int fib(int n)
{
        if(n==0)
                return 0;
        if(n==1)
                return 1;
        return fib(n-1)+fib(n-2);
}
```

```
                fib(4)
        =       fib(3) + fib(2)
        =       [fib(2) + fib(1)] + [fib(1) + fib(0)]
        =       [(fib(1) + fib(0)] + [1] + [1 + 0]
        =       [1+0] + 1 + 1
fib(4)  =       3
```

The call tree can be represented as follows

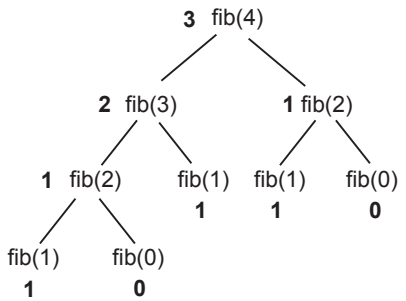


Fig. 5.10.1 Recursive tree

5.10.2.5 Difference between Recursion and Iteration

Sr. No.	Iteration	Recursion
1.	Iteration is a process of executing certain set of instructions repeatedly, without calling the self function.	Recursion is a process of executing certain set of instructions repeatedly by calling the self -function repeatedly.
2.	The iterative functions are implemented with the help of for, while, do-while programming constructs.	Instead of making use of for, while, do-while the repetition in code execution is obtained by calling the same function again and again over some condition.
3.	The iterative methods are more efficient because of better execution speed.	The recursive methods are less efficient.
4.	Memory utilization by iteration is less.	Memory utilization is more in recursive functions.
5.	It is simple to implement.	Recursive methods are complex to implement.
6.	The lines of code is more when we use iteration.	Recursive methods bring compactness in the program.

5.11 Backtracking Algorithmic Strategy

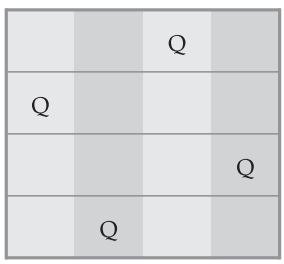
- Backtracking is a method in which
  1. The desired solution is expressible as an n tuple $(x_1, x_2,\ldots x_n)$ where x_i is chosen from some finite set S_i.
  2. The solution maximizes or minimizes or satisfies a criterion function C $(x_1, x_2,\ldots x_n)$.
- The basic idea of backtracking is to build up a **vector,** one component at a time and to test whether the vector being formed has any chance of success.
- The major **advantage** of backtracking algorithm is that we can realize the fact that the partial vector generated does not lead to an optimal solution. In such a situation that vector can be ignored.
- Backtracking algorithm determines the solution by systematically searching the solution space (i.e. set of all feasible solutions) for the given problem.

For example

Example : n-Queen's problem - The n-queen's problem can be stated as follows. " Consider a chessboard of order $n \times n$. The problem is to place n queens on this board such that no two queens can attack each other. That means no two queens can be placed on the same row, column or diagonal. "

The solution to n-queens problem can be obtained using backtracking method.

The solution can be given as below -



⇐ Note that no two queens can attack each other.

5.11.1 Some Terminologies used in Backtracking

Backtracking algorithms determine problem solutions by systematically searching for the solutions using **tree structure.**

For example -

Consider a 4-queen's problem. It could be stated as "there are 4 queens that can be placed on 4×4 chessboard. Then no two queens can attack each other".

Following Fig. 5.11.1 shows tree organization for this problem.

- Each node in the tree is called **a problem state.**

- All paths from the root to other nodes define the **state space of** the problem.
- The solution states are those problem states s for which the path from root to s defines a **tuple** in the solution space.
  In some trees the leaves define the **solution states.**
- **Answer states :** These are the leaf nodes which correspond to an element in the set of solutions, these are the states which satisfy the implicit constraints.

For example- Refer Fig. 5.11.1 (See Fig. 5.11.1 on next page)

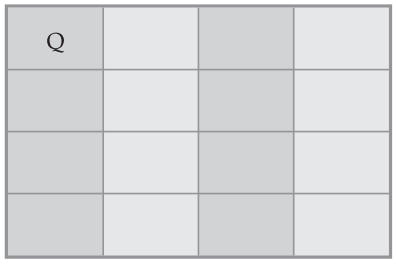
- A node which is been generated and all whose children have not yet been generated is called **live node.**
- The live node whose children are currently being expanded is called **E-node.**
- A **dead node** is a generated node which is not to be expanded further or all of whose children have been generated.

5.11.2 The 4 Queen's Problem

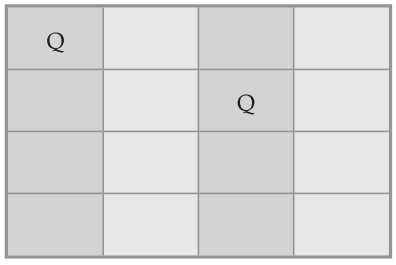
Let us take 4-queens and 4×4 chessboard.

- Now we start with empty chessboard.

Place queen 1 in the first possible position of its row i.e. on 1^{st} row and 1^{st} column.



- Then place queen 2 after trying unsuccessful place - 1(1, 2), (2, 1), (2, 2) at (2, 3) i.e. 2^{nd} row and 3^{rd} column.



- This is the dead end because a 3^{rd} queen cannot be placed in next column, as there is no acceptable position for queen 3. Hence algorithm **backtracks** and places the 2^{nd} queen at (2, 4) position.

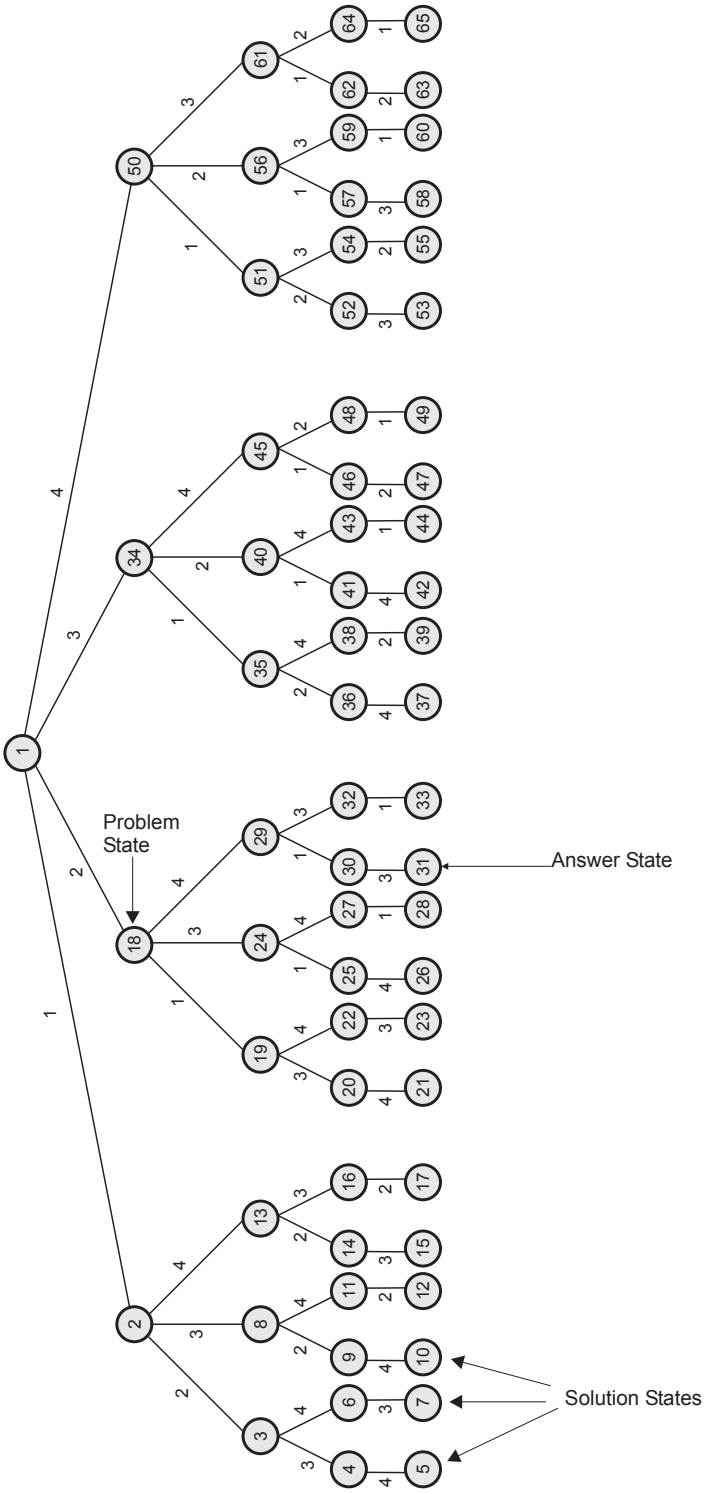
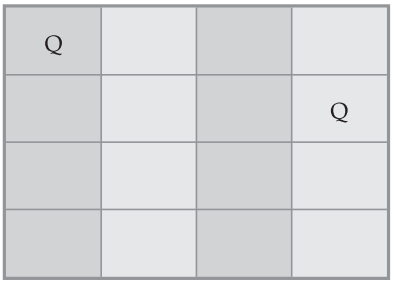
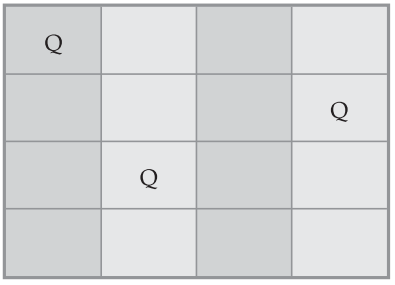


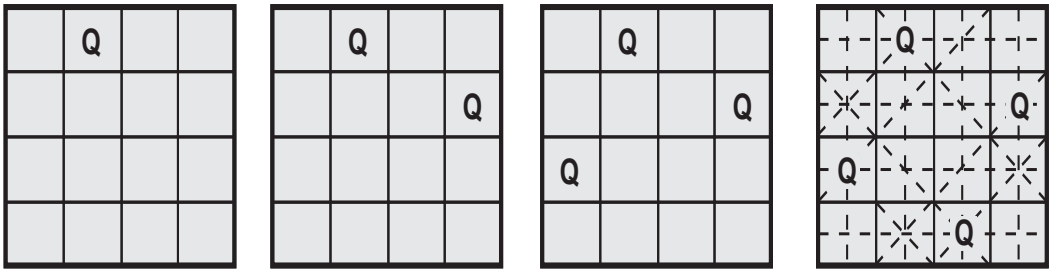
Fig. 5.11.1



- The place 3rd queen at (3, 2) but it is again another dead end as next queen (4th queen) cannot be placed at permissible position.



- Hence we need to backtrack all the way upto queen 1 and move it to (1, 2).
- Place queen 1 at (1, 2), queen 2 at (2, 4), queen 3 at (3, 1) and queen 4 at (4, 3).



Thus solution is obtained.
(2, 4, 1, 3) in rowwise manner.

The state space tree of 4-queen's problem is shown in Fig. 5.11.2.
(See Fig. 5.11.2 on next page)

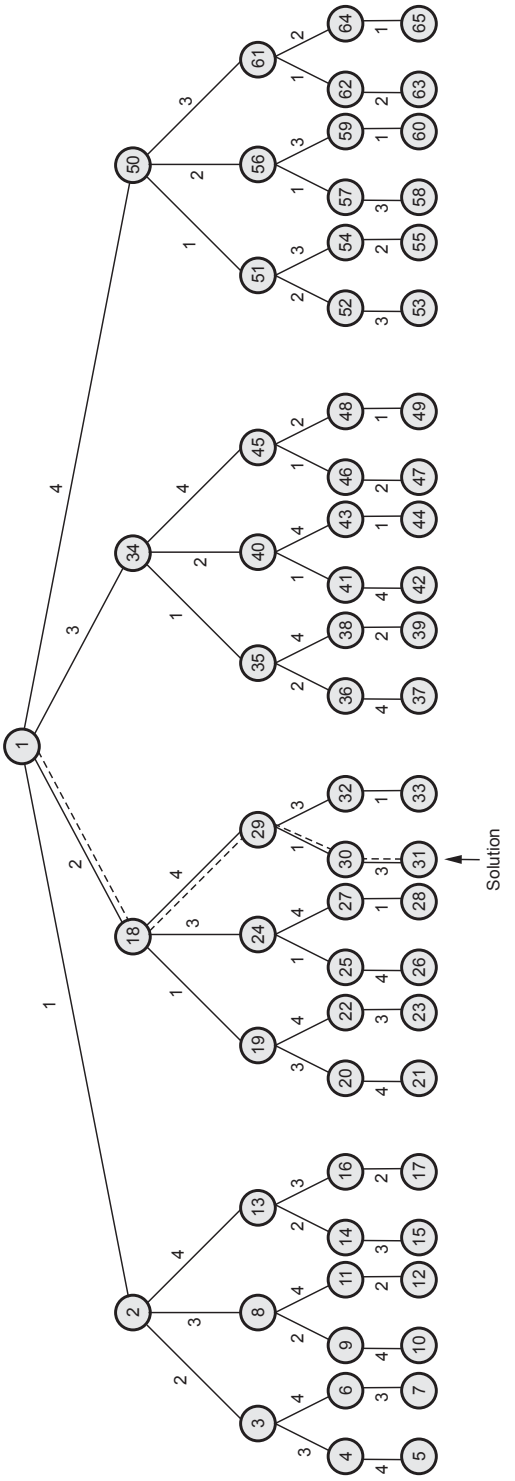


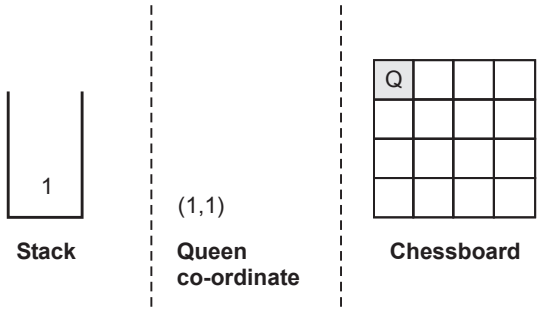
Fig. 5.11.2 State space tree for 4-queen's problem

5.12 Use of Stack in Backtracking

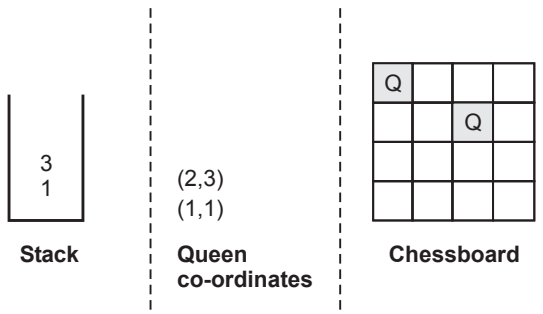
The stack can be used in backtracking to handle the recursive procedures.

Example 5.12.1 *Let us consider, 4 Queen's problem once again to understand the role of stack in backtracking.*

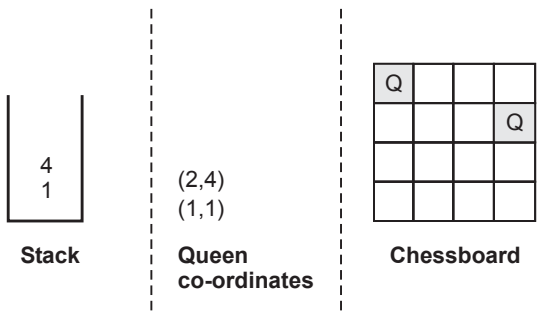
Solution : Step 1 : Place queen 1 on 1st row 1st column push only column number onto the stack.



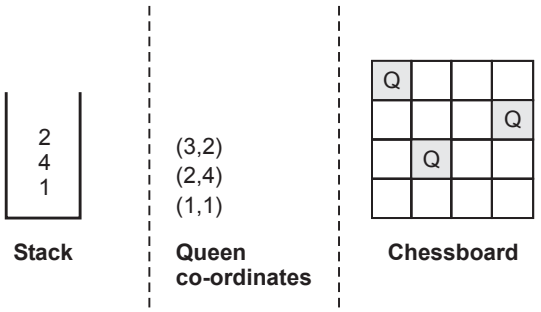
Step 2 : Then place 2 at 2nd row, 3rd column.



Step 3 : This is dead end, because 3rd queen can not be placed in next column as there is no acceptable poistion for queen 3. Hence algorithm backtrack by popping the 2nd queen's position. Now let us place queen 2 at 4th column.

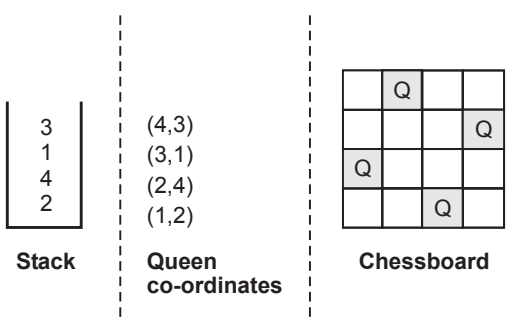


Step 4 : Place 3rd Queen at 2nd column.



Step 5 : Now, there is no valid place for queen 4. So we needed to backtrack all the moves by popping the column index from the stack to try out other possible places.

Step 6 : Finally the possible solution can be.



Algorithm :

1. Start with a queen from first row, first column and search for valid position.
2. If we find a valid position in current row, push the position (i.e. Column number) onto the stack. Then start again on next row.
3. If we don't find a valid position in the current row then we backtrack to previous row i.e. POP the column position for previous row from the stack and search for a valid position.
4. When the stack size is equal to n (for n queens) then that means we have placed n queens on the board. This is a solution to n queen's problem.

❑❑❑

Unit - VI

6 Queue

Syllabus

Basic concept, Queue as Abstract Data Type, Representation of Queue using Sequential organization, Queue Operations, Circular Queue and its advantages, Multi-queues, Linked Queue and Operations. Deque-Basic concept, types (Input restricted and Output restricted), Priority Queue- Basic concept, types(Ascending and Descending).

Contents

6.1 Basic Concept

Definition : The queue can be formally defined as ordered collection of elements that has two ends named as **front** and **rear**. From the front end one can delete the elements and from the rear end one can insert the elements.

For example :

The typical example can be a queue of people who are waiting for a city bus at the bus stop. Any new person is joining at one end of the queue, you can call it as the rear end. When the bus arrives the person at the other end first enters in the bus. You can call it as the front end of the queue.

Following Fig. 6.1.1 represents the queue of few elements.

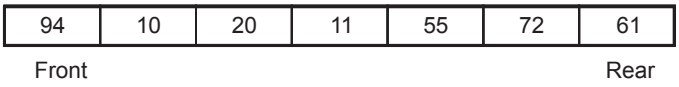


Fig. 6.1.1 Queue

6.1.1 Comparison between Stack and Queue

Sr. No.	Stack	Queue
1.	The stack is a LIFO data structure. That is the element which is inserted last will be removed first from the stack.	The queue is a FIFO data structure. That means the element which is inserted first will be removed first.
2.	The insertion and deletion of the elements in the stack is done from only one end, called top.	The insertion of the element in the queue is done by the end called rear and the deletion of the element from the queue is done by the end called front.

Review Question

1. *Compare stacks and queues.*

6.2 Queue as Abstract Data Type

The ADT for queue is as given below -

AbstractDataType Queue

{

Instances :

Que[MaX] is a finite collection of elements in which insertion of element is by rear end and deletion of element is by front end.

Precondition :

The front and rear should be within the maximum size MAX.

Before insertion operation , whether the queue is full or not is checked.

Before any deletion operation, whether the queue is empty or not is checked.

Operations :

1. Create() - The queue is created by declaring the data structure for it.
2. insert() - The element can be inserted in the queue by rear end.
3. delet() - The element at front end deleted each time.
4. Display() - The elements of queue are displayed from front to rear.

}

Review Question

1. *Explain term : Queue ADT.*

6.3 Representation of Queue using Sequential Organization

- As we have seen, queue is nothing but the collection of items.
- Both the ends of the queue are having their own functionality.
- The Queue is also called as FIFO i.e. a First In First Out data structure. All the elements in the queue are stored sequentially.
- Various operations on the queue are -
  1. Queue overflow.
  2. Insertion of the element into the queue.
  3. Queue underflow.
  4. Deletion of the element from the queue.
  5. Display of the queue.

Let us see each operation one by one

```
'C++' representation of queue.
struct queue
{
        int que [size];
        int front;
        int rear;
} Q;
```

6.4 Queue Operations

1. Insertion of element into the queue

The insertion of any element in the queue will always take place from the rear end.

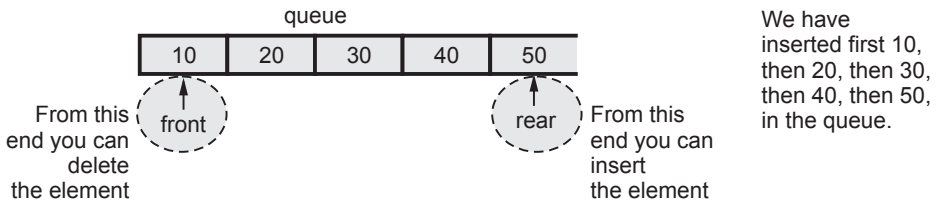


Fig. 6.4.1 Representing the insertion

Before performing insert operation you must check whether the queue is full or not. If the rear pointer is going beyond the maximum size of the queue then the queue overflow occurs.

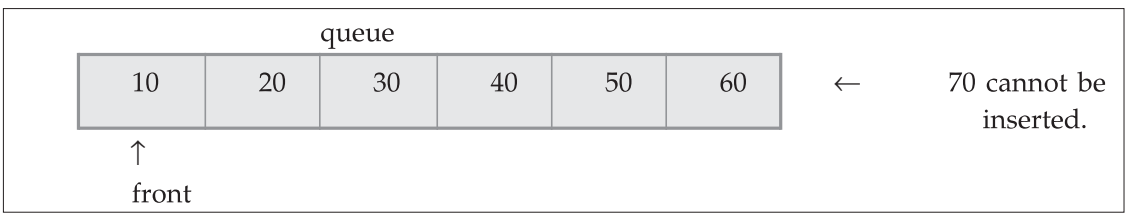


Fig. 6.4.2 Representing the queue overflow

2. Deletion of element from the queue

The deletion of any element in the queue takes place by the front end always.

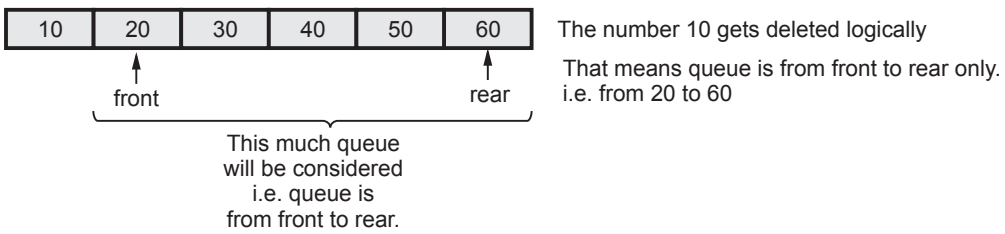


Fig. 6.4.3 Representing the deletion

Before performing any delete operation one must check whether the queue is empty or not. If the queue is empty, you can not perform the deletion. The result of illegal attempt to delete an element from the empty queue is called the queue underflow condition.

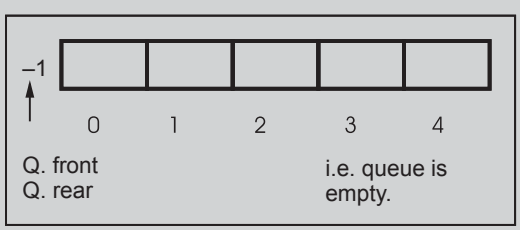
Fig. 6.4.4 Representing the queue underflow

Let us see the C++ implementation of the queue.

C++ Program

```
/*************************************************************
      Program for implementing the Queue using arrays
*************************************************************/
#include<iostream.h>
#include<stdlib.h>
#include<conio.h>
#define size 5
class MyQ
{
private:
struct queue
{
 int que[size];
 int front,rear;
}Q;
public:
  MyQ();
  int Qfull();
  int insert(int);
  int Qempty();
  int delet();
  void display();
};
MyQ::MyQ()
{
 Q.front = –1;
 Q.rear  = –1;
}
/*
 The Qfull  Function
 Input:none
```

Queue data structure declared with array que [ ], front and rear

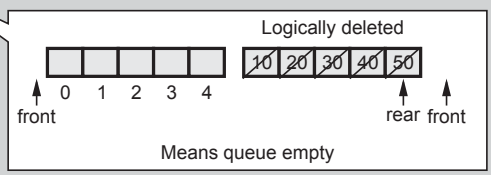


```
 Output:1 or 0 for q full or not
 Called By:main

*/
int MyQ::Qfull()
 {
 if(Q.rear >=size–1)
  return 1;
 else
  return 0;
}
/*
 The insert  Function
 Input:item -which is to be inserted in the Q
 Output:rear value
 Called By:main
 Calls:none
*/
int MyQ::insert(int item)
{
  if(Q.front ==–1)
  Q.front++;
  Q.que[++Q.rear] = item;
  return Q.rear;
}
int MyQ::Qempty()


{
 if((Q.front == – 1) || (Q.front > Q.rear))
 return 1;
 else
 return 0;
}


/*
 The delete Function
 Input:none
 Output:front value
 Called By:main
 Calls:none
*/
int MyQ::delet()

{
 int item;
```

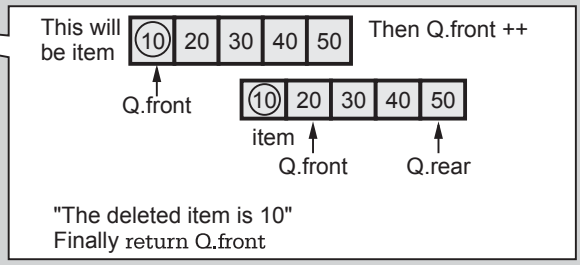
If Queue exceeds the maximum size of the array then it returns1 - means queue full is true otherwise 0 means queue full is false

This condition will occur initially when queue is empty

Always increment the rear pointer and place the element in the queue.



```
 item = Q.que[Q.front];
 Q.front++;
 cout<<"\n The deleted item is "<<item;
 return Q.front;
}
/*
 The display Function
 Input:none
 Output:none
 Called By:main
 Calls:none
*/
void MyQ::display()
{
  int i;
  for(i=Q.front;i<=Q.rear;i++)   <------Printing the queue from front to rear
    cout<<" "<<Q.que[i];
}
void main(void)
{
 int choice,item;
 char ans;
 MyQ obj;
 clrscr();
 do
 {
   cout<<"\n Main Menu";
   cout<<"\n1.Insert\n2.Delete\n3.Display";
   cout<<"\n Enter Your Choice: ";
   cin>>choice;
   switch(choice)
        {
        case 1:if(obj.Qfull())       //checking for Queue overflow
                        cout<<"\n Can not insert the element";
                else
                {
                        cout<<"\n Enter The number to be inserted ";
                        cin>>item;
                        obj.insert(item);
                }
                break;
        case 2:if(obj.Qempty())
                         cout<<"\n Queue Underflow!!";
                         else
                         obj.delet();
                         break;
        case 3:if(obj.Qempty())
```



```
                    cout<<"\nQueue Is Empty!";
                    else
                    obj.display();
                    break;
        default:cout<<"\n Wrong choice!";
                    break;
        }
  cout<<"\n Do You Want to continue?";
  ans =getche();
  }while(ans =='Y'||ans =='y');
}
/********************** End Of Program **********************/
```

Output

```
 Main Menu
1.Insert
2.Delete
3.Display
 Enter Your Choice: 1

 Enter The number to be inserted 10

 Do You Want to continue?y
 Main Menu
1.Insert
2.Delete
3.Display
 Enter Your Choice: 1

 Enter The number to be inserted 20

 Do You Want to continue?y
 Main Menu
1.Insert
2.Delete
3.Display
 Enter Your Choice: 1

 Enter The number to be inserted 30

 Do You Want to continue?y
 Main Menu
1.Insert
2.Delete
3.Display
 Enter Your Choice: 1
```

```
Enter The number to be inserted 40

Do You Want to continue?y
Main Menu
1.Insert
2.Delete
3.Display
Enter Your Choice: 3
10 20 30 40
Do You Want to continue?y
Main Menu
1.Insert
2.Delete
3.Display
Enter Your Choice: 2

The deleted item is 10
Do You Want to continue?y
Main Menu
1.Insert
2.Delete
3.Display
Enter Your Choice: 2

The deleted item is 20
Do You Want to continue?y
Main Menu
1.Insert
2.Delete
3.Display
Enter Your Choice: 3
30 40
Do You Want to continue?
```

Example 6.4.1 *Show how to implement a queue using two stacks ? Analyse the running time of the queue operations.*

Solution : There are two stacks namely stack1, stack2. Following steps can be performed for **insertq** and **deleteq** operations to implement queue using two stacks.

```
insertq (item)
{
        Step 1 : push everything from stack1 to stack2 while stack1 is not empty.
        Step 2 : push item to stack1.
        Step 3 : Push all elements from stack2 to stack1.
}
deleteq ()
{
```

```
    Step 1 : pop an item from stack1 and return it.
}
```

Now we will analyze time for queue operations

insertq operation

- Step 1 will require **O(n)** time.
- Step 2 will 'require O(1).
- Step 3 will require O(n).

Thus total O(n) + O(1) + O(n) = O(n) time will be required by insert operation of queue.

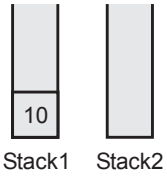
deleteq operation : Step 1 will require O(1) time.

Consider elements 10, 20, 30, 40 to be inserted in queue. The steps are as follows -

Step 1 : Insertq(10)

As there is nothing in stack1 initially hence nothing will be pushed to stack1.

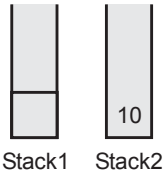
push 10 to stack1



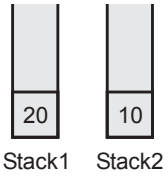
Nothing will be copied to stack1. From stack2 as stack2 is empty. If deleteq operation is to be performed then top element of stack1 will be returned.

Step 2 : Insertq(20)

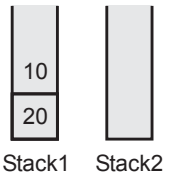
push everything from stack1 to stack2



push 20 to stack1

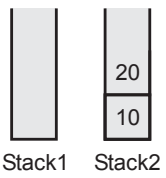


pop everything from stack2 and push it to stack1

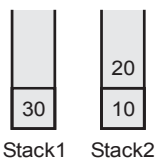


Step 3 : Insertq (30)

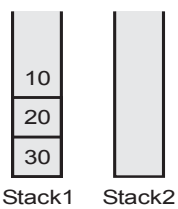
push everything from stack1 to stack2



push 30 to stack1.



pop from stack2 and push it onto stack1

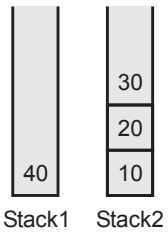


If deleteq operation is to be performed then pop from stack1.

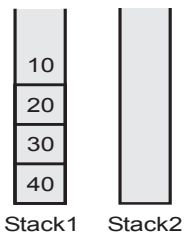
Step 4 : Insertq (40)

push everything from stack1 to stack2

push 40 onto stack1



push everything from stack2 to stack1



If deleteq operation is to be performed the pop from stack1.

Example 6.4.2 *Show how to implement a stack using two Queues ? Analyse the run time of the stack operations ?*

Solution : We will push the elements 10, 20, 30, 40, 50 onto the stack. During the push operations one can invoke pop at any time. Note that these push and pop operations can be performed only with the help of two queues Q1 and Q2.

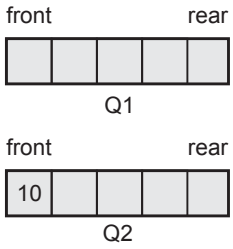
Following steps will be followed during push or pop operations.

```
push (item)
{
Step 1 : Insert item in Q2
Step 2 : One by one delete everything from Q1 and insert them to Q2.
Step 3 : Swap names of Q1 and Q2.
That is change name of Q2 as Q1 and Q1 as Q2.
}
pop ( )
{
Delete an item from Q1
and return it.
}
```

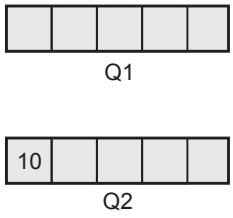
Now let us use above steps.

Step 1 : push 10

Insert 10 to Q2



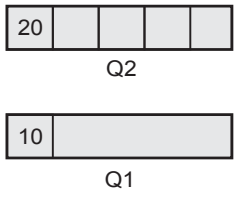
As there is nothing in Q1 no element will be copied to Q2. Now make Q2 as Q1 and Q1 as Q2.



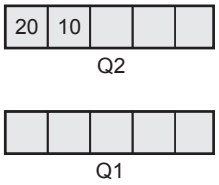
If pop operation needs to be performed then simply delete element from Q1 from front end.

Step 2 : push 20

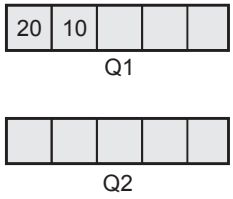
Insert 20 to Q2.



Copy all elements from Q1 to Q2.



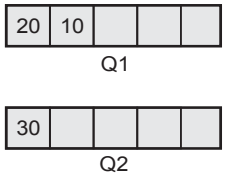
Now make Q2 as Q1 and Q1 as Q2.



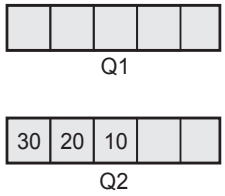
If pop operation needs to be performed then simply delete element from Q1.

Step 3 : push 30.

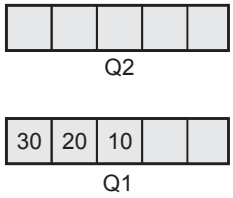
Insert 30 to Q2.



Copy all the elements from Q1 to Q2.



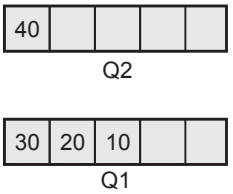
Make Q2 as Q1 and Q1 as Q2.



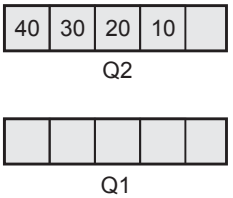
If pop operation needs to be performed then, simply delete element from Q1.

Step 4 : push (40)

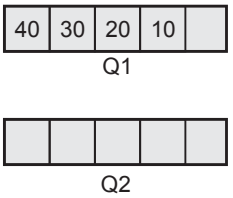
Insert 40 to Q2



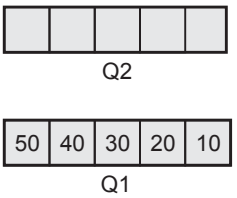
Copy all the elements from Q1 to Q2



Now exchange the names of queues.



If pop operation needs to be performed then simply delete element from Q1. In the similar manner if push (50) operation can be implemented and finally -



Review Question

1. *Explain about the operations of queue with an example.*

6.5 Circular Queue

As we have seen, in case of linear queue the elements get deleted logically. This can be shown by following Fig. 6.5.1.

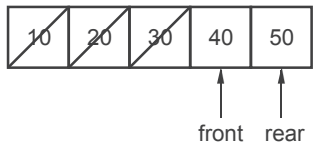


Fig. 6.5.1 Linear queue

We have deleted the elements 10, 20 and 30 means simply the front pointer is shifted ahead. We will consider a queue from front to rear always. And now if we try to insert any more element then it won't be possible as it is going to give "queue full !" message. Although there is a space of elements 10, 20 and 30 (these are deleted elements), we can not utilize them because queue is nothing but a linear array !

Hence there is a concept called circular queue. The main advantage of circular queue is we can utilize the space of the queue fully. The circular queue is shown by following Fig. 6.5.2.

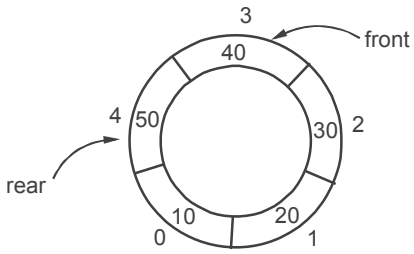


Fig. 6.5.2 Circular queue

Considering that the elements deleted are 10, 20 and 30.

There is a formula which has to be applied for setting the front and rear pointers, for a circular queue.

```
rear = (rear + 1) % size
front = (front + 1) % size
```

rear = (rear + 1) % size

= (4 + 1) % 5

rear = 0

So we can store the element 60 at 0^{th} location similarly while deleting the element.

front = (front + 1) % size

= (3 + 1) % 5

front = 4

So delete the element at 4^{th} location i.e. element 50

Let us see the 'C++' program now,

C++ Program

```
#include<iostream.h>
#include<conio.h>
#define MAX 10
class Queue
{
 int Que[MAX];
 int front,rear;
 public:
 Queue()//constructor defined
 {
        front=–1;
        rear=0;
 }
 void init();
 void insert(int ch);
 int delet();
 void display();
};

void Queue::init()
{
 int i;
 for(i=0;i<MAX;i++)
        Que[i]=0;
}
/*
-------------------------------------------------------------
insert function
-------------------------------------------------------------
*/

void Queue::insert(int item)
{
 if (front==(rear+1)%MAX)
 {
        cout << "Queue is full\n";
 }
 else
 {
        //setting front pointer for a single element in Queue
        if(front==–1)
        front=rear=0;
        else
        rear=(rear+1)%MAX;
```

```
        Que[rear]=item;
 }
}
/*
-------------------------------------------------------
delet function
-------------------------------------------------------
*/
int Queue::delet()
{
 int val;
 if(front==–1)
 {
        cout << "Queue is empty\n";
        return 0; // return null on empty Queue
 }
 val= Que[front];//item to be deleted
 if(front==rear)//when single element is present
 {
        front=rear=–1;
 }
 else
         front=(front+1)%MAX;
 return val;
}
/*
-------------------------------------------------------
Display function
-------------------------------------------------------
*/
void Queue::display()
{
 int i;
 i=front;
 while(i!=rear)
 {
        cout<<Que[i]<<"  ";
        i=(i+1)%MAX;
 }
 cout<<Que[i]<<endl;
}

/*
-------------------------------------------------------
The main function
-------------------------------------------------------
*/
```

```
void main()
{
 int i,choice,item;
 char ans='y';
 clrscr();
 Queue obj;
 obj.init();
 do
 {
        cout<<"   Menu"<endl;
        cout<<"1.Insert \n 2.Delete \n 3.Display"<<endl;
        cout<<"Enter Your choice"<<endl;
        cin>>choice;
        switch(choice)
        {
         case 1:cout<<"Enter the element"<<endl;
                cin>item;
                        obj.insert(item);
                        break;
         case 2:cout<<"The element to be deleted is"<<endl;
                cout<<obj.delet();
                   break;
         case 3:obj.display();
                 break;
        }
        cout<<"Do You Want to Continue?"<<endl;
        ans=getche();
 }while(ans=='Y'||ans=='y');
 getch();
}
```

Output

```
Menu
    1.Insert
    2.Delete
    3.Display
    Enter Your choice: 1

    Enter the element
    10
    Do You Want to Continue?  y
      Menu
    1.Insert
    2.Delete
    3.Display
    Enter Your choice: 1
```

```
Enter the element
20
Do You Want to Continue?  y
   Menu
1.Insert
2.Delete
3.Display
Enter Your choice: 1

Enter the element
30
Do You Want to Continue?  y
   Menu
1.Insert
2.Delete
3.Display
Enter Your choice: 3
10  20  30
Do You Want to Continue?  y
   Menu
1.Insert
2.Delete
3.Display
Enter Your choice: 2
The element to be deleted is
10
Do You Want to Continue?  y
   Menu
1.Insert
2.Delete
3.Display
Enter Your choice: 3
20  30
Do You Want to Continue?  n
```

Example 6.5.1 *Suppose a queue is maintained by circular array QUEUE with N = 12 memory cells. Find the number of elements in the queue when :*

i) FRONT = 4, REAR = 8, ii) FRONT = 10, REAR = 3

iii) FRONT = 5, REAR = 6, iv) Delete two elements after step (iii)

Solution : The circular queue with 12 Cell is.

i)

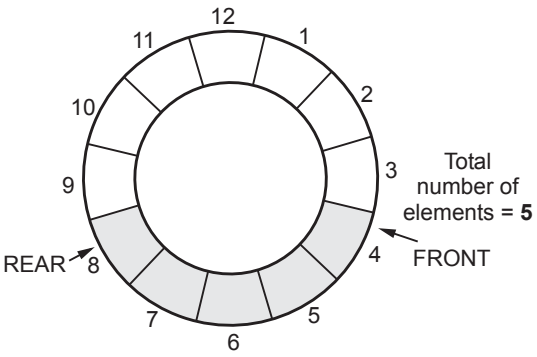


Fig. 6.5.3

ii)

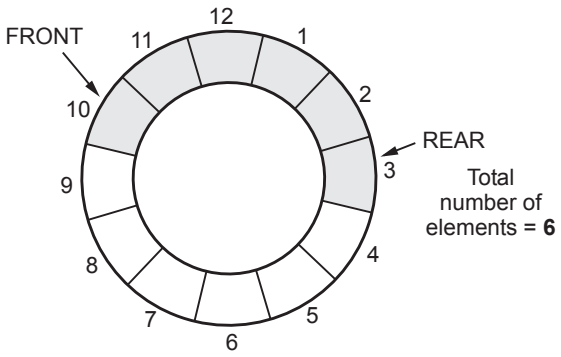


Fig. 6.5.4

iii)

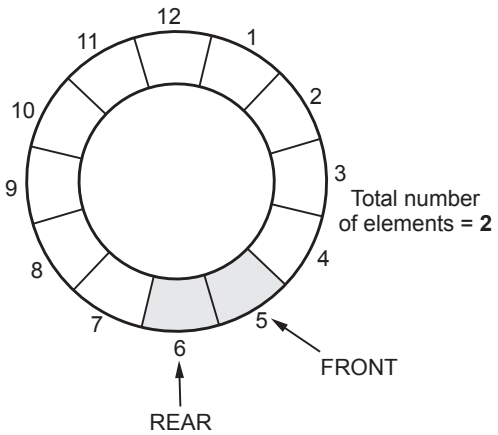


Fig. 6.5.5

iv) The circular queue after deleting 2 elements will be empty.

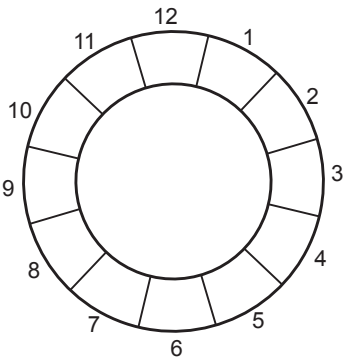


Fig. 6.5.6

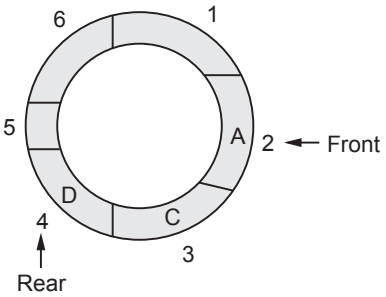
Example 6.5.2 *Consider a circular queue of characters and is of size 6. "_" denotes an empty queue location. Show the queue contents as the following opns. take place :*

i) F is added to the queue
ii) Two letters are deleted.
iii) K, L and M are added to the queue.
iv) Two letters are deleted.
v) R is added to the queue.
vi) Two letters are deleted
vii) S is added to the queue.
viii) Two letters are deleted.

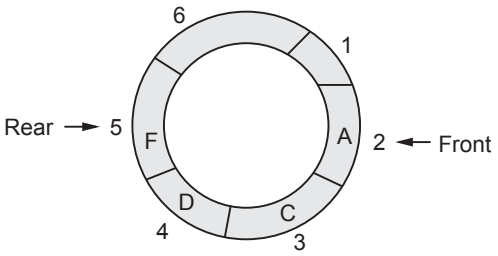
Initial queue configuration is :

FRONT = 2, REAR = 4, Queue : –, A, C, D, –, –

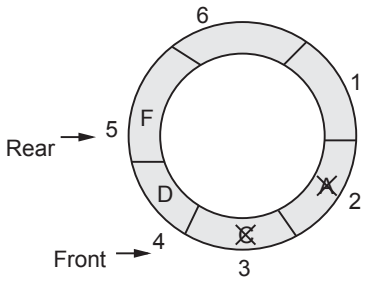
Solution : The initial configuration is,



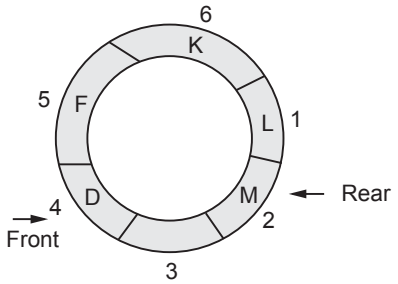
i) F is added



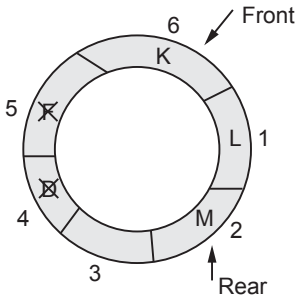
ii) Two letters are deleted



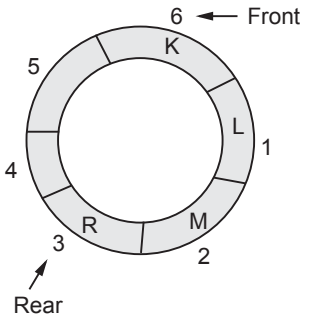
iii) K, L, M are added



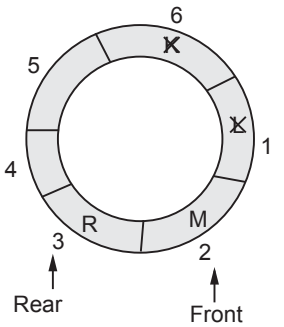
iv) Two letters are deleted



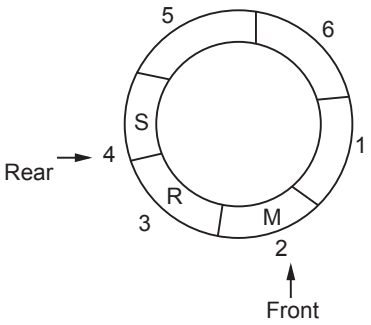
v) R is added



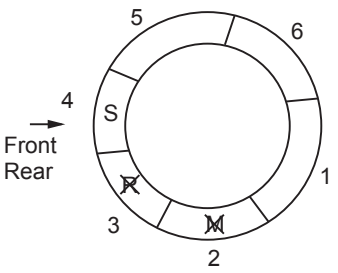
vi) Two letters are deleted

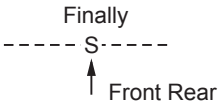


vii) S is added to the queue



viii) Two letters are deleted





Review Question

1. *What is circular queue ? Implement insert and delete operations.*

6.6 Multi-queues

- One of the application of queues is categorization of data. And multiple queues can be used to store variety of data. We can implement multiple queues using single dimensional arrays.
- In a one dimensional array, multiple queues can be placed. Insertion from its rear end and deletion from its front end can be possible for desired queue. Refer Fig. 6.6.1

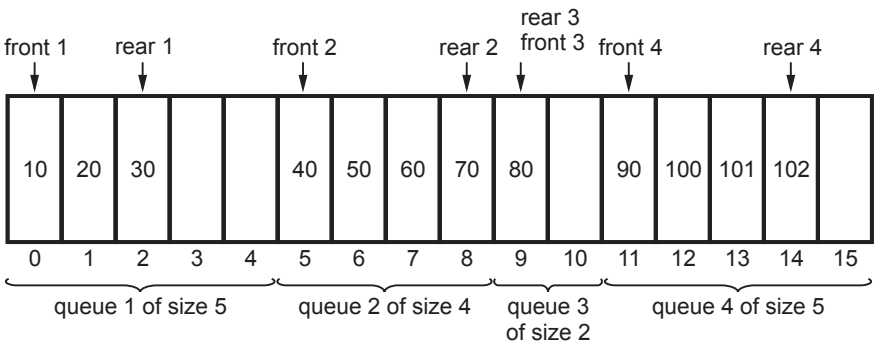


Fig. 6.6.1 Multiple queues using single array

- There are four queues having their own front and rear positioned at appropriate points in a single dimensional array.

- We can perform insertion and deletion of any element for any queue. We can declare the messages "queue full" and "queue empty" at appropriate situations for that particular queue.

C++ Program

```
/************************************************************
Program for implementing the multiple Queues using one
dimensional array.
************************************************************/
#include<iostream>
using namespace std;
#define size 20
class MultiQueue
{
        private:
        /*data structure for multiple queue*/
        struct mult_que {
                        int que[size];
                        int rear[size],front[size];
                }Q;
        public:
        int sz[size];/*stores sizes of individual queue*/
/*
 The set_que function
Purpose:This function initialises all the front and rear positions of individual queues.
All these queues has to be in a single dimensional array
*/
        void set_que(int index)
        {
                int sum,i;
                if(index==1)
                {
                        Q.front[index]=0;
                        Q.rear[index]=Q.front[index]-1;
                }
                else
                {
                        sum=0;
                        for(i=0;i<index-1;i++)
                                sum=sum+sz[i];
                        Q.front[index]=sum;
                        Q.rear[index]=Q.front[index]-1;
        /*computed the rear position based on previous queues*/
                }
        }
/* The Qfull  Function*/
        int Qfull(int index)
```

```
    {
        int sum,i;
        sum=0;
        for(i=0;i<index;i++)
            sum=sum+sz[i];
        if(Q.rear[index]==sum-1)
            return 1;
        else
            return 0;
    }
/*  The insert  Function */
    void insert(int item,int index)
    {
        int j;
        j=Q.rear[index];
        Q.que[++j] = item;
        Q.rear[index]=Q.rear[index]+1;
    }

/*  The Qempty function */
    int Qempty(int index)
    {
        if(Q.front[index]>Q.rear[index])
            return 1;
        else
            return 0;
    }
/*  The delet Function*/
    int delet(int index)
    {
        int item;
        item = Q.que[Q.front[index]];
        Q.que[Q.front[index]]=-1;
        Q.front[index]++;
        return item;
    }
/*  The display Function */
    void display(int num)
    {
        int index,i;
        index=1;
        do
        {
            if(Qempty(index))
                cout<<"\n The Queue "<<index<<" is Empty";
            else
            {
```

```
                        cout<<"\n Queue number "<<index<<" is: ";
                        for(i=Q.front[index];i<=Q.rear[index];i++)
                        {
                                if(Q.que[i]!=-1)
                                        cout<<" "<<Q.que[i];
                        }
                }
                index++;
        }while(index<=num);
    }
};
/*  The main function */
int main(void)
{
    int choice,item,num,i,index;
    char ans;
    MultiQueue MQ;
    cout<<"\n\t\t Program For Multiple Queues";
    cout<<"\n How many Queues do you want?";
    cin>>num;
    cout<<"\n Enter The size of queue(Combined size of all Queues is 20)";
    for(i=0;i<num;i++)
        cin>>MQ.sz[i];
    for(index=1;index<=num;index++)
        MQ.set_que(index);/*front and rear values of each queue are set*/
    do
    {
        cout<<"\n Main Menu";
        cout<<"\n1.Insert\n2.Delete\n3.Display";
        cout<<"\n Enter Your Choice: ";
        cin>>choice;
        switch(choice)
        {
            case 1: cout<<"\n Enter in which queue you wish to insert the item? ";
                    cin>>index;
                    if(MQ.Qfull(index))
                    /*checking for Queue overflow*/
                        cout<<"\n Can not insert the element";
                    else
                    {
                        cout<<"\n Enter The number to be inseted: ";
                        cin>>item;
                        MQ.insert(item,index);
                    }
                    break;
            case 2:cout<<"\n Enter From which queue you wish to delete the item?
";
```

```
                        cin>>index;
                        if(MQ.Qempty(index))
                                cout<<"\n Queue Underflow!!";
                        else
                        {
                                item= MQ.delet(index);
                                cout<<"\n The deleted item is: "<<item;
                        }
                        break;
                case 3:MQ.display(num);
                        break;
                default:cout<<"\n Wrong choice!";
                        break;
        }
        cout<<"\n Do You Want to continue?";
        cin>>ans;
        }while(ans =='Y'||ans =='y');
    return(0);
}
```

Output

```
Program For Multiple Queues
How many Queues do you want?4
Enter The size of queue(Combined size of all Queues is 20)
4 6 5 5
        Main Menu
1. Insert
2. Delete
3. Display
        Enter Your Choice 1
 Enter in which queue you wish to  insert the item?3
 Enter The number to be inseted10
 Do You Want to continue?y
  Main Menu
1. Insert
2. Delete
3. Display
 Enter Your Choice1
 Enter in which queue you wish to  insert the item?4
 Enter The number to be inseted20
 Do You Want to continue?y
 Main Menu
1. Insert
2. Delete
3. Display
 Enter Your Choice1
 Enter in which queue you wish to  insert the item?3
 Enter The number to be inseted30
```

```
Do You Want to continue?y
Main Menu
1. Insert
2. Delete
3. Display
Enter Your Choice3
The Queue 1 is Empty
The Queue 2 is Empty
Queue number 3 is: 10 30
Queue number 4 is: 20
Do You Want to continue?y
Main Menu
1. Insert
2. Delete
3. Display
Enter Your Choice2

Enter From which queue you wish to  delete the item?3
The deleted item is 10
Do You Want to continue?y
Main Menu
1. Insert
2. Delete
3. Display
Enter Your Choice3
The Queue 1 is Empty
The Queue 2 is Empty
Queue number 3 is: 30
Queue number 4 is: 20
Do You Want to continue?n
```

6.7 Linked Queue and Operations

As we have seen that the queue can be implemented using arrays, it is possible to implement queues using linked list also. The main **advantage** in linked representation is that we need not have to worry about size of the queue. As in linked organization we can create as many nodes as we want so there will **not be a queue full condition** at all. The queue using linked list will be very much similar to a linked list. The only difference between the two is in queue the left most node is called front node and the right most node is called rear node. And we can not remove any arbitrary node from queue. We have to remove front node always :

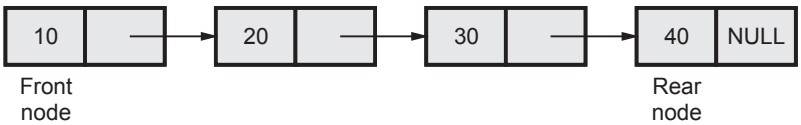


Fig. 6.7.1 Queue using linked list

The typical node structure will be

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

Let us now see the 'C++' program for it.

C++ Program

```
/*************************************************************
Program For implementing the Queue using the linked list.
The queue full condition will never occur in this program.
*************************************************************/
#include<iostream.h>
#include<stdlib.h>
#include<conio.h>
//Declaration of Linked Queue data structure
class Lqueue
{
  private:
  typedef struct node
  {
   int data;
   struct node *next;
  }Q;
  Q *front,*rear;
  public:
  Lqueue();
  ~Lqueue();
  void create(),remove(),show();
  void insert();
  Q *delet();
  void display(Q *);
};

/*
-------------------------------------------------------------
The constructor defined
-------------------------------------------------------------*/
Lqueue::Lqueue()
{
 front=NULL;
 rear= NULL;
}
/*
```

```
        ------------------------------------------------------
        The create function
        ------------------------------------------------------*/
        void Lqueue::create()
        {
        insert();
        }
        /*
        ------------------------------------------------------
        The remove function
        ------------------------------------------------------*/
        void Lqueue::remove()
        {
          front =delet();
        }
        /*
        ------------------------------------------------------
        The show function
        ------------------------------------------------------*/
        void Lqueue::show()
        {
          display(front);
}
        /*
        ------------------------------------------------------
         The insert Function
        ------------------------------------------------------
        */
        void Lqueue::insert()
        {
         char ch;
         Q *temp;
         clrscr();
         temp =new Q;//allocates memory for temp node
         temp–>next=NULL;
         cout<<"\n\n\n\tInsert the element in the Queue\n";
         cin>>temp->data;

         if(front == NULL)//creating first node
         {
           front= temp;
           rear=temp;
          }
          else                //attaching other nodes
           {
                rear–>next=temp;
                rear=rear–>next;
```

```
        }
}

/*
-------------------------------------------------------
The QEmpty Function
-------------------------------------------------------*/
int Qempty(Q *front)
{
 if(front ==NULL)
  return 1;
 else
  return 0;
}
/*
-------------------------------------------------------
The delet Function
-------------------------------------------------------
*/
Q *Lqueue::delet()
{
  Q *temp;
  temp=front;
  if(Qempty(front))
  {
        cout<<"\n\n\t\tSorry!The Queue Is Empty\n";
        cout<<"\n Can not delete the element";
  }
  else
   {
         cout<<"\n\tThe deleted Element Is  "<<temp–>data;
         front=front->next;
         temp->next=NULL;
         delete temp;
   }
 return front;
 }
/*
-------------------------------------------------------
 The display Function
-------------------------------------------------------
*/
void Lqueue::display(Q *front)
{
 if(Qempty(front))
  cout<<"\n The Queue Is Empty\n";
 else
```

```
 {
   cout<<"\n\t The Display Of Queue Is \n ";
   for(;front !=rear->next;front=front->next)
         cout<<"   "<<front->data;
 }
 getch();
}
/*
-------------------------------------------------------------
The destructor defined
-------------------------------------------------------------
*/
Lqueue::~Lqueue()
{
 if((front!=NULL)&&(rear!=NULL))
 {
  front=NULL;
  rear=NULL;
  delete front;
  delete rear;
 }
}

/*
-------------------------------------------------------------
The main Function
Calls:create,remove,show
Called By:O.S.
-------------------------------------------------------------
*/
void main(void)
{
  char ans;
  int choice;
  Lqueue Que;
  do
  {
        clrscr();
        cout<<"\n\tProgram For Queue Using Linked List\n";
        cout<<"\n            Main Menu";
        cout<<"\n1.Insert\n2.Delete \n3.Display";
        cout<<"\n Enter Your Choice";
        cin>>choice;
        switch(choice)
        {
         case 1 :Que.create();
                        break;
```

```
        case 2 :Que.remove();
                        break;
        case 3 :Que.show();
                        break;
       default: cout<<"\nYou have entered Wrong Choice"<<endl;
                        break;
  }
 cout<<"\nDo You Want To See Main Menu?(y/n)"<<endl;
 ans = getch();
}while(ans == 'y' || ans == 'Y');
getch();
}
```

Output

```
      Program For Queue Using Linked List
         Main Menu
1.Insert
2.Delete
3.Display
 Enter Your Choice 1

 Insert the element in the Queue
 10

Do You Want To See Main Menu?(y/n)
            Program For Queue Using Linked List
         Main Menu
1.Insert
2.Delete
3.Display
 Enter Your Choice 1

  Insert the element in the Queue
20

Do You Want To See Main Menu?(y/n)
      Program For Queue Using Linked List
         Main Menu
1.Insert
2.Delete
3.Display
 Enter Your Choice 1

  Insert the element in the Queue
30

Do You Want To See Main Menu?(y/n)
      Program For Queue Using Linked List
```

```
            Main Menu
1.Insert
2.Delete
3.Display
 Enter Your Choice 1

  Insert the element in the Queue
40

Do You Want To See Main Menu?(y/n)

        Program For Queue Using Linked List

            Main Menu
1.Insert
2.Delete
3.Display
 Enter Your Choice3

         The Display Of Queue Is
   10  20  30  40
Do You Want To See Main Menu?(y/n)

        Program For Queue Using Linked List

            Main Menu
1.Insert
2.Delete
3.Display
 Enter Your Choice2

        The deleted Element Is  10
Do You Want To See Main Menu?(y/n)


        Program For Queue Using Linked List

            Main Menu
1.Insert
2.Delete
3.Display
 Enter Your Choice2

        The deleted Element Is  20
Do You Want To See Main Menu?(y/n)

        Program For Queue Using Linked List
```

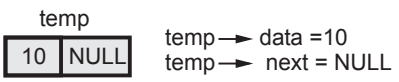
```
          Main Menu
1.Insert
2.Delete
3.Display
 Enter Your Choice3

       The Display Of Queue Is
  30  40
Do You Want To See Main Menu?(y/n)n
```

Program Explanation

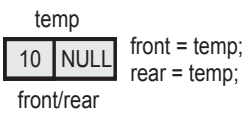
In above program we have defined a constructor Lqueue in which front and rear are initialized to 'NULL'.

In insert function

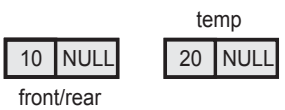
First we will allocate a node called 'temp' with next field assigned with 'Null' value. Suppose we want to insert data 10 then



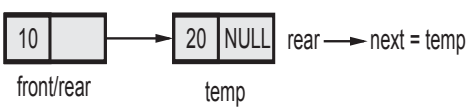
Now mark this node as queue's front and rear



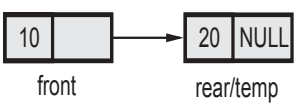
Now next, suppose we want to insert 20 in queue then a new node 'temp' will be created with data 20.



Then, the queue will be formed by setting appropriate front and rear.



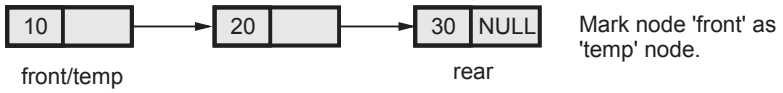
Then



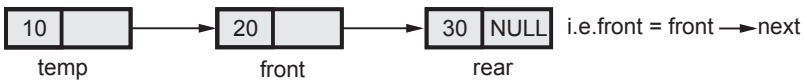
Continuing in this fashion we can create a linked queue.

In delet function

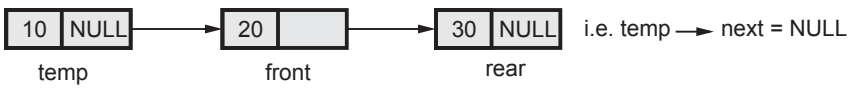
We assume that a queue is created like this -



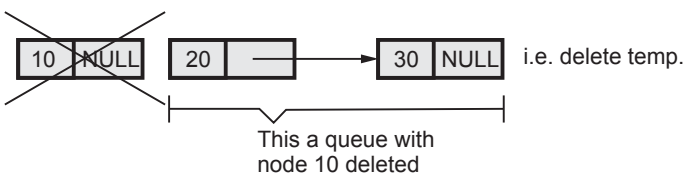
Simply display a message that "The deleted Element is 10" (i.e. temp → data). Then set front as



Then



Then,



Review Questions

1. *Define queue. Discuss about the various representations of a queue.*
2. *Discuss about various representations of queue.*

6.8 Deque

In a linear queue, the usual practice is for insertion of elements we use one end called rear and for deletion of elements we use another end called as front. But in the doubly ended queue we can make use of both the ends for insertion of the elements as well as we can use both the ends for deletion of the elements. That means it is possible to insert the elements by rear as well as by front. Similarly it is possible to delete the elements from front as well as from rear. Just see the following figure to understand the concept of doubly ended queue i.e. deque.

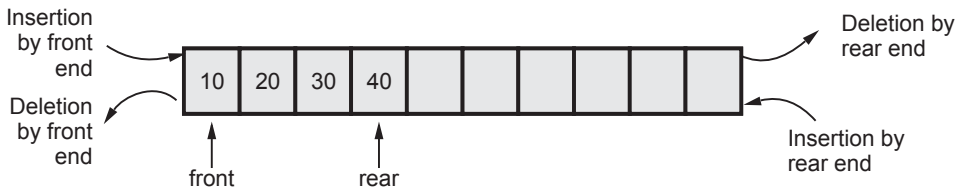
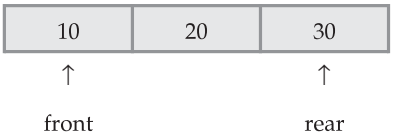


Fig. 6.8.1 Doubly ended queue

As we know, normally we insert the elements by rear end and delete the elements from front end. Let us say we have inserted the elements 10, 20, 30 by rear end.



Now if we wish to insert any element from front end then first we have to shift all the elements to the right.

For example if we want to insert 40 by front end then, the deque will be

(a) Insertion by front end

(b) Deletion by rear end

Fig. 6.8.2 Operations on deque

We can place –1 for the element which has to be deleted.

Let us see the implementation of deque using arrays

```
/*********************************************************************
Program To implement Doubly ended queue using arrays
*********************************************************************/
```

```
#include<iostream>
#include<stdlib.h>
#define size 5
using namespace std;
class Dqueue
{
   private:
         int que[size];
   public:
         int front,rear;
         Dqueue();
         int Qfull();
         int Qempty();
         int insert_rear(int item);
         int delete_front();
         int insert_front(int item);
         int delete_rear();
         void display();
};

Dqueue::Dqueue()
{
   front=-1;
   rear=-1;
   for(int i=0;i<size;i++)
         que[i]=-1;

}

int Dqueue::Qfull()
{
   if(rear==size-1)
         return 1;
   else
         return 0;
}

int Dqueue::Qempty()
{
   if((front>rear)||(front==-1&&rear==-1))
         return 1;
  else
         return 0;
}

int Dqueue::insert_rear(int item)
{
```

```
    if(front==-1&&rear==-1)
        front++;
    que[++rear]=item;
    return rear;
}

int Dqueue::delete_front()
{
    int item;
    if(front==-1)
        front++;
    item=que[front];
    que[front]=-1;
    front++;
    return item;
}

int Dqueue::insert_front(int item)
{
    int i,j;
    if(front==-1)
        front++;
    i=front-1;
    while(i>=0)
    {
        que[i+1]=que[i];
      i--;
    }
    j=rear;
    while(j>=front)
    {
        que[j+1]=que[j];
        j--;
     }
     rear++;
     que[front]=item;
     return front;
}
int Dqueue::delete_rear()
{
    int item;
    item=que[rear];
    que[rear]=-1;/*logical deletion*/
    rear--;
    return item;
}
```

```
void Dqueue::display()
{
      int i;
      cout<<"\n Straight Queue is:";
            for(i=front;i<=rear;i++)
      cout<<"  "<<que[i];
}

int main()
{
      int choice,item;
      char ans;
      ans='y';
      Dqueue obj;
      cout<<"\n\t\t Program For doubly ended queue using arrays";
      do
      {
      cout<<"\n1.insert by rear\n2.delete by front\n3.insert by front\n4.delete by rear";
            cout<<"\n5.display\n6.exit";
            cout<<"\n Enter Your choice ";
            cin>>choice;
            switch(choice)
            {
             case 1:if(obj.Qfull())
                                    cout<<"\n Doubly ended Queue is full";
                        else
                            {
                            cout<<"\n Enter The item to be inserted";
                            cin>>item;
                            obj.rear=obj.insert_rear(item);
                            }
                       break;
          case 2:if(obj.Qempty())
                        cout<<"\n Doubly ended Queue is Empty";
                       else
                       {
                       item=obj.delete_front();
                       cout<<"\n The item deleted from queue is "<<item;
                       }
                       break;
          case 3:if(obj.Qfull())
                        cout<<"\n Doubly ended Queue is full";
                       else
                       {
                        cout<<"\n Enter The item to be inserted";
                        cin>>item;
                        obj.front=obj.insert_front(item);
```

```
                    }
                    break;
        case 4:if(obj.Qempty())
                     cout<<"\n Doubly ended Queue is Empty";
                    else
                    {
                    item=obj.delete_rear();
                    cout<<"\n The item deleted from queue is "<<item;
                    }
                    break;
        case 5:obj.display();
                    break;
        case 6:exit(0);
    }
    cout<<"\n Do You Want To Continue?";
    cin>>ans;
   }while(ans=='y'||ans=='Y');
   return 0;
  }
```

Output

```
            Program For doubly ended queue using arrays
  1.insert by rear
  2.delete by front
  3.insert by front
  4.delete by rear
  5.display
  6.exit
   Enter Your choice 1

   Enter The item to be inserted 10

  Do You Want To Continue?y

 1.insert by rear
 2.delete by front
 3.insert by front
 4.delete by rear
 5.display
 6.exit
  Enter Your choice 1

  Enter The item to be inserted 20

  Do You Want To Continue?y

 1.insert by rear
 2.delete by front
```

```
3.insert by front
4.delete by rear
5.display
6.exit
 Enter Your choice 1

 Enter The item to be inserted 30
 Do You Want To Continue?y

1.insert by rear
2.delete by front
3.insert by front
4.delete by rear
5.display
6.exit
 Enter Your choice 5

 Straight Queue is:  10  20  30
 Do You Want To Continue?
```

Review Questions

1. *Define dequeue and give its example.*
2. *What is dequeue ? Write pseudo code to perform insertion and deletion of element in a linked implementation of dequeue.*

6.9 Priority Queue

The priority queue is a data structure having a collection of elements which are associated with specific ordering. There are two types of priority queues –

1. Ascending priority queue
2. Descending priority queue

Application of Priority Queue

1. The typical example of priority queue is scheduling the jobs in operating system. Typically operating system allocates priority to jobs. The jobs are placed in the queue and position1 of the job in priority queue determines their priority. In operating system there are three kinds of jobs. These are real time jobs, foreground jobs and background jobs. The operating system always schedules the real time jobs first. If there is no real time job pending then it schedules foreground jobs. Lastly if no real time or foreground jobs are pending then operating system schedules the background jobs.

2. In network communication, to manage limited bandwidth for transmission the priority queue is used.
3. In simulation modeling, to manage the discrete events the priority queue is used.

Types of Priority Queue

The elements in the priority queue have specific ordering. There are two types of priority queues –

1. **Ascending Priority Queue -** It is a collection of items in which the items can be inserted arbitarily but only smallest element can be removed.
2. **Descending Priority Queue -** It is a collection of items in which insertion of items can be in any order but only largest element can be removed.

In priority queue, the elements are arranged in any order and out of which only the smallest or largest element allowed to delete each time.

The implementation of priority queue can be done using arrays or linked list. The data structure heap is used to implement the priority queue effectively.

ADT for Priority Queue

Various operations that can be performed on priority queue are -

1. Insertion 2. Deletion 3. Display

Hence the ADT for priority queue is given as below -

Instances :

P_que[**MaX**] is a finite collection of elements associated with some priority.

Precondition :

The front and rear should be within the maximum size **MAX**.

Before insertion operation , whether the queue is full or not is checked.

Before any deletion operation, whether the queue is empty or not is checked.

Operations :

1. Create() – The queue is created by declaring the data structure for it.
2. insert() – The element can be inserted in the queue
3. delet() – If the priority queue is ascending priority queue then only smallest element is deleted each time. And if the priority queue is descending priority queue then only largest element is deleted each time.
4. Display() – The elements of queue are displayed from front to rear.

The implementation of priority queue using arrays is as given below -

1. Insertion operation

While implementing the priority queue we will apply a simple logic. That is while inserting the element we will insert the element in the array at the proper position. For example if the elements are placed in the queue as -

9	12			
que[0] front	que[1] rear	que[2]	que[3]	que[4]

And now if an element 8 is to be inserted in the queue then it will be at 0^{th} location as -

8	9	12		
que[0] front	que[1]	que[2] rear	que[3]	que[4]

If the next element comes as 11 then the queue will be -

8	9	**11**	12	
que[0] front	que[1]	que[2]	que[3] rear	que[4]

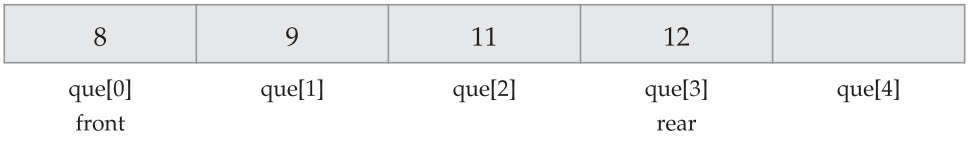
The C++ function for this operation is as given below -

```
int Pr_Q::insert(int rear,int front)
{
        int item,j;
        cout<<"\nEnter the element: ";
        cin>>item;
        if(front ==–1)
                        front++;
        j=rear;
        while(j>=0 && item<que[j])
        {
         que[j+1]=que[j];
         j--;
        }
        que[j+1]=item;
        rear=rear+1;
        return rear;
}
```

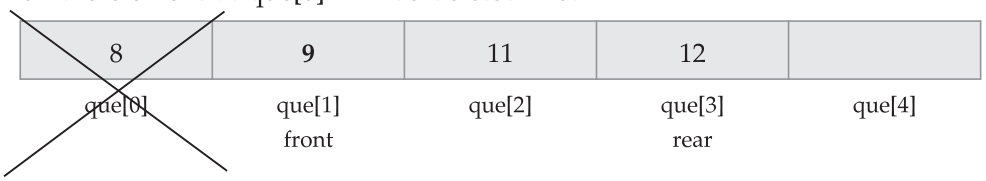
2. Deletion operation

In the deletion operation we are simply removing the element at the front.

For example if queue is created like this –



Then the element at que[0] will be deleted first



and then new front will be que[1].

The deletion operation in C++ is as given below -

```
int Pr_Q::delet(int front)
{
        int item;
        item=que[front];
        cout<<"\n The item deleted is  "<<item;
        front++;
        return front;
}
```

The complete implementation of priority queue is as given below -

C++ Program

```
/*********************************************************
Program for implementing the ascending priority Queue
 *********************************************************/
      /*Header Files*/
      #include<iostream>
      #define SIZE 5
      using namespace std;
      class Pr_Q
      {
       private:
              int que[SIZE];
       public:
              int rear,front;
              Pr_Q();
              int insert(int rear,int front);
```

```
        int Qfull(int rear);
        int delet(int front);
        int Qempty(int rear,int front);
        void display(int rear,int front);
};
Pr_Q::Pr_Q()
{
 front=0;
 rear=–1;
}
int Pr_Q::insert(int rear,int front)
{
 int item,j;
 cout<<"\nEnter the element: ";
 cin>>item;
 if(front ==–1)
        front++;
 j=rear;
 while(j>=0 && item<que[j])
 {
  que[j+1]=que[j];
  j--;
 }
 que[j+1]=item;
 rear=rear+1;
 return rear;
}
int Pr_Q::Qfull(int rear)
{
 if(rear==SIZE–1)
  return 1;
 else
 return 0;
}

int Pr_Q::delet(int front)
{
 int item;
 item=que[front];
 cout<<"\n The item deleted is  "<<item;
 front++;
 return front;
}
int Pr_Q::Qempty(int rear,int front)
{
 if((front==–1)||(front>rear))
```

```
 return 1;
 else
 return 0;
}
void Pr_Q::display(int rear,int front)
{
 int i;
 cout<<"\n The queue is: ";
 for(i=front;i<=rear;i++)
 cout<<" "<<que[i];
}

int main(void)
{
 int choice;
 char ans;
 Pr_Q obj;

 do
 {
  cout<<"\n\t\t Priority Queue\n";
  cout<<"\n Main Menu";
  cout<<"\n1.Insert\n2.Delete\n3.Display";
  cout<<"\n Enter Your Choice: ";
  cin>>choice;
  switch(choice)
  {
        case 1:if(obj.Qfull(obj.rear))
                        cout<<"\n Queue IS full";
               else
                     obj.rear=obj.insert(obj.rear,obj.front);
               break;
        case 2:if(obj.Qempty(obj.rear,obj.front))
                        cout<<"\n Cannot delete element";
               else
                        obj.front=obj.delet(obj.front);
               break;
        case 3:if(obj.Qempty(obj.rear,obj.front))
                        cout<<"\n Queue is empty";
               else
               obj.display(obj.rear,obj.front);
               break;
  default:cout<<"\n Wrong choice: ";
            break;
  }
   cout<<"\n Do You Want TO continue?";
```

```
    cin>>ans;
  }while(ans=='Y' ||ans=='y');
  return 0;
}
```

Output

```
                Priority Queue

Main Menu
1.Insert
2.Delete
3.Display
 Enter Your Choice: 1

Enter the element: 20

 Do You Want TO continue?

                Priority Queue

 Main Menu
1.Insert
2.Delete
3.Display
 Enter Your Choice: 1

Enter the element: 30

 Do You Want TO continue?

                Priority Queue
Main Menu
1.Insert
2.Delete
3.Display
 Enter Your Choice: 1

Enter the element: 10

 Do You Want TO continue?

                Priority Queue
 Main Menu
1.Insert
2.Delete
3.Display
 Enter Your Choice: 1
```

```
Enter the element: 40

 Do You Want TO continue?

              Priority Queue
 Main Menu
1.Insert
2.Delete
3.Display
 Enter Your Choice: 3

 The queue is:  10 20 30 40
 Do You Want TO continue?

              Priority Queue

 Main Menu
1.Insert
2.Delete
3.Display
 Enter Your Choice: 2

 The item deleted is  10
 Do You Want TO continue?
```

Example 6.9.1 *Specify which of the following applications would be suitable for a first-in-first-out queue and Justify your answer :*

i) A program is to keep track of patients as they check into a clinic, assigning them to doctors on a first come, first-served basis.

ii) An inventory of parts is to be processed by part number.

iii) A dictionary of words used by spelling checker is to be created.

iv) Customers are to take numbers at a bakery and be served in order when their numbers come-up.

Solution :

i) No, this application will not be suitable for first come first serve queue because sometimes some serious patients may be come and he/she needs doctors in urgent.

ii) Yes, for this application first-in-first-out queue is suitable because here the part number must be stored in specific manner (order).

iii) No, because words can be chosen in random order.

iv) Yes, bakery customers must be served if first come first serve basis.

Review Questions

1. *What are the applications of the data structure priority queue.*
2. *What is priority queue ? What is its use ? Give the function to add an element in priority queue.*
3. *Write a note on priority queues.*
4. *What is priority queue ? Explain insert and delete operations in detail using multidimensional array implementation.*
5. *List down applications of queue.*

SOLVED MODEL QUESTION PAPER

(As Per 2019 Pattern)

Fundamentals of Data Structures

S.E. (Computer) Sem - I (End Sem)

Time : 2 ½ Hours] **[Total Marks : 70**

Instructions to the candidates :

1) **Answer Q.1 or Q.2, Q.3 or Q.4, Q.5 or Q.6, Q.7 or Q.8.**

2) **Neat diagrams must be drawn wherever necessary.**

3) **Figures to the right indicate full marks.**

4) **Make suitable assumptions if necessary.**

Q.1 **(a)** *Explain sentinel search technique.* **(Refer section 3.2.2)** **[6]**

(b) *Write a non recursive Python Program to illustrate binary search technique* **(Refer section 3.2.3)** **[6]**

(c) *Explain selection sort technique with suitable example.* **(Refer section 3.5.3)** **[6]**

OR

Q.2 **(a)** *Give the difference between linear search and binary search.* **(Refer section 3.2.3)** **[4]**

(b) *Explain Fibonacci search technique with suitable example.* **(Refer section 3.2.4)** **[6]**

(c) *Write a Python program for implementing radix sort technique.* **(Refer section 3.6.1)** **[8]**

Q.3 **(a)** *Explain C++ code for insertion of a node in a singly linked list.* **(Refer section 4.6)** **[6]**

(b) *What is doubly linked list ? Differentiate between singly and doubly linked list.* **(Refer section 4.8)** **[6]**

(c) *Explain different types of linked list.* **(Refer section 4.7)** **[5]**

OR

Q.4 **(a)** *Write a C++ function for addition of two polynomials using linked list.* **(Refer section 4.12)** **[7]**

(b) *Explain generalized linked list with suitable example.* **(Refer section 4.13)** **[6]**

(c) *What are applications of linked list ?* **(Refer section 4.11)** **[4]**

Q.5 **(a)** *What is stack ? Give an ADT of stack.* **(Refer sections 5.1 and 5.2)** **[6]**

(b) *Write C++ functions for push and pop operations. Illustrate these operations with suitable example.* **(Refer section 5.4)** **[6]**

(c) *Write a short note on - Multiple stack.* **(Refer section 5.5)** **[6]**

OR

Q.6 **(a)** *Write a C++ code for evaluation of postfix expression.* **(Refer section 5.8)** **[6]**

(b) *Explain the concept of recursion with suitable example.* **(Refer section 5.10)** **[6]**

(c) *Write a note on - Backtracking algorithmic strategy.* **(Refer section 5.11)** **[6]**

Q.7 **(a)** *What is queue ? Give an ADT of queue.* **(Refer sections 6.1 and 6.2)** **[5]**

(b) *Explain the concept of circular queue in detail.* **(Refer section 6.5)** **[6]**

(c) *Write C++ code for performing insert and delete operations in linked queue.* **(Refer section 6.7)** **[6]**

OR

Q.8 **(a)** *What is priority queue ? Explain.* **(Refer section 6.9)** **[8]**

(b) *Write a C++ program for performing various operations on queue. Illustrate these operations with sample input data.* **(Refer section 6.4)** **[9]**

❑❑❑