Q I)Perform the evaluation of a Postfix expression using the stack.
Example 123*+4- evaluates to 3

Program:

```
#include <iostream.h>
#include <stdlib.h>
#include <math.h>
#include <ctype.h>
constint MAX = 50 ;
class postfix
{
        private :

                int stack[MAX] ;
                int top, nn ;
                char *s ;
        public :
                postfix( ) ;
                voidsetexpr ( char *str ) ;
                void push ( int item ) ;
                int pop( ) ;
                void calculate( ) ;
                void show( ) ;
} ;
postfix :: postfix( )
{
        top = -1 ;
}
void postfix :: setexpr ( char *str )
{
        s = str ;
}
void postfix :: push ( int item )
{
        if ( top == MAX - 1 )
                cout<<endl<< "Stack is full" ;
        else
        {
                top++ ;
                stack[top] = item ;
        }
}
int postfix :: pop( )
{
        if ( top == -1 )
        {
                cout<<endl<< "Stack is empty" ;
                return NULL ;
        }
        int data = stack[top] ;
        top-- ;
        return data ;
}
void postfix :: calculate( )
{
        int n1, n2, n3 ;
        while ( *s )
        {
                if ( *s == ' ' || *s == '\t' )
                {
                        s++ ;
```

```
                continue ;
        }
        if ( isdigit ( *s ) )
        {
                nn = *s - '0' ;
                push ( nn ) ;
        }
        else
        {
                n1 = pop( ) ;
                n2 = pop( ) ;
                switch ( *s )
                {
                        case '+' :
                                n3 = n2 + n1 ;
                                break ;
                        case '-' :
                                n3 = n2 - n1 ;
                                break ;
                        case '/' :
                                n3 = n2 / n1 ;
                                break ;
                        case '*' :
                                n3 = n2 * n1 ;
                                break ;
                        case '%' :
                                n3 = n2 % n1 ;
                                break ;
                        case '^' :
                                n3 = pow( n2 , n1 ) ;
                                break ;
                        default :
                                cout<< "Unknown operator" ;
                                exit ( 1 ) ;
                }

                push ( n3 ) ;
        }
        s++ ;
    }
}
void postfix :: show( )
{
    nn = pop ( ) ;
    cout<< "Result is: " <<nn ;
}

void main( )
{
    charexpr[MAX] ;
    cout<< "\nEnter postfix expression to be evaluated : " ;
    cin.getline( expr, MAX ) ;
    postfix q ;
    q.setexpr( expr ) ;
    q.calculate( ) ;
    q.show( ) ;
}
```

Output :
Enter postfix expression to be evaluated : 1 2 3 * + 4 –
Result is: 3

Q II) Check if The parenthesis of an expression are balanced using stack.
Example (3+(4+x)*7+(y-2*(2+x))

Program:

```
#include<iostream.h>
#include<conio.h>
#include<string.h>

int const MAX=20;
class parnCheck
{
        private:
                char *expr,temp,*str;
                char stackArr[MAX];
                int i,stackCnt,flag,len;
        public:
                parnCheck();
                int checkExpr(char *);
                void push(char);
                char pop();
                int Empty();
};
void parnCheck::push(char n)
{
        if(stackCnt==MAX-1)
          {
                cout<<"\n Stack FUll";
                return;
          }
        stackCnt++;
        stackArr[stackCnt]=n;
}

int parnCheck::Empty()
{
        if(stackCnt=='-1')
                return 1;
        return 0;
}
char parnCheck::pop()
{
        char tmp=NULL;
        if(!Empty())
        {
                tmp=stackArr[stackCnt];
                stackCnt--;
        }
        return tmp;
}
parnCheck::parnCheck()
{
        i=0;
        flag=0;
        len=0;
        temp=' ';
        stackCnt=-1;
        expr=new char[len];
}
```

```
int parnCheck::checkExpr(char  *str)
{

        len=strlen(str)+1;
        expr=new char[len];
        strcpy(expr,str);

        i=0;
        flag=0;
        while(expr[i]!='\r')
        {
                if(expr[i]=='(' || expr[i]=='{' || expr[i]=='[')
                        push(expr[i]);
                if(expr[i]==')' || expr[i]=='}' || expr[i]==']' )
                {
                        temp=pop();
                        if((expr[i]==')' && temp=='(') || (expr[i]=='}' && temp=='{') || (expr[i]==']'
&& temp=='['))
                        {
                                i++;
                                continue;
                        }
                        else
                        {
                                flag=1;
                                break;
                        }
                }
                i++;
        }
        if((flag!=1))
                return 1;
                return 0;
}
void main()
{
        parnCheck pc;
        clrscr();
        char *str;
        int result;
        str="(3+(4 -x)*7+(y-2*(2+x))";
        cout<<"The Expression is"<<str<<endl;
        result=pc.checkExpr(str);
        if(result)
                cout<<"Expression is Balanced\n";
        else
                cout<<"Expression is not Balanced\n";
        getch();
}
```

output

Expression = (3+(4+x)*7+(y-2*(2+x))

The Expression is not Balanced

Q IV): Demonstrate the working of a stack,implement it as an array(all basic functions of a stack).

Program:

```
#include<iostream.h>
#include<conio.h>
#include<process.h>
//The stack class for managing the stack operations
class Stack {
public:
        int top;
        int a[20];

        Stack() {
        top=0;
        }

        // method to pop an item from  the top of stack
        int pop() {
        int item;
        if(top==0) {
                cout<<"Empty Stack";
                item=-1;
           } else {
                item=a[top];
                top=top-1;
                cout<<"Popped top item";
           }
        return item;
        }

        // method to push the item to the stack
        void push(int n,int item) {
        if(top==n) {
                cout<<"The stack is full";
           } else {
                a[top]=item;
                top=top+1;
           }
        }

        voidprintStack();
};

// program to print the contents of the stack.
void Stack::printStack() {
cout<<"Current status of stack : ";
for(int i=0;i<top;i++) {
        cout<<a[i]<<" ";
   }
}

// program entry point
void main() {

   // create a stack object
   Stack stk;
intitem,choice,n=15;
```

```
   //show a menu using a loop
do {
        clrscr();
        cout<<"Menu";
        cout<<endl<<"1.Push";
        cout<<endl<<"2.Pop";
        cout<<endl<<"3.Print";
        cout<<endl<<"4.Exit";
        cout<<endl<<"Enter your choice :";
        cin>>choice;


        //take the appropriate action depending upon the choice.
        switch(choice) {
        case 1:
                cout<<"Enter the item to be pushed : ";
                cin>>item;
                stk.push(n,item);
                break;

        case 2:
                stk.pop();
                break;

        case 3:
                stk.printStack();
                break;

        case 4:
                exit(0);
                break;
        }
        cout<<endl<<"Do you want to continue (1,2)";
        cin>>choice;
}while(choice==1);
getch();
}
```

Q V) Implement stack as a Link list all basic function.

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<iomanip.h>
#include<assert.h>
typedefstruct Stack
{
        int no;
        struct Stack *next;
}S;
classStackLink
{
        private:
                S *top;
                int count;
                intstackEmpty();

        public:
                StackLink();
                intstackFull();
                int push(int);
                int pop();
                intstackTop();
                intstackCount();
                voidstackDisplay();
};

StackLink::StackLink()
{
        count=0;
        top=NULL;
}
intStackLink::stackFull()
{
        S *snew;
        snew=new S();
        if(snew==NULL)
        {
                return 0;
        }
        else
        {
                delete(snew);
                return 1;
        }
}
intStackLink::stackEmpty()
{
        if(count==0)
        {
                return 0;
        }
        else
        {
                return 1;
        }
}
```

```
intStackLink::push(intnum)
{
	if(stackFull()==0)
	{
		return 0;
	}
	else
	{
		S *snew;
		snew=new S();
		assert(snew);
		snew->no=num;
		snew->next=top;
		top=snew;
		count++;
		return 1;
	}
}
intStackLink::pop()
{
	if(stackEmpty()==0)
	{
		return -1;
	}
	else
	{
		int popdata=top->no;
		S *delptr=top;
		top=top->next;
		delete(delptr);
		count--;
		return popdata;
	}


}
voidStackLink::stackDisplay()
{
	if(stackEmpty()==0)
	{
		cout<<"\n \t \t No items to display as stack is empty ☹";
		cout<<"\n ";

	}
	else
	{
		cout<<"\n \t \t Items in the stack ";
		S *walk;
		walk=top;
		while(walk)
		{
			cout<<"\n \t \t ====>"<<walk->no;
			walk=walk->next;
		}
		cout<<"\n";
	}
}
```

```
intStackLink::stackCount()
{
	return count;
}

intStackLink::stackTop()
{
	if(stackEmpty()==0)
	{
		return 0;
	}
	else
	{
		return top->no;
	}
}

void main()
{
	clrscr();
	StackLink sl1;
	char choice;
	intpushitem,popitem,count,topitem,success;
	abc:
	cout<<"\n ====> Select Operation you would like to		perform"<<endl;
	cout<<"\n ====>a.CheckSatck status Full/Space left ";
	cout<<"\n ====>b.Push an item into stack ";
	cout<<"\n ====>c.Pop an item from stack ";
	cout<<"\n ====>d.Count item in the stack ";
	cout<<"\n ====>e.Top item from the stack";
	cout<<"\n ====>f.Display items from the stack";
	cout<<"\n ====>g.Exit";
	cout<<"\n";
	cout<<"\n ====> Enter the choice :- ";
	cin>>choice;
	switch(choice)
	{
		case 'a':
		if(sl1.stackFull()==0)
		    {
		       cout<<" \n \t \t Stack is full";
		       cout<<"\n";
		    }
		else
		    {
		       cout<<" \n \t \t Stack is no full theres a free space";
		       cout<<"\n";
		    }
		gotoabc;

		case 'b':
			cout<<"\n \t \t Enter a Item to be pushed into stack :-";
			cin>>pushitem;
			success=sl1.push(pushitem);
			if(success==0)
			{
				cout<<" \n \t \t Stack is full";
				cout<<"\n";

			}
			else
```

```
                {
                        cout<<" \n \t \t Item pushed into the stack";
                        cout<<"\n";
                }
                gotoabc;

        case 'c':
                popitem=sl1.pop();
                if(popitem==-1)
                {
                        cout<<"\n \t \t No items in stack to be popped";
                }
                else
                {
                        cout<<" \n \t \t Item :"<<popitem<<": is popped from the stack ";
                        cout<<"\n";
                }
                gotoabc;

        case 'd':
                count=sl1.stackCount();
                if(count==0)
                {
                        cout<<" \n \t \t Stack is Empty Total items are : "<<count;
                        cout<<"\n";
                }
                else
                {
                        cout<<" \n \t \t Total elements in stack is : "<<count;
                        cout<<"\n";
                }
                gotoabc;

        case 'e':
                topitem=sl1.stackTop();
                if(topitem==0)
                {
                        cout<<" \n \t \t Stack is Empty ";
                        cout<<"\n";
                }
                else
                {
                        cout<<" \n \t \t Top Item of stack is : "<<topitem;
                        cout<<"\n";
                }
                gotoabc;

        case 'f':
                sl1.stackDisplay();
                gotoabc;

        case 'g':
                exit(1);

    };
    getch();
}
```

Q vii) Write algorithms to (for a linked list):

a. **pushStack**
b. **popStack**
c. **stackTop**
d. **stackEmpty**
e. **stackFull**
f. **stackCount**
g. **destroyStack**

Ans:

a. **Push stack**

Algorithm pushStack (val stack <head pointer>, val data <dataType>)
Insert (push) one item into the stack.

Pre stack is a pointer to the stack head structure
data contains data to be pushed into stack
Post data have been pushed in stack
Return true if successful; flase if memory overflow

1 if(stack full)
  1 success = false
2 else
  1 allocate(newPtr)
  2 newPtr->data = data
  3 newPtr->next = stack->top
  4 stack->top = newPtr
  5 stack->count = stack->count + 1
  6 success = true
3 return success
End pushStack

b. **Pop Stack**

Algorithm popStack (val stack <head pointer>, ref dataOut<data type>)
This algorithm pops the item on the top of the stack and returns it to the user.

Pre stack is a pointer to the stack ead structure
dataOut is a reference variable to receive the data
Post Data have been returned to calling algorithm
Return true if successful; false if underflow

1 if(stack empty)
  1 success = false
2 else
  1 dltPtr = stack->top
  2 dataOut = stack->top->data
  3 stack->top = stack->top->next
  4 stack->count = stack->count-1
  5 recycle(dltPtr)
  6 success = true
3 return success
End popStack

c. Top Stack

algorithm stackTop(val stack <head pointer>,ref dataoOut<datatype>)
This algorithm retrieves the data from the top of stack without changing the stack.
Pre stack is a pointer to the stack head structure dataOut is a reference variable to receive data
Post Data have been returned to calling algorithm
Return true if data returned ,false if underflow

1 if(stack empty)
  1 success=false
2 else
  1 dataOut=stack->top->data
  2 success=true
3 return success
End stackTop

d. Empty Stack

algorithm stackEmpty (val stack <head pointer>)
Determines if stack is empty and return a boolean
Pre stack is a pointer to the stack head structure
Post returns stack status
Return Boolean ,true:stackempty,false:stack contains data
1 if(stack not empty)
  1 result=false
2 else
  1 result =true
3 return result
End stackEmpty

e. Full Stack

algorithm stackFull (val stack <head pointer>)
Determines if stack is full and return a Boolean.
Pre stack is a pointer to the stack head structure
Post returns stack status
Return Boolean ,true: stack full, false: memory available
1 if (memory available)
  1 result=false
2 else
  1 result=true
3 return result
end stackFull

f. Stack Count

algorithm stackCount (val stack <head pointer>)
Returns the number of elements currently in stack.
Pre stack is a pointer to the stack head structure
Post returns stack count
Return integer count of number of elements in stack
1 return (stack->count)
end stackCount

g. Destroy Stack

algorithmdestroyStack (val stack <head pointer>)
This algorithm releases all nodes back to the dynamic memory
Pre stack is a pointer to the stack head structure
Post stack empty and all nodes recycled
Return null pointer

```
1   if (stack not empty)
    1   loop( stack->top to null )
        1   temp=stack->top
        2   stack->top=stack->top->link
        3   recycle ( temp )
2   recycle(stack)
3   Return null pointer
End destroyStack
```

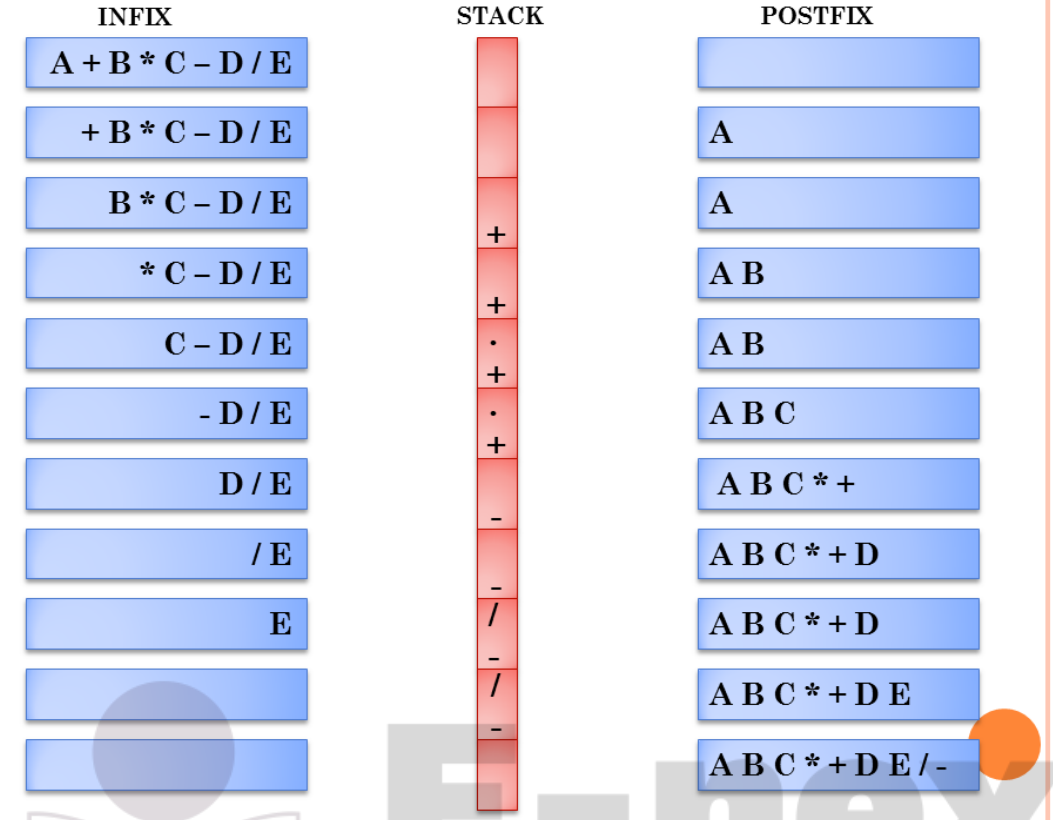
Q viii) Write an algorithm for conversion from infix to postfix along with a diagram.

Algorithm intoPostFix(val formula<string>)
Convert infix formula to postfix
Pre formula is infix notation that has been edited to ensure that there are no syntactical errors.
Post postfix formula has been formatted as a string.
Return – postfix formula.

```
1. stack=createStack
2. set postfix to null string
3. looper = 1
4. loop(looper<= sizeof formula)
    1  token = formula[looper]
    2  if(token is open parenthesis)
        1  pushStack(stack, token)
  3elseif(token is close parenthesis )
        1  popStack(stack, token)
        2  loop (token not open parenthesis)
            1  concatenate token to postfix
            2  popStack(stack, token)
        3  end loop
  4  elseif(token is operator)
        Test priority of token to token at top of stack
        1  stackTop (stack, topToken)
        2  loop (not emptyStack (stack) AND priority(token)<= priority(topToken))
            1. popStack (stack, tokenOut)
            2. concatenate tokenOut to postfix
            3. stackTop(stack,topToken)
        3.     end loop
        4 pushStack(stack, token)
  5  else
                character is operand
        1  concatenate  token  to  postfix
    6. end if

5. end loop
    input formula empty. Pop stack to postfix
  6  .  looper = looper + 1
  7.  loop (not  emptystack(stack))
        1. popStack (stack,token)
        2. concatenate token to postfix
  8. end loop
  9. return postfix
End intoPostFix
```

Diagram :
A + B * C – D / E ……. Converts to …….. A B C * + D E / -



Q IX) Write an algorithm for conversion from infix to prefix along with the diagram

Algorithm intoPreFix(valexpr<string>)
Convert infix expr to postfix
Pre expr is infix notation that has been edited to ensure that there are no syntactical errors.
Post prefixexpr has been formatted as a string.
Return –print prefix expr.

Create OperandStack
Create OperatorStack
1.loop(not an empty input expression) read next token from the input expression
1.1 if (token is an operand)
1.1.1 OperandStack.Push (token)
1.2 else if (token is '(' or OperatorStack.IsEmpty() or OperatorHierarchy(token)>OperatorHierarchy(OperatorStack.Top()))
1.2.1 OperatorStack.Push(token)
1.3 else if(token is ')')
1.3.1 loop(OperatorStack.Top() not equal '(')
1.3.1.1 OperatorStack.Pop(operator)
1.3.1.2 OperandStack.Pop(RightOperand)
1.3.1.3 OperandStack.Pop(LeftOperand)
1.3.1.4 operand = operator + LeftOperand + RightOperand
1.3.1.5 OperandStack.Push(operand)
1.3.2 OperatorStack.Pop(operator)
1.4 else if(operator hierarchy of token is less than or equal to hierarchy of top of the operator stack)
1.4.1 loop(!OperatorStack.IsEmpty() and OperatorHierarchy(token) lessThen Or Equal to OperatorHierarchy (OperatorStack.Top()))
1.4.1.1 OperatorStack.Pop(operator)
1.4.1.2 OperandStack.Pop(RightOperand)
1.4.1.3 OperandStack.Pop(LeftOperand)
1.4.1.4 operand = operator + LeftOperand + RightOperand
1.4.1.5 OperandStack.Push(operand)
1.4.2 OperatorStack.Push(token)
2 loop(!OperatorStack.IsEmpty()) OperatorStack.Pop(operator)
2.1 OperandStack.Pop(RightOperand)
2.2 OperandStack.Pop(LeftOperand)
2.3 operand = operator + LeftOperand + RightOperand
2.4 OperandStack.Push(operand)
3 print OperandStack.Top()
4 OperandStack.Pop()
End

Q.X) Theoretically explain the three steps to convert from infix to postfix notation with an example.

Ans.:

- The steps for converting the expression manually are given here.

Step 1. The actual order of evaluation of the expression in infix notation is determined by inserting parentheses in the expression according to the precedence and associativity of operators.

Step 2. The expression in the innermost parentheses is converted into postfix notation by placing the operator after the operands on which it operates.

Step 3. Step 2 is repeated until the entire expression is converted into a postfix notation.

For example:

- To convert the expression a+b*c into equivalent postfix notation, these steps are followed:

(1)Since the precedence of * is higher than +. the expression ' b* c ' has to be evaluated first. Hence, the expression is written as (a+(b*c))
(2)The expression in the innermost parentheses, that is, b*c is converted into its postfix notation. Hence, it is written as ' bc* '. The expression now becomes (a+bc*)
(3)Now the operator + has to be placed after its operands. The two operands for + operator are 'a' and the expression ' bc* '.
The expression now becomes (abc*+)

Q XI)Briefly explain the towers of Hanoi.also write the algorithm for the same.

Ans : **The Towers of Hanoi**

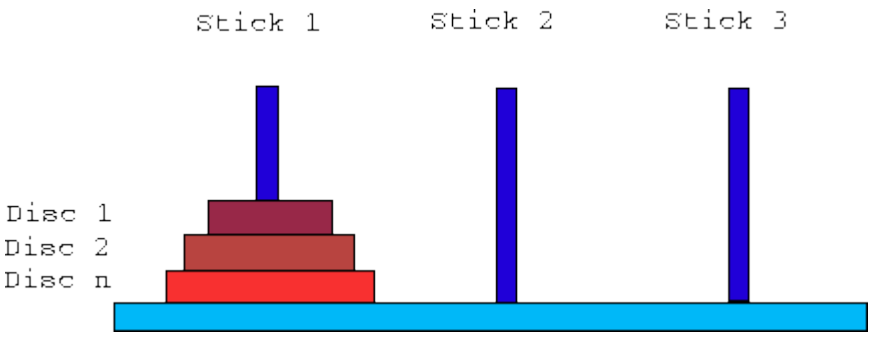
The Tower of Hanoi (also called the Tower of Brahma or Lucas' Tower, and sometimes pluralised) is a mathematical game or puzzle.

It consists of three rods, and a number of disks of different sizes which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape.

The objective of the puzzle is to move the entire stack to another rod, obeying the following rules:

- Only one disk may be moved at a time.
- Each move consists of taking the upper disk from one of the rods and sliding it onto another rod, on top of the other disks that may already be present on that rod.
- No disk may be placed on top of a smaller disk.

With three disks, the puzzle can be solved in seven moves.



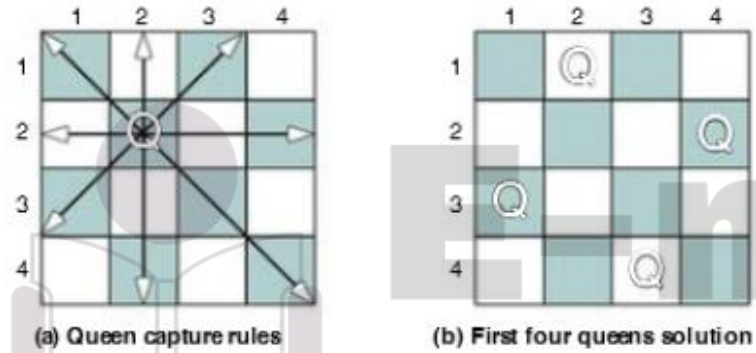
Algorithm:
1. Declare necessary variables such as n, A, B, C etc.
2. Input number of disks
3. If n=1
   move single disk from peg A to peg C and stop.
   Else
   move the top (n-1) disks from peg A to peg B using peg C as auxiliary.
4. Move remaining disks from peg A to peg C.
5. Move (n-1) disks from peg B to peg C using peg A as auxiliary

Q xii)Theoretically explain the eight-queen problem
Ans:

A classic chess problem requires that you place eight queens on the chess board in such a way that no queen can capture another queen. The eight queens puzzle is the problem of placing eight chess queens on an 8×8 chessboard so that no two queens attack each other. Thus, a solution requires that no two queens share the same row, column, or diagonal.

The computer solution to this problem requires that we place a queen on the board and then analyze all of the attack positions to see if there is a queen that could capture the new queen. If there is, then we try another position.

The queen's capture rules and one solution are shown below:



Four Queens Solution

This problem can be solved using a stack and backtracking logic, because only one queen can be placed in any row. So we begin by placing one queen in row 1, column 1 and this position is pushed into the stack. (Step–1)

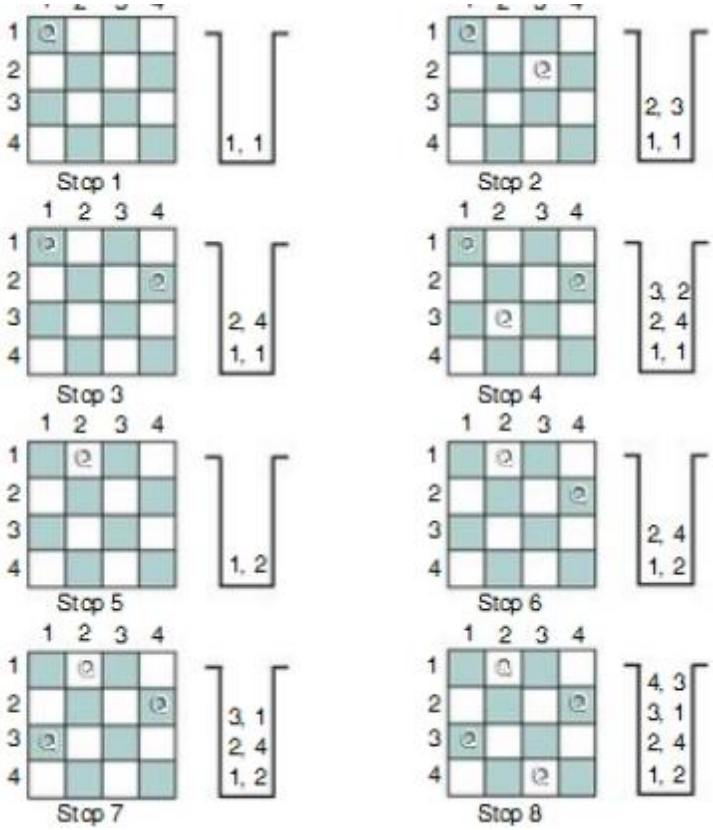
After placing a queen in the first row, look for a position in the second row. Position 2,1 is not possible because the queen in the first row is guarding this location on the vertical. Likewise, 2,2 is guarded on the diagonal. Therefore we place a queen in the third column in row 2 and push this location into the stack (Step-2)

Now try to locate a position in row 3, such that no other queen can capture another queen but none are possible. The first column is guarded by the queen in row 1 and the other three positions are guarded by the queen in row 2. At this point we must backtrack to the second row by popping the stack and continue looking for a position for the second-row queen, because column 4 is not guarded, we place a queen there and push its location into the stack. (Step-3)

Now, again at row 3, we see that the first column is still guarded by the queen in row 1 but that we can place a queen in the second column and now push the location into the stack (Step-4).

When we try to place a queen in row 4, however, we find all positions are guarded. Column 1 is guarded by the queen in row 1 and the queen in row 3. Column 2 is guarded by the queen in row 2 and the queen in row 3. Column 3 is guarded by the queen in row 3, and column 4 is guarded by both the queen in row 1 and row 2.

Now, therefore backtrack to the queen in row 3 and try to find another place for the queen because the queen in row 2 is guarding both column 3 and column 4, there is no option for a queen in row 3. Once again we backtrack by popping the stack and find that the queen in row 2 has nowhere else to go, so now backtrack to the queen in row 1 and move the queen to column 2. This position is shown in the figure below (Step-5).



Four queens Step-by-step Solution

Column 1 in the third row is unguarded, so we place a queen there (step-7). Moving to row 4, find that the first two positions are guarded, the first ny the queen in row 3 and the second by all three queens. The third column is unguarded, however so we can place the fourth queen in this column for a solution to the problem.

Generalizing the solution, we see that we place a queen in a position in a row and then examine all positions in the next row to see if a backtrack to the last-positioned queen and try to position her in the next column. If there is no room in the next column, we fall back again.

Similarly, the eight queen's problem is solved and the queens in the chess board of size 8*8 is shown below: