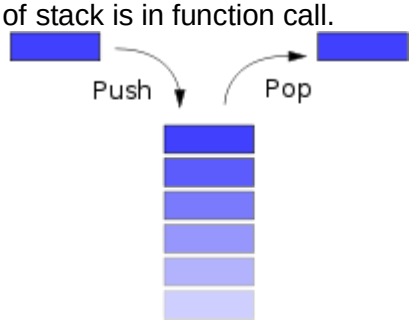
Data Structure

In Computer Science, a **data structure** is a particular way of storing and organizing data in a computer so that it can be used efficiently. Different kinds of data structures are suited to different kinds of applications, and some are highly specialized to specific tasks.

The data structure can be classified into following two types:
Simple Data Structure: These data structures are normally built from primitive data types like integers, floats, characters. For example arrays and structure.
Compound Data Structure: simple data structures can be combined in various ways to form more complex structure called compound structures. Linked Lists, Stack, Queues and Trees are examples of compound data structure.

Stack, Queues using Arrays and Linked List

Stack
In computer science, a stack is a last in, first out (LIFO) *data structure*. A stack can is characterized by only two fundamental operations: *push* and *pop*. The push operation adds an item to the top of the stack. The pop operation removes an item from the top of the stack, and returns this value to the caller.
A stack is a *restricted data structure*, because only a small number of operations are performed on it. The nature of the pop and push operations also mean that stack elements have a natural order. Elements are removed from the stack in the reverse order to the order of their addition: therefore, the lower elements are those that have been on the stack the longest. One of the common uses of stack is in function call.



Stack using array

```
#include<iostream.h>
const int size=5
class stack
{int a[size];      //array a can store maximum 5 item of type int of the stack
int top;           //top will point to the last item pushed onto the stack
public:
stack(){top=-1;}         //constructor to create an empty stack, top=-1 indicate that no item is
                         //present in the array
void push(int item)
{If(top==size-1)
        cout<<"stack is full, given item cannot be added";
else
       a[++top]=item; //increment top by 1 then item at new position of the top in the array a
}
int pop()
```

```
{If (top==-1)
     {out<<"Stack is empty ";
      return -1; //-1 indicates empty stack
     }
else
     return a[top--];//return the item present at the top of the stack then decrement top by 1
}
void main()
{ stack s1;
  s1.push(3);
  s1.push(5);
cout<<s1.pop()<<endl;
cout<<s1.pop()<<endl;
cout<<s1.pop();
}
```

Output is

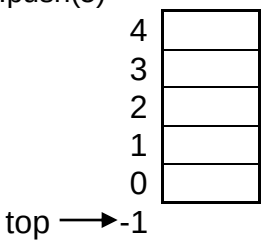
5

3

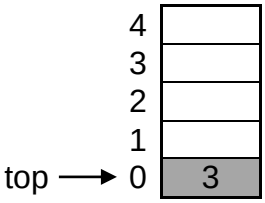
Stack is empty -1

In the above program the statement **stack s1** creates s1 as an empty stack and the constructor initialize top by -1.

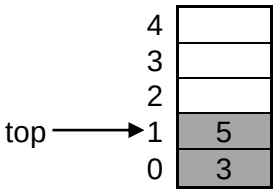
Initially stack is empty



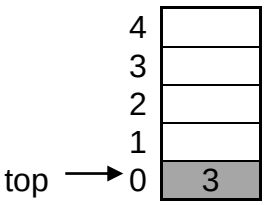
stack after s1.push(3)



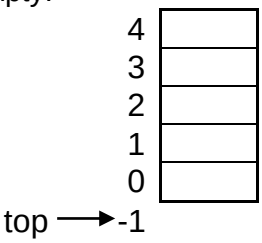
stack after s1.push(5)



After first s1.pop() statement, the item 5 is removed from the stack and top moves from 1 to 0



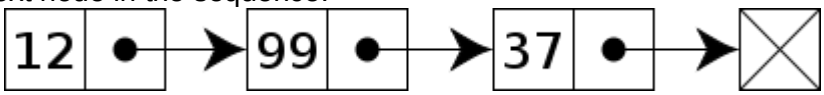
After second s1.pop() statement, the item 3 is removed from stack and top moves from 0 to -1 which indicates that now stack is empty.



After third s1.pop() statement the pop function display error message "stack is empty" and returns -1 to indicating that stack is empty and do not change the position of top of the stack.

Stack using Linked list

In Computer Science, a **linked list** (or more clearly, "singly-linked list") is a data structure that consists of a sequence of nodes each of which contains data and a pointer which points (i.e., a *link*) to the next node in the sequence.



A linked list whose nodes contain two fields: an integer value and a link to the next node

The main benefit of a linked list over a conventional array is that the list elements can easily be added or removed without reallocation or reorganization of the entire structure because the data items need not be stored contiguously in memory or on disk .Stack using linked lists allow insertion and removal of nodes only at the position where the pointer top is pointing to.

Stack implementation using linked list

```
#include<iostream.h>
struct node
{
Int item; //data that will be stored in each node
node * next; //pointer which contains address of another node
}; //node is a self referential structure which contains reference of another object type node

class stack
{node *top;
public:
stack()  //constructor to create an empty stack by initializing top with NULL
{ top=NULL; }
void push(int item);
int pop();
~stack();
};
void stack::push(int item)    //to insert a new node at the top of the stack
{node *t=new node; //dynamic memory allocation for a new object of node type
if(t==NULL)
        cout<<"Memory not available, stack is full";
else
        {t->item=item;
        t->next=top; //newly created node will point to the last inserted node or NULL if
                        //stack is empty
        top=t;          //top will point to the newly created node
        }
}

int stack::pop()//to delete the last inserted node(which is currently pointed by the top)
{if(top==NULL)
        {cout<<"Stack is empty \n";
        return 0; // 0 indicating that stack is empty
        }
else
        {node *t=top; //save the address of top in t
        int r=top->item;  //store item of the node currently pointed by top
        top=top->next;  // move top from last node to the second last node
        delete t; //remove last node of the stack from memory
        return r;
```

```
        }
}
stack::~stack()//de-allocated all undeleted nodes of the stack when stack goes out of scope
{node *t;
while(top!=NULL)
        {t=top;
        top=top->next;
        delete t;
        }
};

void main()
{ stack s1;
  s1.push(3);
  s1.push(5);
  s1.push(7);
  cout<<s1.pop()<<endl;
cout<<s1.pop()<<endl;
cout<<s1.pop()<<endl;
cout<<s1.pop()<<endl;
}
```

Output is

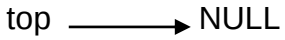
7

5

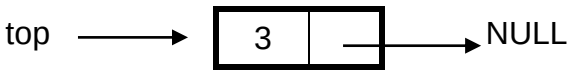
3

Stack is empty 0

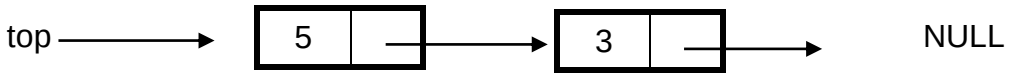
In the above program the statement *stack s1;* invokes the constructor stack() which create an empty stack object s1 and initialize top with NULL.



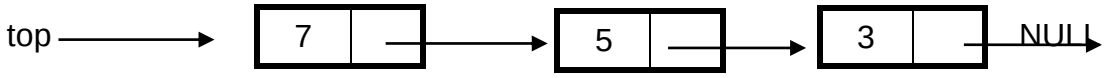
After statement s1.push(3) the stack become



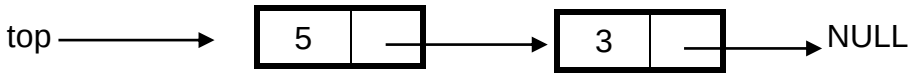
After statement s1.push(5) the stack become



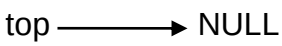
After statement s1.push(7) the stack become

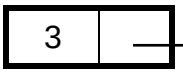


After the first s1.pop() statement the node currently pointed by top (i.e. node containing 7) is deleted from the stack, after deletion the status of stack is



After the second s1.pop() statement the node currently pointed by top (i.e. node containing 5) is deleted from the stack, after deletion the status of stack is



After the third s1.pop() statement the node  currently pointed by top (i.e. node containing 3) is deleted from the stack, after deletion the stack become empty i.e.

Top ⟶ NULL

After the fourth s1.pop() statement, the error message “stack is empty“ displayed and the pop() function return 0 to indicate that stack is empty.

Application of stacks in infix expression to postfix expression conversion

Infix expression **operand1 operator operand2** for example **a+b**
Postfix expression **operand1 operand2 operator** for example **ab+**
Prefix expression **operator operand1 operand2** for example **+ab**

Some example of infix expression and their corresponding postfix expression

Infix expression	postfix expression
a*(b-c)/e	abc-*e/
(a+b)*(c-d)/e	ab+cd-*e/
(a+b*c)/(d-e)+f	abc*+de-/f+

Algorithm to convert infix expression to postfix expression using stack

Let the infix expression INEXP is to be converted in to equivalent postfix expression POSTEXP. The postfix expression POSTEXP will be constructed from left to right using the operands and operators (except “(“, and “)”) from INEXP. The algorithm begins by pushing a left parenthesis onto the empty stack, adding a right parenthesis at the end of INEXP, and initializing POSTEXP with null. The algorithm terminates when stack become empty.
The algorithm contains following steps

1. Initialize POSTEXP with null
2. Add ‘)’ at the end of INEXP
3. Create an empty stack and push ‘(‘ on to the stack
4. Initialize i=0,j=0
5. Do while stack is not empty
6. If INEXP[i] is an operand then
   POSTEXP[j]=INEXP[i]
   I=i+1
   j=j+1
   Goto step 5
7. If INEXP[i] is ‘(‘ then
   push (INEXP[i])
   i=i+1
   Goto step 5
8. If INEXP[i] is an operator then
   While precedence of the operator at the top of the stack > precedence of operator
   POSTEXP[j]=pop()
   J=j+1
   End of while
   Push (INEXP[i])
   I=i+1
   Goto step 5
9. If INEXP[i] is ‘)’ then
   While the operator at the top of the stack is not ‘(‘
   POSTEXP[j]=pop()

J=j+1
End while
Pop()

10. End of step 5
11. End algorithm

For example convert the infix expression (A+B)*(C-D)/E into postfix expression showing stack status after every step.

Symbol scanned from infix	Stack status (bold letter shows the top of the stack)	Postfix expression
	(	
(	(**(**	
A	(**(**	A
+	((**+**	A
B	((**+**	AB
)	**(**	AB+
*	(*****	AB+
(	(***(**	AB+
C	(***(**	AB+C
-	(*(**-**	AB+C
D	(*(**-**	AB+CD
)	(*****	AB+CD-
/	(**/**	AB+CD-*
E	(**/**	AB+CD-*E
)		AB+CD-*E/

Answer: Postfix expression of **(A+B)*(C-D)/E** is **AB+CD-*E/**

Evaluation of Postfix expression using Stack

Algorithm to evaluate a postfix expression P.

1. Create an empty stack
2. i=0
3. while P[i] != NULL
   if P[i] is operand then
   Push(P[i])
   I=i+1
   Else if P[i] is a operator then
   Operand2=pop()
   Operand1=pop()
   Push (Operand1 operator Operator2)
   End if
4. End of while
5. return pop() // return the calculated value which available in the stack.
   End of algorithm

Example: Evaluate the following postfix expression showing stack status after every step
8, 2, +, 5, 3, -, *, 4 /

token scanned from postfix expression	Stack status (bold letter shows the top of the stack) after processing the scanned token	Operation performed
8	**8**	Push 8
2	8, **2**	Push 2
+	**10**	Op2=pop() i.e 2

		Op1=pop() i.e 8 Push(op1+op2) i.e. 8+2
5	10, **5**	Push(5)
3	10, 5, **3**	Push(3)
-	10, **2**	Op2=pop() i.e. 3 Op1=pop() i.e. 5 Push(op1-op2) i.e. 5-3
*	**20**	Op2=pop() i.e. 2 Op1=pop() i.e. 10 Push(op1-op2) i.e. 10*2
4	20, **4**	Push 4
/	5	Op2=pop() i.e. 4 Op1=pop() i.e. 20 Push(op1/op2) i.e. 20/4
NULL	Final result 5	Pop 5 and return 5

Example:
Evaluate the following Boolean postfix expression showing stack status after every step
True, False, True, AND, OR, False, NOT, AND

token scanned from postfix expression	Stack status (bold letter shows the top of the stack) after processing the scanned token	Operation performed
True	**True**	Push True
False	True, **False**	Push False
True	True, False, **True**	Push True
AND	True, **False**	Op2=pop() i.e. True Op1=pop() i.e. False Push(Op2 AND Op1) i.e. False AND True=False
OR	True	Op2=pop() i.e. False Op1=pop() i.e. True Push(Op2 OR Op1) i.e. True OR False=True
False	True, **False**	Push False
NOT	True, **True**	Op1=pop() i.e. False Push(NOT False) i.e. NOT False=True
AND	**True**	Op2=pop() i.e. True Op1=pop() i.e. True Push(Op2 AND Op1) i.e. True AND True=True
NULL	Final result **True**	Pop True and Return True

Queue
Queue is a linear data structure which follows First In First Out (FIFO) rule in which a new item is added at the rear end and deletion of item is from the front end of the queue. In a FIFO data structure, the first element added to the queue will be the first one to be removed.

Linear Queue implementation using Array

```
#include<iostream.h>
const int size=5;
```

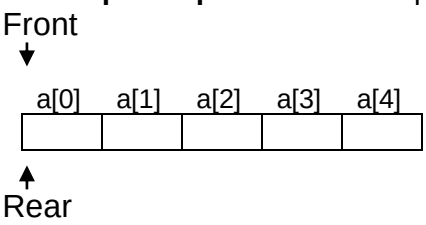
```
class queue
{int front , rear;
int a[size];
public:
queue(){frot=0;rear=0;} //Constructor to create an empty queue
void addQ()
{ if(rear==size)
        cout<<"queue is full<<endl;
 else
        a[rear++]=item;
}
int delQ()
{if(front==rear)
        {cout<<"queue is empty"<<endl; return 0;}

else
        return a[front++];
}
}
void main()
{queue q1;
q1.addQ(3);
q1.addQ(5) ;
q1.addQ(7) ;
cout<<q1.delQ()<<endl ;
cout<<q1.delQ()<<endl ;
cout<<q1.delQ()<<endl;
cout<<q1.delQ()<<endl;
}
```

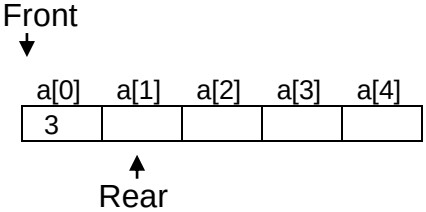
Output is

3
5
7
Queue is empty
0

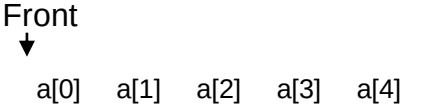
In the above program the statement **queue q1** creates an empty queue q1.

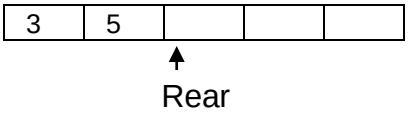


After execution of the statement **q1.addQ(3),** status of queue is

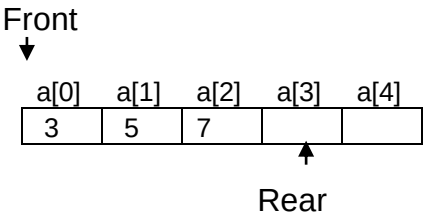


After execution of the statement **q1.addQ(5),** status of queue is

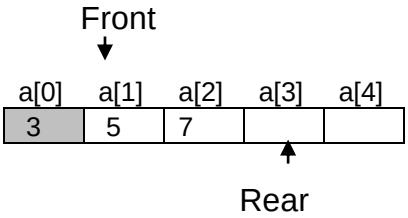




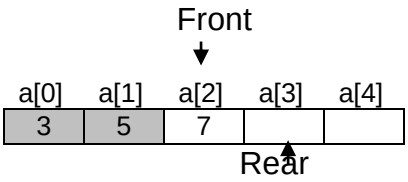
After execution of the statement **q1.addQ(7),** status of queue is



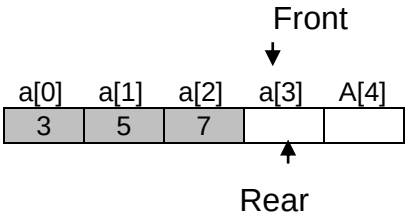
After execution of the first cout<<**q1.delQ()** statement**,** 3 is deleted from queue status of queue is



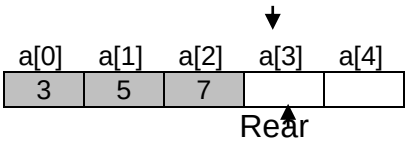
After execution of the second cout<<**q1.delQ()** statement**, 5** is deleted from the queue status of queue is



After execution of the third cout<<**q1.delQ()** statement**, 7** is deleted from the queue. The status of queue is empty



After execution of the fourth cout<<**q1.delQ()** statement**,** the message "queue is empty" displayed and status of queue is



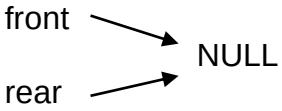
Note that since rear and front moves only in one direction therefore once the rear cross the last element of the array(i.e. rear==size) then even after deleting some element of queue the free spaces available in queue cannot be allocated again and the function delQ() display error message "queue is full".

Queue using linked list

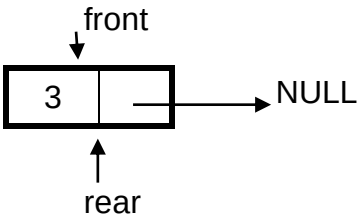
```
#include<iostream.h>
struct node
{
int item;
node *next;
};
class queue
```

```
{
node *front, *rear;
public:
queue() {front=NULL; rear=NULL;}//constructor to create empty queue
void addQ(int item);
int delQ();
};
void queue::addQ(int item)
{node * t=new node;
if(t==NULL)
        cout<<"memory not available, queue is full"<<endl;
else
        {t->item=item;
         t->next=NULL;
        if (rear==NULL)   //if the queue is empty
                {rear=t; front=t;     //rear and front both will point to the first node
                }
        else
                {
                rear->next=t;
                 rear=t;
                }
        }
}
int queue::delQ()
{
if(front==NULL)
        cout<<"queue is empty"<<return 0;
else
        {node *t=front;
         int r=t->item;
         front=front->next; //move front to the next node of the queue
         if(front==NULL)
                rear==NULL;
         delete t;
         return r;
        }
}
void main()
{
queue q1;
q1.addQ(3);
q1.addQ(5) ;
q1.addQ(7) ;
cout<<q1.delQ()<<endl ;
cout<<q1.delQ()<<endl ;
cout<<q1.delQ()<<endl;
cout<<q1.delQ()<<endl;
}
Output is
3
5
7
Queue is empty          0
```

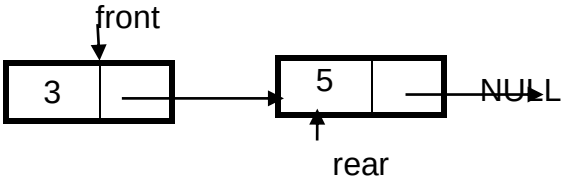
In the above program the statement **queue q1;** invokes the constructor queue() which create an empty queue object q1 and initialize front and rear with NULL.



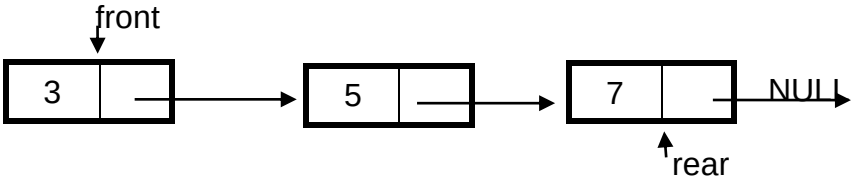
After statement q1.addQ(3) the stack become



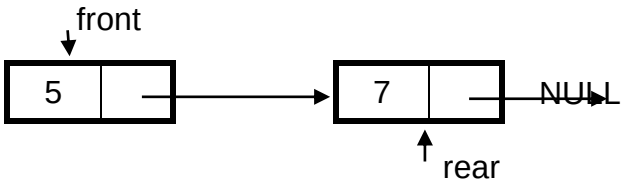
After statement q1.addQ(5) the stack become



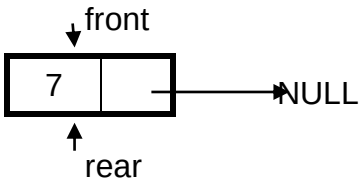
After statement q1.addQ(7) the stack become



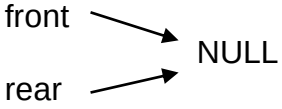
After the first q1.delQ() statement the node currently pointed by front (i.e. node containing 3) is deleted from the queue, after deletion the status of queue is



After the second q1.delQ() statement the node currently pointed by front (i.e. node containing 5) is deleted from the queue, after deletion the status of queue is



After the third q1.delQ() statement the node currently pointed by front (i.e. node containing 7) is deleted from the queue, after deletion the queue become empty therefore NULL is a assigned to both rear and front



After the fourth q1.delQ() statement, the error message “queue is empty“ displayed and the pop() function return 0 to indicate that queue is empty.

STACK AND QUEUE-

POSTFIX CONVERSION/EVALUATION

55. Convert the following infix expression to its equivalent postfix
expression, showing the stack contents for each step of conversion.

X / Y + U* (VW)

Ans 55. X / Y + U* (VW)=((X / Y)+(U*(VW)))

Element	Stack	Postfix
(		
(		
X		X
/	/	X
Y	/	XY
)		XY/
+	+	XY/
(	+	XY/
U	+	XY/U
*	+*	XY/U
(	+*	XY/U
V	+*	XY/UV
-	+*-	XY/UV
W	+*-	XY/UVW
)	+*	XY/UVW-
)	+	XY/UVW-*
)		XY/UVW-*+

56. Convert the following infix expression to its equivalent Postfix
expression, showing the stack contents for each step of conversion.

U * V + R/ (ST)

ANs

56.

OR

Element	Stack	Postfix
U		U
*	*	U
V	*	UV
+	+	UV*
R	+	UV*R
/	+/	UV*R
(	+/(	UV*R
S	+/(	UV*RS
-	+/(-	UV*RS
T	+/(-	UV*RST
)	+/	UV*RST-
	+	UV*RST-/
		UV*RST-/+

57. Translate, following infix expression into its equivalent postfix expression: ((A-B)*(D/E))/(F*G*H)

Ans 57. Equivalent postfix expression:

=((A-B)*(D/E))/(F*G*H)

=((AB-)*(DE/))/(FG*H*)

=AB – DE /* FG* H*/

58. Translate, following infix expression into its equivalent postfix expression: A*(B+D)/E-F-(G+H/K)

Ans 58. Equivalent postfix expression:

= A*(B+D)/E-F-(G+H/K)

= (A*(B+D)/E) – (F – (G + (H/K)))

= (A*(BD+)/E) – (F – (G + (HK/)))

= ((ABD+*) / E) – (F – (GHK/+))

= ABD+* E/F – GHK/+ -

59. Write the equivalent infix expression for 10,3,*,7,1,-,*,23,+

Ans 59. 10 * 3 * (7 – 1) + 23

60. Write the equivalent infix expression for a, b, AND, a, c, AND, OR.

Ans 60. a, b, AND, a, c, AND, OR

(a AND b), (a AND c), OR

(a AND b) OR (a AND c)

61. Evaluate the infix expression.

P: 12 , 7 , 3 , - , / , 2 , 1 , 5 , + , * , + ,)

Ans 61.

Symbol	Stack
12	12
7	12,7
3	12,7,3
-	12,4
/	3
2	3,2
1	3,2,1
5	3,2,1,5
+	3,2,6
*	3,12
+	15
)	15

62. Give postfix form of the following expression A*(B+(C+D)*(E+F)/G)*H

Ans 62. A*(B+(CD+EF+*)/G)*H

A*(B+CD+EF+*G/))*H

(A*(BCD+EF+*G/+))H

(ABCD+EF+*G/+*)*H

ABCD+EF+*G/+*H*

63. Give postfix form expression for: NOT A OR NOT B AND NOT C

Ans 63. =((A NOT) OR ((B NOT) AND (C NOT)))

=((A NOT) OR ((B NOT C NOT AND))

=A NOT B NOT C NOT AND OR

64. Consider the infix expression Q : A+B * C ↑ (D/E)/F. Translate Q into P, where P is the postfix equivalent expression of Q. what will be the result of Q if this expression is evaluated for A, B, C, D, E, F as 2, 3, 2, 7, 2, 2 respectively.

Ans 64. P = ABCDE/^*F/+

2,3,2,7,2
2,3,2->7/2->3
2,3,2,3
2,3->2^3->8
2,3,8
2->3*8->24
2,24,2
2->24/2->12
2,12
2+12
Result of evaluation = 14

65. Change the following infix expression into postfix expression.

(A+B) *C+D/E-F

Ans 65. Equivalent postfix expression:

= (A+B) *C+D/E-F
= (((A+B)*C) + (D/E)) – F
= (((AB+)*C) + (DE/)) – F
= AB+ C* DE/ + F-

66. Evaluate the following postfix expression. Show the status of stack after execution of each operation separately;

F, T, NOT, AND, F, OR, T, AND

Ans 66. F, T, NOT, AND, F, OR, T, AND

Scanned Element	**Operation**	**Stack Status**
F	Push	F
T	Push	F, T
NOT	Pop one operand from stack NOT T = F	F
	Push	F, F
AND	Pop two operands from stack F AND F = F	
	Push	F
F	Push	F, F
OR	Pop two operands from stack F OR F = F	
	Push	F
T	Push	F, T
AND	Pop two operands from stack F AND T = F	
	Push	

	Pop all	F

Result F

67. Evaluate the following postfix expression. Show the status of stack after execution of each operation separately;
T, F, NOT, AND, T, OR, F, AND

Ans 67. T, F, NOT, AND, T, OR, F, AND

Scanned Element	**Operation**	**Stack Status**
T	Push	T
F	Push	T, F
NOT	Pop one operand from stack	T
	NOT F = T	
	Push	T, T
AND	Pop two operands from stack	
	T AND T = T	
	Push	T
T	Push	T, T
OR	Pop two operands from stack	
	T OR T = T	
	Push	T
F	Push	T, F
AND	Pop two operands from stack	
	T AND F = F	
	Push	F

Result F

68. Evaluate the following postfix expression. Show the status of stack after execution of each operation:
5, 2, *, 50, 5, /, 5, –, +

Ans 68. 5, 2, *, 50, 5, /, 5, –, +

Scanned Element	**Stack Status**
5	5
2	5.2
*	10
50	10, 50
5	10, 50, 5
/	10, 10
5	10, 10, 5
–	10, 5
+	15

69. Evaluate the following postfix expression. Show the status if stack after execution of each operation;
60, 6, /, 5, 2, *, 5, –, +

Ans 69. 60, 6, /, 5, 2, *, 5, –, +

Scanned Element	**Stack Status**
60	60
6	60, 6
/	10

5	10, 5
2	10, 5, 2
*	10, 10
5	10, 10, 5
–	10, 5
+	15

70. Evaluate the following postfix expression using a stack and show the contents of stack after execution of each operation;

5, 3, 2, *, 4, 2, /, –, *

Ans 70. 5, 3, 2, *, 4, 2, /, –, *

Scanned Element	**Stack Status**
5	5
3	5, 3
2	5, 3, 2
*	5, 6
4	5, 6, 4
2	5, 6, 4, 2
/	5, 6, 2
	5, 4
*	20

71. Evaluate the following postfix notation. Show status of stack after every step of evaluation (i.e. after each operator):

False, NOT, True, AND, True, False, OR, AND

Ans 71. False, NOT, True, AND, True, False, OR, AND

Scanned Element	**Operation**	**Stack Status**
False	Push onto stack.	False
NOT	Pop one element from the stack i.e. False. Apply NOT on	
False	we get true, push onto stack.	
True	Push onto stack.	True, True
AND	Pop two elements from the stack. Apply AND between them. Push result onto stack.	
True	Push True onto stack	True, True
False	Push False onto stack.	True, True, False
OR	Pop two elements from the stack. Apply OR between them and push result onto stack.	True, True
AND	Pop two elements from the stack. Apply and between them and push result onto stack.	True

72. Evaluate the following postfix notation, show status of stack of every step of evaluation (i.e. after each Operator):

True, False, NOT, AND, False, True, OR, AND

Ans 72. True, False, NOT, AND, False, True, OR, AND

Scanned Element	**Operation**	**Stack Status**
True	Push onto stack.	True
NOT	Pop one element from the stack. Apply NOT on False and push onto stack.	True, False
AND	Pop two elements from the stack. Apply AND between them. Push the result onto stack.	True
False	Push False onto stack.	True, False
True	Push True onto stack	True, False, True
OR	Pop two elements from the stack. Apply OR between them and push result onto stack.	True, True
AND	Pop two elements from the stack. Apply and between them and push result onto stack.	True

73. Evaluate the postfix notation of expression:

50, 60, +, 20, 10, –, *

Ans 73. 50, 60, +, 20, 10, –, *

Scanned Element	**Stack Status**
50	50
60	50, 60
+	110
20	110, 20
10	110, 20, 10
–	110, 10
*	1100
	Pop all

74. Evaluate the following postfix notation of expression:

True, False, NOT, AND, True, True, AND, OR

Ans 74. True, False, NOT, AND, True, True, AND, OR

Scanned Element	**Stack Status**
True	True
False	True, False
NOT	True, True
AND	True
True	True, True
True	True, True, True
AND	True, True
OR	True

75. Evaluate the following postfix notation of expression:
False, True, NOT, OR True, False, AND, OR

Ans 75. False, True, NOT, OR True, False, AND, OR

Scanned Element	**Stack Status**
False	False
True	False, True
NOT	False, False
OR	False
True	False, True
False	False, True, False
AND	False, False
OR	False

76. Evaluate the following postfix notation of expression. Show the status of stack after each expression:
True, False, NOT, OR, False, True, OR, AND

Ans 76. True, False, NOT, OR, False, True, OR, AND

Scanned Element	**Stack Status**
True	True
False	True, False
NOT	True, True
OR	True
False	True, False
False	True, False
True	True, False, True
OR	True, True
AND	True

77. Convert the following infix expression to its equivalent postfix expression. Showing stack contents for the conversion:
(X – Y / (Z + U)* V)

Ans 77. Let us rewrite like (X – Y / (Z + U) * V

Scanned Element	**Stack Status**	**Expression**
(	(	
X	(	X
–	(–	X
Y	(–	XY
/	(	

4 Marks Questions OF STACK AND QUEUE

78. Write the definition of a member function Pop() in C++, to delete a book from a dynamic stack of TEXTBOOKS considering the following code is already included in the program.

```
struct TEXTBOOKS
{
char ISBN[20]; char TITLE[80];
```

```
TEXTBOOKS *Link;
};
class STACK
{
TEXTBOOKS *Top;
public:
STACK() {Top=NULL;}
void Push();
void Pop();
~STACK();
};
```

Ans 78.

```
void STACK::POP()
{
if (Top!=NULL)
{
TEXTBOOKS *Temp;
Temp=Top;
cout<<Top->ISBN<<Top->TITLE<<"deleted"<<endl;
Top=Top>
Link;
delete Temp;
}
else
cout<<"Stack Empty"<<endl;
}
```

79. Write the definition of a member function PUSH() in C++, to add a new book in a dynamic stack of BOOKS considering the following code is already included in the program:

```
struct BOOKS
{
char ISBN[20], TITLE[80];
BOOKS *Link;
};
class STACK
{
BOOKS *Top;
public:
STACK()
{Top=NULL;}
void PUSH();
void POP();
~STACK();
};
```

Ans 79.

```
void STACK::PUSH()
{
BOOKS *Temp;
Temp=new BOOKS;
gets(Temp->ISBN);
gets(Temp->TITLE);
Temp->Link=Top;
Top=Temp;
```

```
}
```

80. Write a complete program in c++ to implement a dynamically allocated Stack containing names of Countries.

Ans 80.

```
#include<iostream.h>
#include<stdio.h>
struct Node
{       char Country [20] ; Node *Link; };
class Stack
{ Node *Top;
public:
Stack( )
{ Top = NULL; }
void Push() ;
void Pop() ;
void Display() ;
~Stack () ;
};
void Stack::Push( )
{
Node *Temp = new Node;
gets(Temp -> Country);
Temp -> Link = Top;
Top = Temp;
}
void Stack::Pop( )
{
if (Top !=NULL)
{
Node *Temp = Top;
Top = Top -> Link;
delete Temp;
}
else
cout<<“stack Empty”;
}
void Stack::Display( )
{
Node *Temp = Top;
while (Temp! = NULL)
{
cout<<Temp -> Country <<endl;
Temp = Temp -> Link;
}
}
Stack::~Stack ( )
{
while (Top!=NULL)
{ NODE *Temp=Top;
Top=Top->Link;
delete Temp;
```

```
}
}
void main ( )
{ Stack ST;
char Ch;
do
{ cout<<“p/O/D/Q” ;
cin>>Ch;
switch (Ch)
{
case ‘P’ : ST.Push( ); break;
case ‘O’ :ST.Pop(); break;
case ‘D’ :ST.Disp();
}
} while (Ch!=’Q’);
}
```

81. Write a complete program in C++ to implement a dynamically allocated Queue containing names of Cities.

Ans 81.

```
#include <iostream.h>
#include <conio.h>
struct NODE
{ char City[20];
NODE *Next;
};
class Queue
{ NODE *Rear,*Front;
puplic:
Queue( )
{ Rear=NULL;Front=NULL;
}
void Qinsert( );
void Qdelete( );
void Qdisplay( );
~Queue( );
} ;
void Queue::Qinsert( )
{
NODE *Temp;
Temp=new NODE;
cout<<”Data:”;
gets (Temp->City);
Temp->Next=NULL;
if (Rear==NULL)
{
Rear=Temp;
Front=Temp;
}
else
{
Rear->Next=Temp;
Rear=Temp;
```

```
}
}
void Queue::Qdelete( )
{
if (Front!=NULL)
{
NODE *Temp=Front;
cout<<Front->City<<"Deleted \n";
Front=Front->Next;
delete Temp;
if (Front==NULL)
Rear=NULL;
}
else
cout<<"Queue Empty..";
}
Queue:: Qdisplay( )
{ NODE *Temp=Front;
while (Temp!=NULL)
{
cout<<Temp->City<<endl;
Temp=Temp->Next;
}
}
Queue:: ~Queue( )//Destructor Function
{ while (Front!=NULL)
{ NODE *Temp=Front;
     Front=Front->Next; delete Temp;
}
}
void main( )
{ Queue QU;
char Ch;
do
{
:
:
} while (Ch!='Q');
     }
```

82. Write a function QUEINS( ) in C++ to insert an element in a dynamically allocated Queue containing nodes of the following given structure:

```
struct Node
{ int PId ; //Product Id
char Pname [20] ;
NODE *Next ;
} ;
```

ANs 82.

```
void QUEINS (Node *&Front, Node *&Rear)
{ Node *Temp = new Node;
cin>>Temp->PId;
gets (Temp->Pname);
```

```
//or cin>>Temp->Pname;
//cin.get1ine(Temp->Pname);
Temp->Next = NULL;
if(Rear == NULL)
Front = Temp;
e1se
Rear -> Next = Temp;
Rear = Temp;
}
```

83. Write a function QUEDEL( ) in C++ to display and delete an element from a dynamically allocated Queue containing nodes of the following given structure:

```
struct NODE
{
int Itemno;
char Itemname[20];
NODE *Link;
} ;
```

Ans 83.

```
class Queue
{ Node *Front, *Rear;
public:
QUEUE( )//Constructor to initialize Front and Rear
{
Front = NULL;
Rear = NULL;
}
void QUEINS( );      //Function to insert a node
void QUEDEL( );      //Function to delete a node
void QUEDISP( );     //Function to display nodes
~Queue();            //Destructor to delete all nodes
};
void Queue::QUEDEL( )
{ if (Front!=NULL)
{NODE *Temp=Front;
cout<<Front->Itemno<<" ";
cout<<Front->Itemname<<"Deleted";
Front=Front->Link;
delete Temp;
if (Front NULL)
Rear=NULL;
}
else
cout<<"Queue Empty..";
}
```

84. Write a function in C++ to **delete** a node containing Book's information, from a **dynamically allocated Stack** of Books implemented with the help of the following structure.

```
struct Book
{
int BNo ;
char BName[20] ;
```

```
     Book *Next ;
     } ;
```
Ans 84.
```
     struct Book
     {
     int BNo ;
     char BName[20] ;
     Book *Next ;
     } ;
     class Stack
     {
     Book *Top;
     public:
     Stack( )
     {
     Top = NULL;
     }
     void Push( );
     void Pop( );
     void Display( );
     };
     void Stack::Pop( )
     {
     Book *Temp;
     if( Top= = NULL)
     cout<<"Stack Underflow…";
     else
     {
     cout<<"\nThe Book number of the
     element to delete: "<<Top->BNo;
     cout<<"\nThe Book name of the
     element to delete: "<<Top->BName;
     Temp=Top;
     Top=Top->Next;
     delete Temp;
     }
          }
```

85. Write a function in C++ to perform Insert operation in dynamically allocated Queue containing names of students.
```
     Struct NODE
     { char Name[20];
     NODE *Link;
     };
```
Ans 85.
```
     class Queue
     {    NODE *front,*rear;
     public:
     Queue( )
     {    front = rear = NULL;     }
     void Insert( );
     void Delete( );
```

```
void Display( );
};
void Queue::Insert( )
{
NODE *ptr;
ptr=new NODE;
if(ptr= = NULL)
{    cout<<"\nNo memory to create a
new node….";
exit(1);
}
cout<<"\nEnter the name….";
gets(ptr->Name);
ptr->Link=NULL;
if(rear= = NULL)
front=rear=ptr;
else
{
Rear->Link=ptr;
rear=ptr;
}
}
```

86. Write a function in C++ to perform a PUSH operation in a dynamically allocated stack considering the following :

```
struct Node
{
int X,Y ;
Node *Link ;
} ;
class STACK
{
Node *Top ;
public :
STACK( )
{Top = Null ;}
void PUSH( ) ;
void POP( ) ;
~STACK( ) ;
} ;
```

ANs 86.

```
struct Node
{ int X,Y ;
Node *Link ;
} ;
class STACK
{ Node *Top ;
public :
STACK( )
{ Top = NULL;
}
void PUSH( ) ;
```

```
void POP( ) ;
~STACK( ) ;
} ;
void STACK::PUSH( )
{ Node *Temp;
Temp=new Node;
if(Temp= =NULL)
{ cout<<"\nNo memory to create the node…";
exit(1);
}cout<<"Enter the value of X and Y";
cin>>Temp->X>>Temp->Y;
Temp->Link=Top;
Top=Temp;
}
```

87. Write a function QINSERT () in C++ to perform insert operation on a Linked Queue which contains client no and client name. Consider the following definition of NODE in the code of QINSERT()

```
  struct Node
     {  long int Cno;     //Client No
     char Cname[20];   //Clieht Name
     NODE *Next;

     };
```

Ans 87.

```
void QINSERT()
{  NODE *P=new NODE();
 cout<<"Enter the client number";
 cin>>P->Cno;
 cout<<"enter the client name";
 gets(P->Cname);
 P->Next=NULL;
 if(front==NULL && rear==NULL)
 { front=P;
  rear=P;
  }
  else
  { rear->Next=P;   rear=P;   }  }
```

88. Write a function PUSHBOOK() in C++ to perform insert operation on a Dynamic Stack, which contains Book_no and Book_Title. Consider the following definition of NODE, while writing your C++ code.

```
     struct NODE
      { int Book_No;
      char Book_Title[20];
       NODE *Next;
     };
```

Ans 88.

```
void PUSHBOOK(NODE *top)
```

```
{ NODE *NEW=new NODE;
  cout<<"Enter the book number";
  cin>>NEW->Book_No;
  cout<<"Enter book title";
  gets(NEW->Book_Title);
  NEW->Next=NULL;
  if(top==NULL)
  top=NEW;
  else { NEW->Next=top; top=NEW;}
  }
```

89. Write a function POPBOOK() in C++ to perform delete operation on a Dynamic Stack, which contains Book_no and Book_Title. Consider the following definition of NODE, while writing your C++ code.

```
struct NODE
 { int Book_No;
 char Book_Title[20];
  NODE *Link;
};
```

Ans 89.

```
void POPBOOK(NODE *top)
{
  cout<<"deleting top element from stack\n";
  cout<<"Book No"<<top->Book_No<<endl;
  cout<<"Book title"<<top->Book_Title<<endl;
  NODE *temp=top;
  top=top->Link;
  delete(temp);
  }
```

90. Write a function in C++ to perform a DELETE operation in a dynamically allocated queue considering the following description:

```
struct Node
{         float U,V;
     Node *Link;
};
class QUEUE
{    Node *Rear, *Front;
public:
     QUEUE( ) { Rear =NULL; Front= NULL;}
     void INSERT ( );
     void DELETE ( );
     ~QUEUE ( );
};
```

Ans 90.

```
void DELETE()
{    Node *temp;
```

```
          if(front==NULL)                    // No element in
     the queue
          {    cout<<"UNDERFLOW……..";
          }
          else
          {    temp=front;
          front=front->Link;// Making the second node as the
          first one
               temp->Link = NULL;
               delete temp;   // deleting the previous first
          node.
          }
     }
```

91. Define stackpop( ) to delete nodes, for a linked list implemented stack having the following structure for each node:

```
     struct Node
     {    char name[20];
          int age;
          Node *Link;
     };
     class STACK
     {    Node * Top;
     public:
          STACK( ) { TOP=NULL;}
          void stackpush( );
          void stackpop( );
          ~STACK( );
     };
```

Ans 91.

```
     void stackpop( )          // Pop from the beginning
     {    Node *temp;
          if(top==NULL)
          {    cout<<"UNDERFLOW…….";
          }
          else
          {    temp=Top;
               Top=Top->Link;
               temp->Link = NULL;
               delete temp;
          }}
```

92. Write an algorithm to insert an element from a linked queue depending upon user's choice.

Ans 92.

Algorithm for user choice:

1. ch=0 // Initialize a variable for user choice
2. Read ch //Enter, user choice 1 for push and 2 for pop
3. If(ch == 1) then /* if top is at end of Array-Queue */

4. call insert function
5. Else
6. call delete function
7. End

Algorithm for inserting in Linked-Queue:

/* Allocate space for ITEM to be inserted */
1. NEWPTR = new node
2. NEWPTR -> INFO = ITEM; NEWPTR -> LINK=NULL
/* Insert in the queue */
3. If rear = NULL Then
{
4. front = NEWPTR
5. rear = NEWPTR
6. else
{
7. rear -> LINK = NEWPTR
8. rear=NEWPTR
}
9. END.

93. Write an algorithm to delete an element from a linked queue depending upon user's choice.

Ans 93.

Algorithm for user choice:

1. ch=0 // Initialize a variable for user choice
2. Read ch //Enter, user choice 1 for push and 2 for pop
3. If(ch == 1) then /* if top is at end of Array-Queue */
4. call insert function
5. Else
6. call delete function
7. End

Algorithm for deleting in Linked-Queue:

1. If front==NULL Then
2. Print "Queue Empty"
3. Else
{
4. ITEM=front->INFO
5. If front=rear Then
{
6. front=rear=NULL
}
7. Else
8. front = front + 1
}
9. END

95. Write an algorithm to insert an element in a circular queue implemented as an array.

Ans 95.

Let Q be a Queue having size N. DATA be the element to be inserted. F and R denote front and rear positions in the queue.

```
1. If (R=N-1) then R=0
   else
   R=R+1
2. If (F=R)
   { write ('Queue overflow')
     Goto step 5
   }
3. Q[R]=DATA
4. If (F=-1)then
   F=F+1
5. End
```

96. Write an algorithm to deleting an element from a circular queue implemented as an array.

Ans 96.

Let Q be a Queue having size N. DATA be the temporary variable which stores the deleted element (if possible). F and R denote front and rear positions in the queue.

```
1. If (F<0) then
   {
    write("Queue Underflow")
     goto step 4
   }
2. DATA=Q[F]
3. If(F=R) then
   { F=R=-1 }
   else
   {
    if(F=N-1)
    F=0
    else
    F=F+1
   }
4. End
   ------------------------------------------------
```