# 1. Reverse string

Step 1:- Start

Step 2:- Read str [50],revstr[50]

Step :- *strptr = str

step 4:- *revptr =revstr

Step5 :-len = -1

Step 6 :- Print the entered string str

step 7 :-while (*strptr)

Step 8 :-srptr++

Step 9 :- len ++

Step 10 :-Repeat step 11-14 while (len>=0)

Step 11:- Srptr--

step 12:- *revptr= strptr

Step 13 :- revptr++

step 14 :-len- -

Step 15 :- *rev ptr = '\0'

Step 16 :-Print the reverse of a string revstr

Step 17:- Stop

## 2. Implement Pattern matching algorithm

Step 1:-start

step 2:- Declare char txt [20], and Pat [20]

step 3 :- Read a,b, i,j

step 4:- Enter the string

Step 5:- Enter the pattern to find

Step 6 :- Assign the length of string to the Variable

step 7:-Assign the length of text to the Variable b

step 8 :- Check for (i=0,i<=a-b;i++) then

for (j=0; j<a;j++)

step 9:- If txt [+j] not equal to Pat [j],

then break the forloop, then goto step 12

step 10:- If Value of j is equal to the Value of a

step 11 :- Then print Pattern found at index j+2

step 12:- stop

## 3. Search in 2D Array

Step I:-Start

Step 2 :-Input m,n,i, j,srchno

Step 3:-'Set Count=0, a[50] [S0]

Step 4 :- Repeat steps 5-6 while (i<m)

step 5 :-Goto step 6 while (i<n)

Step G :- a[i][j] an array is created

step 7 :-Enter the search number (srchno)

Step 8 :- Repeat the steps 9-10 while (i<m)

Step 9 :-Goto step 10 while (j<n)

Step 10 :- check a[i][j]= srchno

Step 11 :-If true print the index position

Step 12 :- lf (count ==0) Print "not found"

step 13 :- stop

## 4. Appending of Arrrays

step 1:-start

Step 2 :-Accept Variables m,n to step size of 1st and 2nd array

step 3:- Input elements into ar[50] & br[50]

step 4 :- Repeat Step 5 while (i <m)

Step 5 :- cr[k] = ar[i

Step 6:- Repeat step 7 while (j<n)

Step 7 :- cr [k+j]=br [j]

Step 8:- Print appended array by using for loop until j(m +n) Satisfies

step 9:- stop

## 5.Binary search

Step1:-Start

step 2:-Accept a Value in max as the no.. of element of the array

Step 3 :- Accept max element ito the array list

Step 4 :-Accept the Value to be searched in the Variable item

step 5 :- store the Position of Ist element of the list in first and that of last in last

Step 6:-Repeat step 7 to 11 till while (first < = last)

Step 7:- Find middle Position using the formula (First + last)/2 and store it in middle

step 8:- Compare the search Value in item with the element at the middle of the list

step 9:-If the middle element contains the search Value in item then stop search,display the Position and goto step 12

Step 10 :- If search Value is smaller than the middle element then set last =middle-1

Step 11 - If the Search Value is larger than middle element then set First = middle +1

Step 12 :-stop

## 6. SParse Matrix

Step 1:-start

step 2 :- Read m,n to stop rder of mmatrin

Step 3 :- input element into matrix ar [10] [10]

Step 4 :-Print entered matrix

step 5:-Repeat step 6-10 while (i<m)

step 6:- Repeat step 7-10 wbile (i<n)

step 7:- If (ar [i][j] !=0)

step 8:- br [s] [0] = i

Step 9:- br  [s] [i] = j

Step 10 :- br [s] [2]=ar [i] [j]

step 11 :- continue step 12,13 while (i<s)

Step 12 :- Goto step 13 While (i<3)

step 13:-Print br [i][j]

Step 14:-stop

## 7. Create a singly linked list of N nodes

Step 1:- Start

Step 2:- Include all the header files which are used in the Program

Step 3:-Declare all the user defined functions

Step 4:- Define a node structurre

Step 5:- Stop

Void (create C)

Step 1:- start

step 2:- create a new node with given Value

Step 3 :- check whether list is empty

(st node = NULL)

step 4 :--if it in NULL, then Print memory "Cannot be allocated "

step 5:-else

add data for the First node and same in num

Step 6 :- set st Node -> data=n Data;

St Node -> hextptr= NULL;

nd Buffer=st Node

Step 7:-Repeat steps 8 to 12 (While i<=n)

Step 8:-create next node with given Value

Step 9:- check whether list is empty (n Node = NULL)

step 10:- If it is empty then Print memory Cannot be allocated"

Step 11:- else enter the data for hexthode

Step 12 :-n Node -> data = n Data

n Node -> nextptr = NULL

nd Buffer -> next ptr =nNode

nd Buffer= nd Buffer -> nextptr

Step 13 :- stop

Void display()

Step 1:- start

step 2:- Declare the variables

Step 3:- check whethes the list is empty

(ndBuffer = =NULL)

Step 4 :-Print list is empty

Step 5:- else

step 6 :-Repeat step1 while (nd Buffer!==NULL)

Step 7:- Print Data = % d, nd Buffer->data

ndBuffer =ndBuffer -> nextptr

Step 8 :- stop

Display list ()

Step 1:- start

Step 2:- Input the numbers of Nnodes list num

step 3:- create

Step 4 :- Print Data entered in the list

Step 5:- stop

## 8. Deletion from Singly Linked List

Step 1:- Start

step 2:- Include all the header files which are used in the progoam

Step 3 :-Declare all the user defined functions

Step 4 :- Define a node structure

Step 5 :- Stop

Void (create C)

Step 1:- start

Step 2:- create a newnode with given Value

Step 3 :- check whether list is empty (stnode=NULL)

step 4:-if it is NULL, then Print "memory cannot be allocated"

step 5 :-else

add data to the first node and save in num

Step 6 :- set stnode -> num =num

stnode -> nextPtr=NULL

temp = stnode

step 7:-Repeat steps 8 to 12 While(i<=n)

step 8 :- create next node with given Value

step 9:- check whether list is Empty (fnnode==NULL)

step 10 :- If it is empty then Point "memory cannot be allocated"

Step 11:- else

enter the data for next node.

Step 12 :- set

fnnode -> num=num

Fnnode-> next Ptr = NULL

tmp -> next Ptr =fnnode

tmp = tmp -> nextptr

Step 13 :-stop

Void (delete C)

Step 1:- start

Step 2:- Declare the Variables whether list is empty (stnode ==NULL)

step 3 - check

Step 4 :- if it is empty ,then Print "There is no nodes in the list"

Step 5:- else, define two Node pointers to del and Prenode, and

set Prenode with stnode and Repeat step 6&7 while (i<=Pos)

Step 6:-set Prenode =todel

todel = todel -> next Ptr

Step 7:- If todel = = NULL then break.

step 8:-If todel==stnode then set

Sthode == stnode -> nextptr

Preode -> next Ptr=todel ->nextptr

todel->nextptr=NULL

free (todel)

step 9:-else Print" Deletion Cannot be Possible from that Position"

step 10: stop

Void display()

Step 1:- start

Step 2:- check whether the list is Empty (stnode==NULL)

step 3 :- If it is empty,then point "No data found in the list"

step 4 :- else set tmp = stnode

step5:- Repeat step 6 and 7 while (tmp!=NULL)

step 6 :- Print the elements

step 7:- set tmp=tmp -> nextptr

step 8:- stop

Void main ()

Step 1:-start

step2:- Declare the Variables number of nodes, save in n

step 3:- Enter number of nodes, save in n

step 4 :- call create function will argument n

step 5:- Point" Data entered in the list"

step 6 :-Call dis play function

step 7:-Enter the Position of node to delete and save in Pos

step 8 :- check whether (Pos <=1 0r Pos >=n)

step 9:-If it is tue then Print "Deletion Cannot Possible friom that Position"

Step 10 :- check whether CPos>I and posan)

step 11:- 1f it is true then Print" Deletion Completed successfully"

step 12:- call delete function with argument Pos

Step 13:- Print the new list

step 14 :-Call display function

Step 15:- stop

## 9. Doubly linked list

Step 1:-Start

Step 2:-Struct node * lptr

step 3 :-Read num to Store data

Step 4:-struct node* head 1,Create stuct node Struct node** tail 1, int dat

Step 5:- Print struct node newnode, temp

step 6:-is newnode =(struct node*) malloc(size of Cstruct node),newnode- > data = dat

Step 7:- newnode->rptr =NULL

Step 8:-newnode ->lptr = NULL

step 9:- stop

Case 1: -head not equal to null

step 1 :- start

step 2 :- If (head!= NULL) repeat steps 3 to 10

Step 3:- Print head !=newnode

Step 4 :- temp head 1

Step 5 :- Repeat step 5 to 9 while (temp ->rptr= N ULL)

Step 6:-Print temp = temp -> rptr

step 7:- temp = newnode-> lptr

Step 8:- newnode -> rptr=NULL;

step 9 :-*tail= newnode then temp->rptr

step 10 :-return head I

Step 11 :-stop

Case 2: Read head and tail

step 1:- start

step 2:- while (head !=NUL)

Step 3 :- Point head = head ->rptr

step 4:- while (tail != head)

Step 5 :-Print tail = tail->lptr

Step 6 :-If (tail== head)

step 7:-Print tail =tail ->lptr

step 8:- Stop

Case 3 display ()

Step 1:- start

step 2:- Accept the Variables i, n, value

step 3 :- stouct node* head, tail

step 4 :- Print "Enter the limit"

Step 5:-Read n

step 6 :-Repeat step 1 to 9 while (i<n)

Step 7 :- Print "Enter the no."

Step 8:-Read Value

Step 9 :- head = Create (head, &tail, Value)

Step 10 :- Print the data in the forward direction is Printed below"

Step 11 :-write head

Step 12 :-Point" The data in the backward direction is Printed below"

Step 13:-display tail, head

Step 14 :-Stop

**10. Implement stack using array**

step 1: start step
2: set top= -1
step 3: Accept the variable n to store size of stack
step 4: Reach choice step 5: if(choice == 1)
push()
step 6: else if(choice == 2)
call pop function step 7: else
if(choice == 3)
display()
step 8: else if(choice == 4)
print “exit point” step 9: else
print ”Invalid choice” step 10: Repeat
step 5-9 while(choice !=4) step 11: stop

**<u>Push()</u>** step 1:
start step 2:
if(top<n-1)
Read the number to be pushed and stored in variable val
step 3: top++ step 4: stack [top]=val step 5: else step 6:
print ”stack over flow” step 7: stop

**<u>Pop()</u>** step 1: start step 2:
if(top>-1) return stack
[top] top……….. step 3:
else print “stack
underflow”
step 4: stop

**<u>display()</u>** step 1: start step 2:
if(top>=0) step 3: Repeat step 4
while ( i>=0) step 4: print stack [ i]
step 5: else print “stack is
empty”
step 6: stop

## 11.Implement stack using linked list

step 1: start step 2: Accept
variable choice
step 3: if (choice ==1)
push(val) step 4: else if
(choice ==2)
pop()
step 5: else if(choice == 3)
display()
step 6: else if(choice == 4) print "exit point"
step 7: else print "invalid choice" step 8:
Repeat steps from 3 to 7 while (choice == 4) step
9: stop

**<u>Push()</u>** step 1: start
step 2: if(top == NULL)
step 3: top =(struct node 4) malloc(1*size of( struct node))
step 4: top → ptr=NULL
step 5: top → info=a step
6: else
step 7: temp=(struct node*)malloc(1*size of( struct node))
step 8: temp→ info=a step 9: temp→ ptr=NULL step 10:
top = temp step 11: count ++
step 12: stop

**<u>Pop()</u>** step 1: start step 2: set
top1=top step 3: if (top1 ==
NULL) step 4: print "stack
underflow" step 5: else step 6:
top=1=top1→ptr step 7:
return top→ info step 8:
free(top) step 9: top = top1
step 10: count - - step 11: stop

**<u>display()</u>** step 1: start
step 2: set top1=top
step 3: if (top1 == NULL)
step 4: else
step 5: Repeat 6 and 7 while(top1 != NULL)
step 6: return top1→info step 7: top1=
top1→ptr step 8: stop

## 12.Evaluation of postfix expression

Step 1: start Step 2: initialize the stack

Step 3: Scan the given expression from left to right.

Step4: a) If the scanned character is an operand, push it into the stack.

b) If the scanned character is an operator, POP 2 operands from stack and perform operation and PUSH the result back to the stack.

Step 5: Repeat step 3 till all the characters are scanned.

Step 6: When the expression is ended, the number in the stack is the final result.

Step 7: stop

## 13.Implement Queue using array

step 1: start step 2: input
variable choice step 3:
if(choice == 1)
insert()
step 4: else if(choice == 2)
del()
step 5: else if(choice == 3)
display()
step 6: else if(choice == 4)
print “exit point” step 7: else
print ”Invalid choice” step 8: Repeat
step 3-7 while(choice !=4) step 9: stop

**insert()** step 1: start step 2: if(rear==n)
print “overflow” step 3: else step 4: if
(front == -1) step 5: set front =0 step 6:
accept variable val to input element step 7:
queue [rear]=val step 8: increment rear
step 9: stop

**del()** step 1: start step 2: if(front =
-1 && front<rear) step 3: return
queue[front] step 4: front ++ step
5: else print “queue
underflow” step 6: stop

**display()** step 1: start step 2:
if(front == rear)
print ”Queue is empty” step
3: else step 4: print element
of queue step 5: stop

## 14.Impelement Queue using linked list

Step 1: start
Step 2: Include all the header files which are used in the program. And declare all the user defined functions.
Step 2 :Define a 'Node' structure with two members data and next
Step 3 : Define two Node pointers 'front' and 'rear' and set both to NULL.
Step 4 : Implement the main method by displaying Menu of list of operations and make suitable **function calls in the main method to perform user selected operation.** Step 5: stop

**Insert()**

Step 1: start
Step 2:Create a newNode with given value and set 'newNode → next' to NULL.
Step 3: Check whether queue is Empty (rear == NULL)
Step 4: If it is Empty then, set front = newNode and rear = newNode.
Step 5: If it is Not Empty then, set rear → next = newNode and rear = newNode. Step 6:stop

**delete()**

Step 1: start
Step 2: Check whether queue is Empty (front == NULL).
Step 3 : If it is Empty, then display "Queue is Empty!!! Deletion is not possible!!!" and terminate from the function
Step 4: If it is Not Empty then, define a Node pointer 'temp' and set it to 'front'.
Step 5: Then set 'front = front → next' and delete 'temp' (free(temp)). Step 6: stop

**display ()**

:

Step 1 :start
Step 2: Check whether queue is Empty (front == NULL).Step 2 - If it is Empty then, display 'Queue is Empty!!!' and terminate the function.
Step 3 : If it is Not Empty then, define a Node pointer 'temp' and initialize with front.
Step 4 : Display 'temp → data --->' and move it to the next node. Repeat the same until 'temp' reaches to 'rear' (temp → next != NULL). Step 5 : Finally! Display 'temp → data ---> NULL'.
Step 6: stop

**15.Search an element in a binary search tree**

step 1: start

step 2: Allocate the memory for free step 3: set the data part to the value and set the left and right pointer of tree, point to NULL. step 4: if the item to be inserted, will be the first element of the tree, then the left and right of this node
will point to NULL.
step 5: else check if the item is less than the root element of the tree if this is true then recursively perform this operation with the left of the root. step 6: if this false, then perform this operation recursively with the right sub-tree of the root.
step 7:stop

## 16.implement exchange sort

step 1: start

step 2: Accept a value in n as no of elements

Step 3: Accept n element into array a

Step 4:Repeat step 5-7 (n-1) times

Step 5: Repeat step 6 until 2nd last element of the list. If they are not in proper order, swap the element

Step 6: starting from 1st position, compare 2 adjacent elements in the list if they are not in order, swap the element

Step 7: Revise the list by excluding last element in the current list

Step 8: print sorted array a

Step 9: stop

## 17.implement selection sort

step 1: start step 2: Accept 2 value in N as no of elements of array.
Step 3:Accept N elements into the array AR
Step 4: Repeat steps 5-9 (N-1) times
Step 5: Assume the $1^{st}$ element in the list as the smallest and store it in min and its position in pos
Step 6:Repeat step 7 until last element in the list
Step 7: compare the next element in the list with value of min. If it is found smaller, store it in min and position in pos
Step 8: if the $1^{st}$ element in the list and value in min are not same, then swap the $1^{st}$ element with the element at position pos
Step 9: Revise list by including $1^{st}$ element in the current list
Step 10: print sorted array ar
Step 11:stop

**18.implement insertion sort**

Step 1: Start
Step 2: Read n to store total no of elements
Step 3: input elements into array a[100]
Step 4: Repeat 5-10 while(i<n-1)
Step 5: set j= i
Step 6: Repeat 7-10 while (j>0 && a[ j-i]>a[j])
Step 7: temp=a[ j]
Step 8: a[ j]=a[ j-i]
Step 9: a[j-1]=temp
Step 10: j - -  Step
11: print a[i]
Step 12: stop