Insertion at begining

1. Start
2. Declare array A[n], and i and value
3. Read array
4. set i=n, Repeat steps 5 and 6 until i==0
5. A[i+1] = A[i]
6. Decrement i by one
7. Increment array size by one
8. A[0] = Value
9. Print new array
10. Stop

Insertion at Given position

1. Declare array A[n], i and values, and pos.
2. Read array
3. Read pos
4. Increment array size by one
5. i=n-1 repeat the step 6 and 7 until i==pos
6. A[i]= A[i-1]

7. Decrement i by one

8. A[pos] = value

10. print new array

Delete from Begining

1. Declare array A[n] and i
2. set i=0, Repeat steps 3 and 4 until i=n-1
3. A[i] = A[i+1]
4. increment i by one
5. Decrement n by one
6. print new array
7. stop

Delete from Given Position

1. Declare array A[n], i, pos
2. Read array, pos
3. Check pos > n
4. set i=pos-1, Repeat steps 5 and 6 until ~~i<(n-1)~~ i<(n-1)

6. $A[i] = A[i+1]$
7. increment i by one
8. Decrement n by one
9. print new array
10. stop.

Circular linked list

Step1 Start

Step2 Enter the value to be deleted

Step3 check if head = Null ~~then p~~ if it is print underflow or list empty

Step4 Declare temp = head and previous

Step5 If temp → data = key

Step 6 link previous → next with temp → next if there are more than one node in temp → next = Null otherwise assign prev → next = Null

Step7 set head = prev → next if temp == head

Step8 free temp

Step9 set temp = prev → next if temp != null otherwise assign Null

Step 10 If current node, does not contain key to delete, then simply update previous and current node. Say prev = temp and set temp = temp → next

Step11 repeat ~~St~~ step 5 to 10

~~5. free the temp~~

Circullar queue

Step 1 Start

2. declare the que and other variables

3. ~~Declare~~
Read the choice from the user

4. Read the element to be inserted from the user and call the enque function by passing the value

5. If front == -1 and rear == -1 then set front = 0 , rear = 0 and set queue [rear] = element

6. else if rear + 1 % max == front or front = rear+1 the print queue is overflow

7. else set rear = rear + 1 % max and set queue [rear] = element

Search

Step

1. Read the element to be searched in the que

2. check if item == queue [i] the print item found and its position i+1

3. If c == 0 then print item not found.

Doubly linked list

step 1. start

step 2. Create a new node

step 3. If head = = Null then

set new node → prev = null

new node → next = null

step 4. set head = new node

step 5. Define temp 1, temp 2

step 6. set temp 1 = head

step 7. If temp → data = key && temp → next = null

then temp → next = new node

new node → prev = temp

newnode → next = Null

step 8 else

while temp → data != key || temp → next !

= null

prev = temp

temp = temp → next

If temp → next = = key

new node → next = temp → next

newnode → next = temp → next

temp2 = temp → next

temp2 → prev = newnode

temp → next = newnode

If temp → data != key || temp → next = null

print: insertertion cannot be done as the node not found

Step9: Stop

<u>Deletion</u>

1. ~~Re~~ Start

2. Read the element to be deleted

3. Check if head == Null then print list is empty

4. Else set temp = head

5. while temp → data != item && temp → next != Null

6. check if temp → prev = null && temp → next = null then set head = null and free (temp), exit

~~7 exit~~

7. Else If temp → prev = null, head = temp → next
head → prev = null
free (temp), exit

8. If temp → next = = null
x = temp → prev
x → next = null, free (temp)
exit

9. Else ~~temp~~ x = temp → prev
x → next = temp → next
y = temp → next
y → prev = temp → prev
free (temp), exit

10. If temp → data != Item and temp → next = Null
print element not found.

11. Stop.

Merging

Step 1 : Start

Step 2 : Declare the variables

Step 3 : Read the size of 1st array

Step 4 : Read elements of first array in sorted order

Step 5 : Read the size of 2nd array

Step 6 : Read the elements of 2nd array in sorted order

Step 7 : Repeat step 8 and 9 while i<m & j<n

Step 8 Check if a[i] >= b[j] then c[k++] = b[j++]

Step 9 Else c[k++] = a[i++]

Step 10 Repeat step 11 while i<m

Step 11 c[k++] = a[i++]

Step 12 Repeat step 13 while j<n

Step 13 : c[k++] = b[j++]

Step 14 print 1st array

Step 15 print 2nd array

Step 16 print sorted array

Step 17 End

Stack operations

Step

1. Start
2. Declare the node and required variables
3. Declare the functions for Push, Pop, display and Search an element
4. Read the choice from the user

~~5. If the user~~ Push

5.1 Declare the new node & allocate memory for the new node

2. set newnode → data = value
3. check if top == null then set newnode → next = null
4. set new → next = top
5. set top = newnode & then print insertion is successful

6. ~~step~~ Pop

1. check if top == Null the print stack is empty
2. else declare a pointer variable temp and initialize it to top
3. Print the element that being deleted
4. Set temp = temp → next

8. free the temp

display

1. check if top == Null then print stack is empty

2. else declare temp and temp = top

3. Repeat steps below while temp → next != Null

4. print temp → data

5. temp = temp → next

search

1. Declare ~~to~~ ptr and initialize ptr = top

2. check if ptr = Null then stack is empty

3. else read the element to searched

4. Repeat 5 to 8 while ptr != null

5. check if ptr → data == item then print element found and set flag = 1

6. else set flag = 0

7. ptr = ptr → next

8. If flag == 0 then element not found

5 ~~free the temp~~

Circullar queue

Step 1 Start

2. declare the que and other variables

3. ~~Dellare~~
Read the choice from the user

4. Read the element to be inserted from the user and call the enque function by passing the value

5 If front = = -1 and rear = = -1 then set front = 0, rear = 0 and set queue [rear] = element

6 else if rear+1 % max = = front or front = rear+1 the print queue is overflow

7 else set rear = rear+1 % max and set queue [rear] = element

Search

Step

1. Read the element to be searched in the que

2. check if item = = queue [i] the print item found and its position i+1

3. If c = = 0 Then print item not found.

Set operations

union

step 1 Read the cardinality of 2 sets

2. check if m!=n then cannot perform union

3. Else read the elements in both the sets

4 Repeat step 5 to 7 while i<m

5 C[i]= A[i] | B[i]

6 print C[i]

7 i= i+1

intersection

1. Read the cardinality of 2 sets

2. If m!=n print cannot perform intersection

3. else read the elements in both the sets

4 Repeat 5 to 7 until i<m

5. C[i]= A[i] & B[i]

6 print C[i]

7 i= i+1

difference

1. If $m != n$ print cannot do difference

2. Read element in both sets

3. Repeat 4 - 6 until $i<m$

4. check
if $A[i] = 0$ then $C[i] = 0$

5. Else if $B[i]==1$ the $C[i]=0$

6 else $C[i]=1$

7 $i = i + 1$

8 Repeat 9 - 10 until $i<m$

9. print $C[i]$

10 $i = i+1$

BST

Insertion

1. pass the value to insert pointer and also root pointer
2. check if !root then allocate memory for the root
3. set the value to the info part of the root and then set left and righ part of the root
4. check if root → info > x then call the insert pointer to insert to left of the root
5. check if root → info < x then call the insert pointer to insert to the right of the root
6. Return the root

deletion

1. root ptr = root
2. if (! ptr) then print node not found
3. if ptr → info < x the call delete pointer by passing the right pointer and item
4. else if ptr → info > x then call the delete pointer by passing the left

pointer and item

5. Check if ptr -> info == item then check if ptr -> left == ptr -> right then free ptr and return null

6. Else if ptr -> left == Null then set ptr -> right and free ptr, return p1

7. while p1 -> left not equal to null, set p1 -> left ptr -> left and free ptr, return p2

9. Return ptr

Search

1 . Read the element to be searched

2. while ptr check if item > ptr -> info then ptr = ptr -> right

3. Else if item < ptr -> info the ptr = ptr -> left

4 Else break

5 check if ptr then print that the element is found

6. else print element not found in tree and return root

7 If the user choose to perform traversal call the traversal function and pass the root pointer

8 If root not equals to null recursively call the function by passing root → left

9. print root → info

10. Call the traversal function recursively by passing root → right

Disjoint set

Step

1. Start

2. Declare the structure and related structure variable

3. Declare a function makeset()

3.1 Repeat step 3.2 to 3.4 until i<n

3.2. dis.parent [i] is set to i

3.3 set dis. ~~~~ rank [i] ~~~~ = 0

3.4 increment i by 1

4. Declare a function display set

4.1 Repeat step 4.2 and 4.3 until i<n

4.2 print dis.parent[i]

4.3 i = i+1

4.4 Repeat step 4.5 and 4.6 until i<n

4.5 print dis.rank[i]

4.6 i = i + 1

5. declare function find and pass x to the function

5.1 check if dis parent [n] != x then set the return value to dis.parent [x]

5.2 return dis.parent [x]

6 Declare a function union and pass two variables x and y.

6.1 set x set to find (x)

6.2 set y set to find (y)

6.3 check if x set == y set then return return

6.4 check if dis.rank [xset] < dis.rank [yset] then

6.5 set yset = dis parent [yset]

6.6 set -1 to dis rank [x set]

6.7 else if check dis.rank [xset] > dis.rank [yset]

6.8 set x set to disparent [yset]

6.9. set -1 to dis.rank [y set]

6.10 Else dis.parent [y set] = x set

6.11 set dis.rank [x set]+1 to dis.rank [xset]

6.12 set -1 to dis.rank [y set]

7 Read the number of elements

8 call the function make set

9. Read the choice from user to perform union find and display operation

10. If the user choose to perform union operation read the element to perform union. then call the function to perform union operation

11. If the user choose to perform find operation read the element to check if connected

11.1 check if find (x) == find(y) then print connected component

11.2 Else print Not connected component

12 If the user choose to perform display operation call the function display set

13 End