Program No-1

Aim: merging

Program Algorithm

Step 1 : Start

Step 2 : Declare the variables

Step 3 : Read the size of first array

Step 4 : Read elements of first array in ~~sto~~ sorted order.

Step 5 : Read the size of second array

Step 6 : Read the elements of second array in sorted order

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

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

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

Step 10 : Repeat step 11 while i<m

Step 11 : c[k++] = a[j++]

Step 12 : Repeat step 13 while j<n

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

Step 14 : point the first Array

Step 15 : point the second array

Step 16 : point the merged Array

Step 17 : End

Program No - 2

Aim: Stack Operations.

Program

Step 1 : Start

Step 2 : Declare the node and the required variables

Step 3 : Declare the functions for push, pop, display, and search an element.

Step 4 : Read the choice from the user.

Step 5 : If the user choose to push an element, then read the element to be pushed & call the function to push the element by the function. passing the value to the function.

Step 5.1 : Declare the newnode & allocate memory for the newnode.

Step 5.2 : Set newNode → data = value

Step 5.3 : check if top = = null then set newNode → next = null

Step 5.4 : Set newnode → next = top

Step 5.5 - Set top = newnode & then point insertion is successful.

Step 6 : If user choose to pop an element from the stack then call the function to pop the element.

Step 6.1 : Check if top = = Null then point stack is empty.

Step 6.2 : Else declare a pointer variable temp and initialize it to top.

Step 6.3 : Point the element that being deleted

Step 6.4 : Set temp = temp → next

Step 6.5 : free the temp

Step 7 : If the user choose the display then call the function to display the element in the stack.

Step 7.1 - Check if top = = Null then point stack is empty.

Step 7.2 - Else declare a pointer variable temp & initialize it to top.

Step 7.3 - Repeat steps below while temp → next ! = null.

Step 7.4 - print temp→data

Step 7.5 - Set temp = temp→next

Step 8 - If the users choose to search an element from the stack then call the function to search an element

Step 8.1 - Declare a pointer variable ptr and other necessary variable.

Step 8.2 - Initialize ptr = top

Step 8.3 - Check if ptr = null then print stack empty

Step 8.4 - Else read the element to be searched.

Step 8.5 - Repeat step 8.6 to 8.8 while ptr != null

Step 8.6 - check if ptr→data == item then print element founded and to be located and set flag = 1

Step 8.7 - Else set flag = 0

Step 8.8 - Increment i by 1 and set ptr = ptr→next

Step 8.9 - Check if flag = 0 then print the ps element not found

Step 9 - End

Program No - 3

Aim: Circular Queue Operations.

Step 1 : Start

Step 2 : Declare the queue and other variables

Step 3 : Declare the functions for enques, dequeue search and display.

Step 4 : Read the choice from the user.

Step 5 : If the user choose the choice enqueue. then Read the element to be inserted from the user and call the enqueue function by passing the value.

Step 5.1 : check if front == -1 && rear == -1 then set front = 0 , rear = 0 and set queue [rear] = element

Step 5.2 : Else if rear+1 % max == front or front = rear + 1 then print queue is overflow

Step 5.3 : Else set rear = rear +1 % max and set queue [rear] = element.

Step 6 : If the user choice is the option dequeue then call the function dequeue.

Step 6.1 : Check if front == -1 and rear == -1 then print queue is underflow

Step 6.2 : Else check if front == rear then print the element is to be deleted then set front = -1 and rear = -1.

Step 6.3 : Else print the element to be dequeued
Set front = front + 1 % max

Step 7 : If the user choice is to display the queue then call the function display.

Step 7.1 : Check if front = -1 and rear = -1 then print queue is empty

Step 7.2 : Else repeat the step 7.3 while i <= rear
and set i - i + 1 % max

Step 7.3 : Print queue [i]

Step 8 : if the user choose the search then call the function to search an element in the queue

Step 8.1 : Read the element to be searched in the queue.

Step 8.2 : Check if item == queue [i] then print item found and its position and increment by 1

Step 8.3 : Check if c == 0 then print item not found

Step 9 : End.

Program No-4

Aim: Doubly linked list Operation.

Step 1 : Start

Step 2 - Declare a structure and related variables

Step 3 - Declare functions to create a node, insert a node in the begining at the end and given position, display the list and search an element in the list.

Step 4 - Define function to create a node, declare the required variables.

Step 4.1 - Set memory allocated to the node = temp then set temp → prev = null and temp → next = null. ~~and temp → ne~~

Step 4.2 - Read the value to be inserted to the node.

Step 4.3 - Set temp → n = dat and increament count by 1

Step 5 : Read the choice from the user to program different operation on the list

Step 6 - If the user choose to perform insertion Operation at the beginning then call the function to perform the insertion.

Step 6.1 : Check if head == null then call the function to create a node, perform step 4 to 4.3

Step 6.2 : Set head = temp and temp != head.

Step 6.3 : Else call the function to create a node, perform step 4 to 4.3 then set temp -> next = head, set head -> prev = temp and head = temp.

Step 7 : If the user choice is to perform insertion at the end of the list, then call the function to perform the insertion at the end.

Step 7.1 : check if head == null then call the function to create a newnode then set temp = head and then set head = temp1

Step 7.2 - Else call the function to create a newnode then set temp1 -> next = temp, temp -> prev = temp1 and temp1 = temp.

Step 8 : If the user choose to perform insertion in the list at any position then call the function to perform the insertion operation

Step 8.1 - declare the necessary variable.

Step 8.2 - Read the position where the node head to be inserted, set temp 2 = head

Step 8.3 - check if pos < 1 or pos >= count + 1 then print the position is out of range.

Step 8.4 - check if head == null and pos = 1 then print "Empty list cannot insert other than 1st position.

Step 8.5 - check if head == null and pos - 1 then call the function to create newnode, then set temp = head and head = temp 1

Step 8.6 - while i < pos then set temp 2 = temp2 → next then increment i by 1

Step 8.7 - Call the function to create a newnode and then set temp → prev = temp2 and temp → next = temp 2 → next → prev = temp.

step 9 - If the user choose to perform deletion operation is the list then all the function to perform the deletion operation.

step 9.1 :- declare the neseccary variables.

step 9.2 - Read the position where node need to be deleted set temp 2 - head

step 9.3 - check if pos < 1 or pos >= count + 1, then paint position out of range.

step 9.4 - check if head == null then paint the list is empty

step 9.5 - while if i < pos then temp 2 = temp 2 -> next and increment i by 1

step 9.6 : if i == 1

if temp 2 -> next == null

then

print "node deleteled

free (temp 2) set temp 2 = head = null

Step 9.7 - if temp2 ⇒ next == null

then

temp2 → prev → next = null

free (temp2)

print node deleted.

Step 9.8 - temp2 → next → prev ⇒ temp2 - prev

If i1 = 1

then

temp2 → prev → next = temp2 → next

Step 9.9 - if i == 1

head = temp2 → next

print "node deleted"

free (temp2)

count --

Step 10 : Display Operation.

- set temp2 = n.

if temp2 = null

print " list is empty"

while

temp2 → next != null

print

temp2 → n then

temp2 = temp2 ⇒ next

step 11 : Search Operation.

Set temp2 = head

if temp2 = = null

Point " the list is Empty"

step 12 : Read the value to be searched

While temp2 != null

if temp2 → n = = data

then

print "element found

position count + 1

else

~~set~~ temp2 = temp2 ⇒ next

count ++

Point "element not founded"

step 13 : End.

Program No-5

Aim : Binary Search Tree

Step 1 : Start

Step 2 : Declare a structure and structure pointers for insertion deletion and search operations and also declare a function for inorder traversal

Step 3 : Declare a pointer as root and also the required variable.

Step 4 : Read the choices.

Step 5 : Insertion.

check if !root then allocate memory for the root

Step 5.1 : Set the value to the info part of the root then set left and right part of the root to null and return root.

Step 5.2 : Check if root → info > x then call the insert pointer to insert to left of the root.

if root → info < x then call the insert
pointer to insert to left of the root.

Step 5.3 – Return the root.

Step 6 – Deletion Operation.

check if not ptr the point node
is not found.

else if
ptr → info < x //call delete
pointer by
passing the right
pointer and the
item.

else if ptr → info > x // then call
delete pointer by passing
the left pointer and the
item.

if ptr → info == item

check if ptr → left == ptr → right

free (ptr)

return.

```
Else if
 ptr -> left == null
  Set
   p1. ptr -> right
    free(ptr)
     return p1
Else if
  ptr -> right == null
   Set
    p1. ptr -> left
      free(ptr)
       return p1

  Else
    Set p1 = ptr -> right
        p2 = ptr -> right

While
  {
   p1 -> let != null
    Set
       p1 -> left . ptr -> left
        free(ptr)
         return p2.

   return ptr.
```

Step 7 - Search Operation.

Read the element to be searched.

while (ptr)

if item > ptr → info

ptr = ptr → right

else if

item < ptr → info

ptr = ptr → left

else break.

// Check if ptr then point that the element is found

else point "Element not found"

return root.

Step 8 : Traversal Operation.

if root != null

root → left.

Point root → Info.

// call the traversal function recursively by passing root → right.

Program No-6

Aim : Set Operations

Step 1 : Start

Step 2 : union operation,

Read the cadinality of 2 sets.

if m != n.

print " cannot perform union "

else

Read the element in both sets.

Step 3 : Repeat ~~4&5~~ 4 and 5 until i < m

Step 4 : C[i] = A[i] | B[i]

print C[i]

i++

Step 5 C[i] = A[i] | B[i] // cardinality of 2 sets. m != n.

print C[i]

i++

Step 6 : difference operation.

Read the cardinality of 2 sets.

m != n.

print " cannot perform set difference operations.

else
read the relements in both sets

Repeat 7 ~~and~~ 8. untill i<n.

Step 7 - if A[i]=0 then C[i]=0.

else if B[i]==1 then C[i]=0

else C[i]=1

increment i++

Step.8 : Repeat step 9. untill i<m

print C[i]

i++.

Program No - 7

: AIM : Disjoint sets.

Step 1 : Start

Step 2 : Declare the structure and related structure variable

Step 3 : Declare a function makeset()

Step 3.1 : Repeat step 3.2 to 3.4 until i<n

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

Step 3.3 : set dis.rank [i] is equal to 0

Step 3.4 : Increment i by 1

Step 4 : Declare a function display set

Step 4.1 : Repeat step 4.2 and 4.3 until i<n

Step 4.2 : point dis.parent [i]

Step 4.3 - Increment i by 1

Step 4.4 - Repeat step 4.5 and 4.6 until i<n

Step 4.5 - point dis.rank [i]

Step 4.6 - increment i by 1

Step 5 - Declare a function find and par x to be the function.

Step 5.1 - Check if dis.parent [x] != x

then set the return value to

dis.parent[x]

Step 5.2 ÷ return dis.parent [x]

Step 6 - Declare a function union and par two variables

x and y

Step 6.1 - Set x set to find [x]

Step 6.2 - Set y set to find (y)

Step 6.3 - check if x set = = yset then return.

Step 6.4 - check if dis.rank [x set] < dis. rank [yset]

then.

Step 6.5 - set yset = dis. parent [y set]

Step 6.6 - Set -1 to dis.rank [x set]

Step 6.7 - Else if check dis. rank [x set] > dis.

rank [y set]

Step 6.8 - Set x set to dis. parent [y set]

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

Step 6.10 - Else disparent (y set] = x set.

Step 6.11 - set dis.rank [x set] to 1 to dis.rank

[x set]

Step 6.12 - set -1 to dis.rank (y set)

Step 7 - Read the number of elements

Step 8 - call the function make set

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

Step 10 - if the user choice to perform union operation. read the element to perform union. then all the function to perform union operation.

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

Step 11.1 - check if find (x) == find (y) then print Connected Component.

Step 11.2 - Else print Not Connected Component

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

Step 13 - End.