OUTPUT

Enter size of array 1 : 2

Enter size of array 2 : 2

Enter elements of array 1 in sorted order : 3

4

Enter elements of array 2 in sorted order : 1

2

Elements in C : 1 2 3 4

(1) Merge two sorted arrays and store in a third array.

Algorithm

1. Start
2. Read the size of two arrays into variables a and b
3. Read the elements in both arrays A and B in sorted order
4. Declare and set i = j = k = 0, array C[]
5. Repeat steps 6 through 8 while k < a + b
6. if i < a and j < b
   then check
   (i) if A[i] <= B[j]
   then set C[k] = A[i];
   set i = i + 1
   (ii) else
   set C[k] = B[j]
   set j = j + 1
   [end of inner if structure]
7. else if i == a
   then repeat steps while j < b
   set C[k] = B[j]
   set j = j + 1
   set k = k + 1
   [end of inner loop]
8. else
   Repeat steps while i < a
   set C[k] = A[i]
   set i = i + 1
   set k = k + 1
   [end of inner loop]
   [end of step 6 if structure]
   [end of step 5 loop]
9. Prints elements in the resultant array C
10. Stop

OUTPUT

```
Enter choice to perform:
1. Push        2. Pop
3. Search      4. Display      5. Exit

Choice : 2

UNDERFLOW...!
Enter choice to perform:
1. Push        2. Pop
3. Search      4. Display      5. Exit

Choice : 1

Enter data : 4

Enter choice to perform:
1. Push        2. Pop
3. Search      4. Display      5. Exit

Choice : 1

Enter data : 2

Enter choice to perform:
1. Push        2. Pop
3. Search      4. Display      5. Exit

Choice : 4

Stack from top to bottom:
2 -> 4 ->
Enter choice to perform:
1. Push        2. Pop
3. Search      4. Display      5. Exit

Choice : 2

2 Deleted...!
```

(2) Singly linked stack - Push, pop, linear search

Algorithm

1. Start
2. Create a structure 'node' with data members 'data' and 'next' where 'next' is a 'node' pointer
3. Declare a node pointer 'top' and initialize it with NULL
4. Read input 'choice' from users to perform stack operations
5. if choice == 1
   then read data 'd' from user
   call PUSH(d)
6. else if choice == 2
   then call POP()
7. else if choice == 3
   then read data 'd' from user
   call SEARCH(d)
8. else if choice == 4
   call DISPLAY()
   [end of if structure]
9. Stop

PUSH(d)

1. Create an instance of node 'newnode'
2. Set newnode -> data = d
3. Set newnode -> next = top
4. Set top = newnode
5. Return

POP()

1. if top == NULL
   print 'UNDERFLOW' and return
   [end of if structure]

2. Set top = top → next
3. Return

SEARCH (d)

1. Declare a variable c and set c = 1
2. if top == NULL
   print 'Empty stack'
   and return
   [end of if structure]
3. Declare a 'node' pointer temp and set temp = top
4. Repeat steps 5 to 7 while True
5. if temp → data == d
   then print 'item found at node :' c
   and return
   [end of if structure]
6. Set temp = temp → next
7. Set c = c + 1
   [end of step 4 loop]
8. Print 'item not found'
9. Return

DISPLAY()

1. if top == NULL
   print 'Stack is empty' and return
2. else
   set temp = top
   [end of if structure]
3. Repeat steps 4 and 5 while temp ≠ null
4. Print temp → data
5. Set temp = temp → next
   [end of step 5 loop]
6. Return

OUTPUT

Enter circular queue size : 3

Enter choice to perform :
1. Insert 2. Delete
3. Display 4. Search 5. Exit

Choice : 1

Enter data : 1

Enter choice to perform :
1. Insert 2. Delete
3. Display 4. Search 5. Exit

Choice : 1

Enter data : 2

Enter choice to perform :
1. Insert 2. Delete
3. Display 4. Search 5. Exit

Choice : 1

Enter data : 3

Enter choice to perform :
1. Insert 2. Delete
3. Display 4. Search 5. Exit

Choice : 4

Enter search item : 2

2 found at location 2

Enter choice to perform :
1. Insert 2. Delete
3. Display 4. Search 5. Exit

Choice : 2

1 deleted

(3) Circular Queue – Add, Delete, Search

Algorithm

1. Read queue size from user 'size'
2. Declare an array named 'cqueue[]' with size as 'size'.
   Declare and initialize variables 'front' and 'queue' with -1
3. Read input 'choice' from user to perform operations
4. if choice == 1
   then read data 'd'
   call enQueue(d)
5. else if choice == 2
   call deQueue()
6. else if choice == 3
   call display()
7. else if choice == 4
   then read data 'd'
   call search(d)
8. Stop

enQueue(d)

1. if front == -1
   then set front = rear = 0
   set cqueue[rear] = d
2. else if (rear == front - 1) or (front == 0 and rear == size - 1)
   then print 'Overflow' and return
3. else if rear == size - 1 and front ≠ 0
   then set rear = 0
   set cqueue[rear] = d
4. else
   set rear = rear + 1
   set cqueue[rear] = d
5. Return

deQueue()

1. if front == -1
   then print 'underflow' and return
2. else if rear == front
   then set rear = front = -1
3. else if front == size - 1
   then set front = 0
4. else
   set front = front + 1
5. Return

display()

1. if front == -1
   print 'Queue is empty' and return
2. Create variables i and set i = front
3. if i == rear
   print cqueue [i] and return
4. Repeat steps 5 to 8 while true
5. print cqueue [i]
6. if i == rear
   then return
7. if i == size - 1 and i ≠ rear
   set i = 0
8. else
   set i = i + 1
9. Return

Search(d)

1. Declare a variable i and set i = front
2. if front == -1
   print 'Queue is empty' and return
3. if i == rear
   then :

check if d == cqueue[i]
then print i+1 'location' and return
else print 'Search failed' and return

4. Repeat steps 5 to 8 while True
5. if data == cqueue[i]
then print i+1 'location'
and return
6. if i == rear
then print 'Search failed' and return
7. if i == size - 1 and i ≠ rear
then set i = 0
8. else set i = i + 1
9. Return

OUTPUT

1. Insert at beginning
2 Insert at end
3 Insert at position i
4 Delete at i
5 Display from beginning
6. Search for element
7. Exit

Enter choice : 1
Enter value to node : 1
Enter choice : 1
Enter value to node : 2
Enter choice : 1
Enter value to node : 3
Enter choice : 2
Enter choice to node : 7
Enter choice : 5
Linked list elements from beginning : 3 2 1 7

(4) Doubly linked list - Insertion, Deletion, Search

Step 1 : Start
Step 2 : Declare a structure and related variables
Step 3 : Declare functions to create a node, insert a node in the beginning, 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
Step 4.2 : Read the value to be inserted to the node
Step 4.3 : Set temp → n = data and increment count by 1
Step 5 : Read the choice from the user to perform different operation on the list.
Step 6 : If the user chooses 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 steps 4 to 4.3
Step 6.2 : Set head = temp and temp1 = 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~~ 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

create a newnode then set temp = head and then set head = temp 1

Step 7.2 : Else call the function to create a newnode then set temp1 → next = temp, temp → prev = temp 1 and temp1 = temp

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

Step 8.1 : Declare the necesary variable

Step 8.2 : Read the position where the node needs to be inserted, set temp2 = 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 = temp1

Step 8.6 : while i < pos then set temp2 = temp2 → next then increment i by 1

Step 8.7 : Call the function to create a new node and then set temp → prev = temp2, temp → next = temp2 → next → prev = temp, temp2 → next = temp

Step 9 : If the user chooses to perform deletion operation in the list then all the function to perform the deletion operation

Step 9.1 : Declare the necesary variables

Step 9.2 : Read the position where node needs to be deleted, set temp2 = head

Step 9.3 : check if pos < 1 or pos >= count + 1, then

print position out of range

Step 9.4 : Check if head == null then print the list is empty

Step 9.5 : While i < pos then set temp2 = temp2 → next and increment i by 1

Step 9.6 : Check if i == 1 then check if temp2 → next == null then print node deleted Free (temp2) set temp2 = head = null

Step 9.7 : Check if temp2 → next == null then temp2 → prev → next = ~~null~~ null then free (temp2) then print node deleted

Step 9.8 : temp2 → next → prev = temp2 → next then print node deleted then free temp2 and decrement count by 1

Step 9.9 : Check if i == 1 then head = temp2 → next then print node deleted then free temp2 and decrement count by 1.

Step 10 : If the user chooses to perform the display operation then call the function to display the list

Step 10.1 : Set temp 2 = n

Step 10.2 : Check if ~~temp2~~ temp2 ~~== n~~ == null then print list is empty

Step 10.3 : While temp2 → next 1 = null then print temp2 → n then temp2 = temp2 → next.

Step 11 : If the user chooses to perform the search operation then call the function to perform search operations

Step 11.1 : Declare the necessary variables

Step 11.2 : Set temp2 = head

Step 11.3 : Check if temp2 == null then print the list is empty

Step 11.4 : Read the value to be searched

Step 11.5 : While temp2 != null then check if temp2 -> n == data then print element found at position count + 1

Step 11.6 : Else set temp2 = temp2 -> next and increment count by 1

Step 11.7 : Print element not found in the list

Step 12 : End

OUTPUT

Input choice to perform:
1. Union 2. Intersection 3. Difference 4. Exit
Enter the cardinality of first set: 3
Enter the cardinality of second set: 3
Enter element to first set: (0/1) 1
0
1
Enter element of second set: (0/1) 0
1
0
Element of set 1 union set 2 : 1 1 1

(5.) Set data structure and set operations (Union, intersection and difference) using bit string

Step 1 : Start
Step 2 : Declare the necessary variables
Step 3 : Read choice from the user to perform set operation
Step 4 : If the user choose to perform union
Step 4.1 : Read the cardinality of 2 sets
Step 4.2 : Check if m1 = n then print cannot perform union
Step 4.3 : Else read the elements in both the sets
Step 4.4 : Repeat the step 4.5 to 4.7 until i < m
Step 4.5 : C [i] = A [i] | B [i]
Step 4.6 : print C [i]
Step 4.7 : Increment i by 1
Step 5 : Read the choice from the user to perform intersection

step 5.1 : Read the cardinality of 2 sets
step 5.2 : Check if m1 = n then print cannot perform intersection
step 5.3 : Else read the elements is both the sets
step 5.4 : Repeat the step 5.5 to 5.7 until i < m
step 5.5 : C[i] = A[i] & B [i]
step 5.6 : Print C[i]
Step 5.7 : Increment i by 1
Step 6 : If the user choose to perform set difference operation

Step 6.1 : Read the cardinality of 2 sets
Step 6.2 : Check if m1 = n then print cannot perform

set difference operations

step 6.3 : Else read the elements in both sets

Step 6.4 : Repeat the step 6.5 to 6.8 until $i<n$

Step 6.5 : Check if $A[i]==0$ then $C[i]=0$

step 6.6 : Else if $B[i]==1$ then $C[i]=0$

step 6.7 : Else $C[i]=1$

step 6.8 : Increment i by 1

step 7 : Repeat the step 7.1 and 7.2 until $i<m$

step 7.1 : Print $C[i]$

step 7.2 : Increment i by 1

OUTPUT

1. Insert in Binary tree
2. Delete from Binary tree
3. Inorder traversal of Binary tree
4. Search
5. Exit

Enter choice : 1

Enter new element : 50

Root is 50

Inorder traversal of binary tree is : 50

1. Insert in Binary tree
2. Delete from Binary tree
3. Inorder traversal of Binary tree
4. Search
5. Exit

Enter choice : 1

Enter new element : 25

Root is 50

Inorder traversal of binary tree is : 25 50

(6.) Binary Search Trees - Insertion, deletion, search

Step 1 : Start

Step 2 : Declare a structure and structure pointer 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 choice from the user to perform insertion, deletion, searching and inorder traversal.

Step 5 : If the user chooses to perform insertion operation then read the value which is to be inserted to the tree from the user.

Step 5.1 : Pass the value to the insert pointer and also the root pointer.

Step 5.2 : Check if !root then allocate memory for the root

Step 5.3 : Set the value to the info part of the root and then set left and right part of the root to null and return root

Step 5.4 : Check if root → info > x then call the insert pointer to insert to left of the root

Step 5.5 : Check if root → info < x then call the insert pointer to insert to the right of the root

Step 5.6 : Return the root

Step 6 : If the user choose to perform deletion operation then read the element to be deleted from the tree pass the root pointer and the item

to the delete pointer

Step 6.1 : Check if not ptr then print node not found

Step 6.2 : Else if ptr → info < x then call delete pointer by passing the right pointer and the item.

Step 6.3 : Else if ptr → info > x then call delete pointer by passing the left pointer and the item

Step 6.4 : Check if ptr → info == item then check if ptr → left == ptr → right then free ptr and return null

Step 6.5 : Else if ptr → left == null then set P1 ptr → right and free ptr, return P1

Step 6.6 : Else if ptr → right == null then set P1. ptr → left and free ptr, return P1

Step 6.7 : Else set P1 = ptr → right and P2 = ptr → right

Step 6.8 : While P1 → left not equal to null, set P1 → left ptr → left and free ptr, return P2

Step 6.9 : Return ptr

Step 7 : If the user choose to perform search operation then call the pointer to perform search operation.

Step 7.1 : Declare the necessary pointers and variables

Step 7.2 : Read the element to be searched

Step 7.3 : While ptr check if item > ptr → info then ptr = ptr → right

Step 7.4 : Else if item < ptr → info then ptr = ptr → left

Step 7.5 : Else break

Step 7.6 : Check if ptr then print that the element is found

Step 7.7 : Else print element not found in tree and return root

Step 8 : If the user choose to perform traversal then call the traversal function and pass the root pointers.

Step 8.1 : If root not equals to null recursively call the functions by passing root -> left

Step 8.2 : Print root -> info

Step 8.3 : Call the traversal function recursively by passing root -> right

OUTPUT

How many element ? 4

Menu

1. Union
2. Find
3. Display

Enter choice:

1

Enter element to perform union

3

4

Do you want to continue (1/0)

1

(7.) Disjoint sets and the associated operations (create, union, find)

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 : Print dis.parent[i]

Step 4.3 : Increment i by 1

Step 4.4 : Repeat steps 4.5 and 4.6 until $i<n$

Step 4.5 : Print dis.rank[i]

Step 4.6 : Increment i by 1

Step 5 : Declare a function find and pass x to the function.

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

Step 5.2 : Return dis.parent[n]

Step 6 : Declare a function union and pass two variables x and y

Step 6.1 : Set xset to find(x)

Step 6.2 : set yset to find(y)

Set 6.3 : Check if xset == yset then return

Set 6.4 : Check if dis.rank[xset] < dis.rank[yset] then -

Step 6.5 : Set yset = dis.parent [yset]

Step 6.6 : Set -1 to dis.rank [xset]

Step 6.7 : Else if check dis.rank [xset] > dis.rank [yset]

step 6.8 : Set xset to dis.parent [yset]

Step 6.9 : Set -1 to dis.rank [yset]

step 6.10 : Else dis.parent [yset] = xset

step 6.11 : Set dis.rank [xset]+1 to dis.rank [xset]

step 6.12 : Set -1 to dis.rank [yset]

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 chooses to perform union operation, read the element to perform union, then call the function to perform union operation

Step 11 : If the user chooses 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~~ chooses to perform display operation call the function display set.

step 13 : End