Merging.

Step1 : start

Step2 : Declare variables.

Step3 : Read the size of 1^{st} array.

Step4 : Read elements of 1^{st} array in sorted orde

Step5 : Read the size of 2^{nd} array

Step6 : Read the element of 2^{nd} array in sorted order.

Step7 : Repeat step 8 and 9 while i<m & j<n

Step8 : check if a[i] >= b[j] then c[k++] = b[j++

Step9 : Else c[k++] = a[j++]

Step10 : Repeat step 11 while i<m

Step11 : c[k++) = a[j++]

Step12 : Repeat step 13 while j<n.

Step13 : c[k++] = a[j++]

~~Step 13: Repeat step 13 while j<n~~

Step 14: Print the 1st array

Step 15: Print the 2nd array.

Step 16: Print the merged array.

Step 17: End

O/P

Size of firstarray.

2.

Enter value in sorted order.

3
6.

Size of second array.

enter value in sorted order.

4
5
7
8.

Array A:

3 6.

Array B:

4 5 7 8.

merged array:

3 4 5 6 7 8.

STACK OPERATIONS

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 passing the value to the function.

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

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 print insertion is successful.

Step6: if user choose to pop an element from the stack then call the function to pop the element

Step6.1: check if top==NULL then print stack is empty.

Step6.2: Else declare a pointer variable temp and initialize it to top

Step6.3:- Print the element that being deleted.

Step6.4: set temp=temp→next.

Step 6.5: ~~set~~ free the temp.

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

Step7.1: check if top==NULL then print stack is empty.

Step7.2: Else declare a pointer variable temp & initialize it to top

Step7.3: Repeat steps below while temp→next!=null

~~Step 7.3: Repeat steps below while~~

Step 7.4: print temp -> data

Step 7.5: set temp = temp -> next.

Step 8: If the user 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 element not found.

Step9: End.

O/P

Menu

1. push.

2. POP

3. display

4. search.

5. Exit.

Enter the choice : 1.

Enter the element to be inserted : 2.

Insertion is successful

.Menu.

1. push.

2. POP.

3. display.

4. Search.

5. Exit

Enter the choice : 1.

Enter the element to be inserted: 4.

~~[illegible]~~ insertion is successful

Menu.
1. push.
2. POP
3. display
4. search.
5. Exit

Enter the choice: 1

Enter the element to be inserted: 10.

Insertion is successfull.

Menu.
1. push.
2. POP.
3. display.
4. search.
5. Exit.

Enter the choice: 3.

4 → 2 → null.

Menu

1. push.

2. POP.

3. display

4. search

5. Exit.

Enter the choice: 4.

Enter the item which is to be searched: 2.

Item found at location: 2

Menu

1. push.

2. POP.

3. display

4. search.

5. Exit.

Enter the choice: 5.

Circular Queue Operation

Step 1: start.

Step 2: Declare the queue and other variables

Step 3: Declare the functions for enqueue, 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

Step6: if the user choice is the option.
dequeue then call the function dequeue

Step6.1: check if front==-1 and rear==-1 then
Print queue is underflow.

Step6.2: Else check if front == rear then
Print the element is to be deleted.
then set front = -1 and rear =-1

Step6.3: Else print the element to be
dequeued set front = front+1% max

Step7: if the user choice is to display.
the Queue then call the function.
display.

Step7.1: check if front=-1 and rear =-1
then ~~print~~ print Queue is empty.

Step7.2: Else repeat the step 7.3 while ~~i~~
i<= ~~rear~~

Step7.3: print queue[i] and set i=i+1 %.
max

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 it position and increment i by 1.

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

Step 9: End

Output

Menu

1 → Insert

2 → Delete.

3 → Display.

4 → Search.

5 → Exit

Enter the choice : 1

Enter the numbers to insert : 10

Menu

1 → Insert.

2 → Delete.

3 → Display

4 → Search.

5 → Exit.

Enter the choice : 1

Enter the number to insert : 20.

Menu.

1. Insert.

2. Delete.

3. Display

4. Search.

5. Exit

Enter the choice: 1

Enter the number to insert: 30.

menu.

1. Insert.

2. Delete.

3. Display.

4. Search.

5. Exit

Enter the choice: 3

10, 20, 30.

menu.

1 -> Insert.

2 -> Delete

3. Display.
4. search.
5. Exit
Enter the choice = 4.
Enter the element which is to be searched: 30
Item found at location 3.

Menu.
1. Insert.
2. Delete.
3. Display.
4. Search.
5. Exit.
Enter the choice: 2.
10 was deleted!

Menu.
1. Insert
2. Delete.
~~3. search~~
3. Display
4. search.
5. Exit
Enter the choice: 5.

~~st~~ Set operations

Step1 : start

Step2 : ~~Detou~~ Declere the neccessary variable

Step3 : Read the choice from the user to perform set operation

Step4 : If the user choose to perform union

Step4.1 : Read the cardinality ~~of~~ of 2 sets.

Step 4.2 : check if m != 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

Step4.5 : C[i] = A[i] | B[i]

Step 4.6 : print C[i]

Step4.7 : increment i by 1

Step 5: Read the choice from the user to Perform intersection.

Steps 5.1: Read the cardinality of 2 sets.

Steps 5.2: check if m!=n then print cannot perform intersection.

Step 5.3: Else read the elements in both sets.

Step 5.4: Repeat the step 5.5 to 5.7 untill i<n

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 m!=n then print cannot perform set difference operation

Step 6.3: Else read the element 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 6.6: 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

Press 1 for union.

Press 2 for intersection.

Press 2 for subtraction

Press 4 for exit

enter choice 1.

Enter the size of set1

3

Enter the element of set1

1
2
3

Enter the element of set 2

2
3

Union: 123.

Press 1 for union

~~Press~~
Press 2 for Intersection.

Press 3 for subtraction

Press 4 for exit.

Enter ur choice: 2

enter the size of set: 1

3

enter the element of set1

1
2
3

enter the size of set2

2

enter the element of set2

3
4

Intersection : 3

Press 1 for union

Press 2 for intersection

Press 3 for subtraction

Press 4 for exit

enter your choice 3

Enter the size of set1

3

enter the element of set1

1
2
3

Enter the size of set 2.

2

Enter the element of set 2.

3

2.

difference : 1

Press 1 for union

Press 2 for intersection.

Press 3 for subtraction.

Press 4 for exit.

enter choice = 4.

Binery search Tree

Step 1: Start

Step 2: Declare a structure and structure pointers for insertion delection and search operations and also declare a function for in order traversal

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

Step 4: Read the ~~declare~~ choice from the user to perform insertion, delection. Searching and inorder traversal.

Step 5: If the user choose 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.

Step5.2: check if ! root then allocate memory for the root.

Step5.3: set the value to the info part of the ~~root~~ root and then set left and right part of the root to null and return root.

Step5.4: 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.

Step5.6: Return the root.

Step 6: if the user choose to perform deletion operation then read the element to be deleted from the tree 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 = ~~R~~Ptr → right and
P2 = Ptr → right.

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

Step 6.9: Return Ptr.

Step 7: If the user choose to perform search operation the 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~~ item > Ptr → info then Ptr = Ptr → Right.

Step 7.4: ~~Else~~ 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

Step 9 : End.

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: 20.

root is 20.

Inorder traversal of binary tree is: 20

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

Inorder traversal of binary tree is: 20 25

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

Search operation in binary tree

enter the element to be searched: 5

element 25 which was searched is found

1. Insert is Binary tree
2. Delete from Binary tree
3. Inorder traversal of Binary tree
4. Search
5. exit

Enter choice : 5

End of program