Size of first array
2
Enter 2 value in sorted order
3
6
Size of Second array
4
Enter 4 value in sorted order
4
5
8
9
A list is:
3 6
B list is:
4 5 8 9
C list is:
3 4 5 6 8 9

Program No: 1

AIM: Program to perform merge operation in two sorted array

Step 1 Start
step 2 Declare the variables
step 3 Read the size of first array
step 4 Read elements of first array in sorted order.
step 5 Read the size of second array
step 6 Read the elements of second array in sorted order.
step 7 Repeate step 8 and 9 while i<m and j<n
step 8 Check if a[i]>=b[j] then c[k++]=b[j++]
step 9 Else c[k++]=a[i++]
step 10 Repeate step 11 while i<m
step 11 c[k++]=a[i++]
step 12 Repeate step 13 while j<n
step 13 c[k++]=b[j++]
step 14 Print the first array
step 15 Print the second array
step 16 Print the merged array
step 17 End

Stack using linked list

Menu

1. Push
2. Pop
3. Display
4. Search
5. Exit

Enter your choice: 1
Enter the value to be inserted: 2
Insertion is successful!!!

Menu

1. Push
2. Pop
3. Display
4. Search
5. Exit

Enter your choice: 1
Enter the value to be inserted: 34
Insertion is successful!!!

Menu

1. Push
2. Pop
3. Display
4. Search
5. Exit

Enter your choice: 1
Enter the value to be inserted 45
Insertion is Successful!!!

Program No: 2

AIM: Program to perform operations in singly linked stack.

Step1 Start
Step2 Declare the node and the required variables
step3 Declare the functions for push pop display and search
step4 Read the choice from the user to push, pop display or search an element
step5 If the user choose to push an element, then read the element to be pushed and call the function to push the element by passing the value to the function.
step5.1 Declare the newNode and allocate memory for the newNode
step5.2 Set newNode→data = value
step5.3 Check if top == null then set newNode→next = null
step5.4 Else set newNode→next = top
step5.5 Set top = newNode and then print insertion is succesful
step6 If the user choose to pop an element from the stack then call the function to ~~call~~ 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 is being deleted.
step6.4 Set temp = temp→next
step6.5 free the temp
step7 If the user choose to display the element in the stack then call the function to display the element in the stack.

Menu
1. Push
2. Pop
3. Display
4. Search
5. Exit
Enter your choise : 2
Deleted element : 45

Menu
1. Push
2. Pop
3. Display
4. Search
5. Exit
Enter your choise : 3
34 → 2 → Null

Menu
1. Push
2. Pop
3. Display
4. Search
5. Exit
Enter your choise : 4
Enter item which is to be searched : 2
item found at location 2.

Step 7.1 Check if top == Null then print Stack is empty
Step 7.2 Else declare a pointer variable temp and initialize it to top.
Step 7.3 Repeate step 7.4 and 7.5 while temp → next != null
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 is empty
Step 8.4 Else read the element to be searched from user
Step 8.5 Repeate the step 8.6 to 8.8 while ptr != null
Step 8.6 Check if ptr → data == item then print element found and its location 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 element not found
Step 9 End.

Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Press 4: Search the element
Enter your choice 1
Enter the element which is to be inserted 1

Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Press 4: Search the element
Enter your choice 1
Enter the element which is to be inserted 2

Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Press 4: Search the element
Enter your choice 1
Enter the element which is to be inserted 3

Press 1: Insert an element
Press 2: Delete an element
Press 3 Display the element
Press 4 Search the element
Enter your choice 1
Enter the element which is to be inserted 4

Program No: 3

AIM: Program to perform operations in Circular Queue.

Step 1 Start
Step 2 Declare the queue and required variables
Step 3 Declare the functions for enqueue, dequeue, display and search
step 4 Read the choice from the user to enqueue, dequeue display and search
step 5 If the user choose the option enqueue then read the element to be inserted from the user, then call the function enqueue and pass the value to the function.
step 5.1 Check if front == -1 and rear == -1 then set front = 0, rear = 0 and set queue[rear] = element
step 5.2 Else if rear+1 mod max == front or front == rear+1 then print Queue is overflow.
step 5.3 Else set rear = rear+1 mod max and set queue[rear] = element
step 6 If the user choose 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 mod max.
step 7 If the user choose the option to display the queue the call the function display

Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Press 4: Search the element
Enter your choice 2
The dequeued element is 1.

Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Press 4: Search the element
Enter your choice 3
Elements in a queue are: 2, 3, 4.

Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Press 4: Search the element
Enter your choice 4
Enter the element which is to be searched 4.
Item found at location 3.

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

Step 7.3 Print queue(i) and set i = i+1 incrementation.

step 8 If the user choose to search an element in the queue, then call the function to search an element in 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 c by 1

step 8.3 Check if c == 0 then print item not found

step 9 End

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

Program No: 4

AIM: Program to perform operation in doubly linked list

Step1 Start
Step2 Declare a structure and related structure variables
Step3 Declare functions to create a node, insert a node in the beginning, insertion at the end, insertion at the given position, display the list and search an element in the list.
Step4 Define a function to create a node, declare the required variables.
Step4.1 Set memory allocated to the node = temp. then set temp→prev = null and temp→next = null.
Step4.2 Read the value to be inserted to the node.
Step4.3 Set temp→n = data and increment count by 1.
Step5. Read the choice from the user to perform different operation on the list
Step6 If the user choose to perform insertion operation at the beginning then call the function to perform the insertion.
Step6.1 Check if head == null then call the function to create a node, perform step4 to step4.3.
Step6.2 Set head = temp and temp1 = head.
Step6.3 Else call the function to create a node, perform step4 to step4.3. Then set temp→next = head set head→prev = temp and head = temp.
Step7 If the user choose to perform insertion operation 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 new node. then set temp = head and then set head = temp1.

Step 7.2 Else. call the function to create a new node then set temp1 → next = temp, temp → prev = temp1 and temp1 = temp

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

step 8.1 Declare the necessary variables.

step 8.2 Read the position where the node node need 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 new node, 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.
temp2 → next → prev = temp.
temp2 → next = temp.

step 9 If the user choose to perform deletion operation in the list then call the function to perform the deletion operation.

step 9.1 Declare the necessary variables

step 9.2 Read the position where node need 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 temp2 = temp2→next and increment i by1
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 then free (temp2). then print node deleted.
Step 9.8 temp2→next→prev = temp2→prev. then check if i!=1 then temp2→prev→next = temp2→next
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 choose to perform the display operation then call the function to display the list.
Step 10.1 Set temp2 = h
step 10.2 Check if temp2 = null then print list is empty
step 10.3 While temp2→next != null then print temp2→n then temp2 = temp2→next.
Step 11. If the user choose to perform the search operation then call the function to perform search operation.
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 the check if temp2→n == data then print element found at position. count ++.
Step 11.6 Else set temp2 = temp2→next and increment count by 1
step 11.7 Print element not found in the list.

Output

Input choise 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 set1 union set2: 1 1 1

Program No: 5

AIM: Program to perform set operation.

Step1 Start
Step2 Declare the necessary variable
Step3 Read the choise from the user to perform set operation
Step4 If the user choose to perform union
Step4.1 Read the cardinality of two sets
Step4.2 Check if m1=n then print cannot perform union
Step4.3 Else read the elments in both the sets.
Step4.4 Repeate the step 4.5 to 4.7 until i<m
Step4.5 c[i] = A[i] | B[i]
Step4.6 print c[i]
Step4.7 Increment i by 1
Step5 Read the choise from the user to perform intersection.
Step5.1 Read the cardinality of two sets
Step5.2 Check if m1=n then print cannot perform intersection
Step5.3 Else read the elments in both the sets
Step5.4 Repeate the step 5.5 to 5.7 until i<m
Step5.5 c[i] = A[i] & B[i]
Step5.6 print c[i]
Step5.7 increment i by 1
Step6 If the user choose to perform set difference operation.
Step6.1 Read the cardinality of two sets
Step6.2 Check if m1=n then print cannot perform set difference operation
Step6.3 Else read the elments in both sets.
Step6.4 Repeate the step 6.5 to 6.8 until i<n.
Step6.5 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<n
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

Program No: 6.

AIM: Program to perform binary tree operations.

Step 1 Start

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

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

Step 4 Read the choice from the user to perform insertion deletion, 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.

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 the 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 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 > 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 pointer

Step 8.1 If root not equal to null recursively call the function 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.

Program No: 7

AIM: Program to perform operations on disjoined set

step1 Start
step2 Declare the structure and related structure variable.
step3 Declare a function makeSet()
step3.1 Repeat step 3.2 to 3.4 until i<n
step3.2 dis.parent[i] is set to i
step3.3 set dis.rank[i] is equal to 0.
step3.4 Increment i by 1
step4 Declare a function display set
step4.1 Repeat step 4.2 and 4.3 until i<n
step4.2 print dis.parent[i]
step4.3 Increment i by 1
step4.4 Repeat step 4.5 and 4.6 until i<n
step4.5 print dis.rank[i]
step4.6 Increment i by 1
step5 Declare a function find and pass x to the function
step5.1 Check if dis.parent[n] != n then set the return value to dis.parent[n].
step5.2 return dis.parent[n]
step6 Declare a function Union and pass two variables x and y
step6.1 set xset to find(x)
step6.2 set yset to find(y)
step6.3 Check if xset == yset then return.
step6.4 Check if dis.rank[xset] < dis.rank[yset] then
step6.5 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 makeset
step 9. Read the choice from user to perform union find and display operation.
step 10. If the user choose to perform union operation read the element to perform union. then call 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.