Program No: 1

Aim:- Write a program to merge two sorted arrays and store in a third array.

Step 1 : Start

Step 2 : Declare the variables

Step 3 : Read the size of first array

Step 4 : Read the elements of first array in sorted order

Step 5 : Read the second array's size

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

Step 7: Repeat 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 : Otherwise $c[k++] = a[i++]$

Step 10 : Repeat step 11 while $j<m$

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

Step 12 : Repeat step 13 while $j<n$

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

Step 14 : Print first array

Step 15 : Print the second array

Step 16 : Stop

Output :-

```
Size of first array
2
Enter 3 value in sorted order
1
5
7
Enter size of second array
3
Enter 3 value in sorted order
2
4
8
A list is :
        1 5 7
B list is :
        2 4 8
C list is :
        1 2 4 5 7 8
```

Program No: 2

Aim: Write a program to perform singly linked stack operations

Step 1: Start

Step 2: Declare the node and the required variables

Step 3: Declare functions for pop, push, display and search

Step 4: Read the choice from the user to pop, push, display or search the element.

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

Step 6(i) Declare the new Node and allocate memory for the new Node

(ii) set newNode -> data = value

(iii) Check if top == null then set newNode -> next == NULL

(iv) otherwise, set newNode -> next = top, top = newNode

(v) Print insertion is successful

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

(i) Check if top == NULL then print stack is empty

(ii) Otherwise declare a pointer variable temp and initialize it to top.

(iii) Print the element that is being deleted

(iv) Set temp = temp -> next

(v) free the temp

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

(i) check if top == NULL then print Stack is empty

(ii) otherwise declare a pointer variable temp and initialize it to top

(iii) Repeat step 7(iv) and 7(v) while temp -> next != NULL

(iv) Print temp -> data

(v) Set temp = temp -> next

Step 8: If the user choose to search an element from the stack then call the function to seach an element

(i) Declare the pointer variable ptr and other variables

(ii) initialize ptr = top

(iii) Check if ptr = NULL then print stack is empty.

(iv) otherwise read the element to be searched from user

(v) Repeat step 8(vi) ~~and~~ to 8(viii) while ptr != NULL

(vi) check if ptr -> data == item then print the element found at its location and set flag = 1

(vii) otherwise set flag = 0

(viii) Increment i by 1 and set ptr = ptr -> next

(ix) check if flag == 0 then print element not found

Step 9: Stop.

Output :-

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

Enter your choice : 1

Enter the element to be insert : 34

Insertion is success

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

Enter your choice : 4

Enter items to be searched : 34

Item found at location 1

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

Enter your choice : 2

The deleted element : 34

Program No: 3

Aim: Write a program to implement Circular Queue Operations

Step 1: Start

Step 2: Declare the queue and required variables

Step 3: Declare the function for enqueue, dequeue, display and search.

Step 4: Read the choice from the user to enqueue, dequeue, display or search an element

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

(i) Check if front == -1 and rear == -1 then set front = 0 rear = 0 and set ~~enqueue~~ queue[rear] = element

(ii) Otherwise rear + 1 mod max == front or front == rear + 1 then print queue is Overflow

(iii) else set rear = rear + 1 % max and set queue[rear] = element

Step 6: If the user choose the option dequeue then call the function dequeue

(i) Check if front == -1 and rear == -1 then print queue is underflow

(ii) otherwise check front == rear then print the element is to be deleted. Then set front = -1 and rear = -1

(iii) Print the element to be dequeued. set front = front + 1 % max.

Step 7: If the user choose the option to display the queue the call the function display.

(i) Check if front == -1 and rear == -1 then print Queue is empty.

(ii) Otherwise repeats the step (iii) while i <= rear

(iii) Print queue[i] and set i = i+1 % max

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

(i) Read the element to be searched in the queue.

(ii) Check if item == queue[i] then print item found and its position and increment c by 1.

(iii) Check if c == 0 then print item not found

Step 9: Stop

Output :-

1. Insertion
2. Deletion
3. Display
4. Search

Enter your choice : 1

Enter the element to be inserted : 45

1. Insertion
2. Deletion
3. Display
4. Search

Enter your choice : 1

Enter the element to be inserted : 88

1. Insertion
2. Deletion
3. Display
4. Search

Enter your choice : 2

The deleted element is 45

1. Insertion
2. Deletion
3. Display
4. Search

Enter your choice : 3

Elements in queue are 88

Program No: 4

Aim: Write a program to implement the operation on Doubly linked list.

Step 1: Start

Step 2: Declare a structure and related structure variables

Step 3: Declare functions to create a node insert a node at the beginning, insertion at the end insertion at the given position, display the list and search an element in the list

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

(i) Set memory allocated to the node = temp then set temp → prev = null and temp → next = null.

(ii) Read the value to be inserted to the node

(iii) 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 choose to perform insertion operation at the beginning then call the function to perform the insertion

Step 6 (i) check if head == null then call the function to create a node, perform step 4 to step 4(iii).

(ii) set head = temp and temp1 = head

(iii) otherwise call the function to create a node. Perform step 4 to step 4(iii). Then set temp → next = head. Set head → prev = temp and head = temp

Step 7: 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.

(i) Check if head == null then call the function to create a new node then set temp = head and then set head = temp1

(iii) Otherwise 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

(i): Declare necessary variables

(ii): Read the position where the node need to be inserted set temp2 = head.

(iii) Check if pos < 1 or pos >= count+1 then print the position is out of range.

(iv) Check if head == null and pos != 1 then print "Empty list cannot insert other than 1st position"

(v) Check if head == null and pos = 1 then call the function to create new node, then set temp = head and head = temp1

(vi) while i < pos then set temp2 = temp2 → next then increment i by 1

(vii) Call the function to create a new node and then set temp → prev = temp2, temp → next = temp2 → next 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(i): Declare the necessary variables

(ii): Read the position where node need to be deleted

Set temp2 = head.

(iii) Check if pos < 1 or pos >= count + 1 then print position out of range.

(iv) check if head == null then print the list is empty

(v) while i < pos then temp2 = temp2 -> next and increment i by 1.

(vi) check if i == 1 then check if temp2 -> next == null then print node deleted free (temp2) set temp2 = head = null

(vii) check if temp2 -> next == null then temp2 -> prev -> next = null then free (temp 2) then print node deleted.

(viii) temp2 -> next -> prev = temp2 -> prev then check if i != 1 then temp2 -> prev -> next = temp2 -> next.

(ix) 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.

(i) Set temp2 = h

(ii) Check if temp2 = null then print list is empty.

(iii) 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

(i) Declare the necessary variables

(ii) set temp 2 = head

(iii) Check if temp2 == null then print the list is empty

(iv) Read the value to be searched.

Step 11(v): while temp2 != = null the check if temp2 -> n = = data then print element found at position count + 1.

(vi) Otherwise set temp2 = temp2 -> next and increment count by 1

(vii) Print element not found in the list

Step 12: Stop.

Output :-

1. Insert at beginning
2. Insert at end
3. Insert at specific location
4. Delete at the specific location
5. Display from beginning
6. Search for element
7. Exit

Enter your choice : 1
Enter value to node : 2

Enter your choice : 1
Enter value to node : 3

Enter your choice : 2
Enter value to node : 4

Enter choice : 3
Enter position to be inserted : 2
Enter the value to node : 6

Enter your choice : 5
Linked list elements from beginning : 3 2 4
Enter choice : 6

Enter value to search : 4
Data found in 3 position
Enter choice : 6
Enter value to search : 6

Program No: 5

Aim: Write Program to perform set data structure and Set operations using Bit String

Step 1: Start

Step 2: Declare the necessary variable

Step 3: Read the choice from the user to perform set operation

Step 4: If the user choose to perform union

(i) Read the cardinality of two sets

(ii) check if m!=n then print cannot perform union

(iii) else read the elements in both the sets

(iv) Repeat the step 4(v) to 4(vii) until i<n

(v) C[i] = A[i] | B[i]

(vi) print C[i]

(vii) increment by 1

Step 5: Read the choice from the user to perform intersection

(i) Read the cardinality of two sets

(ii) check if m!=n then print cannot perform intersection

(iii) else read the elements in both the sets

(iv) Repeat the step 5(v) to 5(vii) until i<n

(v) C[i] = A[i] | B[i]

(vi) Print C[i]

(vii) increment i by 1

Step 6: If the user choose to perform set difference operation

(i) Read the cardinality of two sets

(ii) check if m!=n then print cannot perform set difference operation.

Step 6(iii): else read the elements in both sets

(iv): Repeat the step 6(v) do 6(viii) until i<n

(v): check if A[i]==0 then C[i]=0

Step (vi): else if B[i]==1 then C[i]=0

(vii): else C[i]=1

(viii): increment i by 1

Step 7: Repeat the step 7 (i) and 7 (ii) until i<m

(i) Print C[i]

(ii) increment i by 1

Step 8: Stop.

Output :-

1. Union
2. Intersection
3. Difference
4. Exit

Enter your choice: 1

Enter cardinality of first set: 3

Enter cardinality of second set: 3

Enter elements of first set (0/1): 1 0 1

Enter elements of second set : 1 0 0

Elements of set1 union set2 (0/1): 1 0 1

Program No: 6

Aim: Write a program to implement the operation on Binary Search Trees.

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 root.

(i) The value to the insert pointer and also the root pointer

(ii) check if !root then allocate memory for the root

(iii) 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.

iv) check if root → info > x then call the insert pointer to insert to left of the root.

(v) check if root → info < x then call the insert pointer to the right of the root.

(vi) Return the root.

Step 6: If the user choose to perform deletion operation then read the element to a deleted from the tree. Pas the root pointer and the item to the delete pointer.

(i) chuck if not ptr then print node not found.

Step 6 (ii): else if ptr → info < x the call delete pointer by passing the right pointer and the item

(iii) else if ptr → info > x then call delete pointer by passing the ~~right~~ left pointer and the item

(iv) check ptr → info == item then click if ptr → left == ptr right then free ptr and return null.

(v) else if ptr → left == null then set p1 = ptr → left and free ptr, return p1

(vi) else ~~if~~ if ptr → right == null then set p1 = ptr → ~~right~~ left and free ptr, return p1

(vii) else set p1 = ptr → right and p2 = ptr → right

(viii) while p1 → left not equal to null, set p1 → left = ptr → left and free ptr, return p2

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

(i) Declare the necessary pointer and variable

(ii) Read the element to be searched

(iii) while ptr check if item > ptr → info then ptr = ptr → right

(iv) else if item < ptr → info then ptr = ptr → left

(v) else break

(vi) check if ptr then print that the element is found

(vii) 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

(i) if root not equal to null recursive call the function by passing root → left.

(ii) Print root → info

(iii) Call the traversal function recursing by passing root → right.

Step 9: Stop

Output :-

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

Enter your choice: 1

Enter new element : 23

root is 23

Inorder traversal of binary tree is : 23

Program No: 7

Aim: Write a program to implement the operations on disjoint sets.

Step 1: Start

Step 2: Declare the structure and related structure variable

Step 3: Declare a function make set()

Step 3(i): Repeat step 3.(ii) to 3(v) until i<n

(ii): dis.parent[i] is set to i

(iii): set dis.rank[i]=0

(iv): increment i by 1

Step 4: Declare a function display set

(i): Repeat step 4(ii) and 4(iii) until i<n

(ii): Print dis.parent[i]

(iii): increment i by 1

(iv): Repeat step 4(v) and 4(vi) until i<n

(v): print dis.rank[i]

(vi): Increment i by 1

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

(i): check if dis.parent[n]!=x then set the return value to dis.parent[x]

(ii): return dis.parent[x]

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

(i): set x set to find(x)

(ii): set y set to find(y)

(iii): check if x set == y set then return.

(v): check if dis.rank < dis.rank[y set] then

Step 6(v): set yset = dis.parent [yset]

(vi): set -1 to dis.rank [xset]

(vii): ~~set~~ else if check dis.rank [xset] > dis.rank [yset]

(ix): set -1 to dis.rank [yset]

(x): else dis.parent [yset] = xset

(xi): set dis.rank [xset] +1 to dis.rank [xset]

(xii): set -1 to dis.rank [yset]

Step 7: Read the no: 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 choose to perform union operation read the element to perform union, then call the function to perform union operations.

Step 11: If the user choose to perform find operation read the element to perform union operation if connected

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

(ii): else print not connected component

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

Step 13: Stop.

Output :-

```
Enter the no: of elements : 7

        MENU
    * * * * * * * * * *

1. Union
2. Find
3. Display

Enter choice : 1

Enter elements to perform unio: 3

5

Do you wish to continue ?(1/0)

0
```