Pgmo 1: Aim: Merge two Sorted arrays and store in a third array

Step1: Start

step2: Declare the variables.

step3: Read the size of first array

step4: Read elements of first array in sorted order.

step5: Read the size of second array.

Step6: Read the elements of second array in sorted order.

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

step8. Check if a[i] >= b[j] then c[k++] = b[j++]

step9: ~~Check if a[i] >= b[j] then~~ Else c[k++] = a[i++]

step10. Repeat step 11 while i<m

step11: c[k++] = a[i++]

step12 Repeats step 13 while j<n

step13 c[k++] = b[j++]

step14: Print the first array

step15: Print the second array

step16: Print merged array

step17: Stop

Output

Enter size of array 1 : 4
Enter array 1 in sorted order : 2 6 7 8

Enter size of array 2 : 3
Enter array 2 in sorted order : 1 3 5

Merged array is 1 2 3 5 6 7 8

Pg: 2 Aim: Singly linked Stack - Push, Pop, linear search.

Step1: Start

Step2: Declare the node and the required variables.

Step3: Declare the function for push, pop, display & search.

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

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

Step 5.1 Declare the newNode and 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 Else set newNode → next = top

Step 5.5 Set top = newNode and then print insertion is successfull.

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

step 6.1. ~~If the~~ Check if top == Null then print stack is empty

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

step 6.3. Print the element that is being deleted.

step 6.4. Set temp = temp → next.

step 6.5. Free the temp.

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

Step 7.1. Check if top == NULL then print stack is empty.

step 7.2. Else declare a pointer variable temp and initialise 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 Repeat the step 8.6 to 8.8 while ptr != null.

step 8.6 Check if ptr → data == item then print element found and it's 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: stop.

Output

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

Enter your choice :2
Stack is Empty.

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

Enter your choice: 3
Stack is Empty.

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

Enter your choice : 1
Enter the element to be inserted: 34
Insertion is success.

1. Push
2. Pop

3. Display

4. Search

5. Exit

Enter your choice : 4

Enter item to be searched : 56

Item not found.

1. Push

2. Pop

3. Display

4. Search

5. Exit

Enter your choice: 5

Pgm 8: Aim: Circular Queue

step1: Start

step2: Declare the queue and required variables.

step3: Declare the functions for enqueue, dequeue, display and search.

step4: Read the choose from the user to enqueue, dequeue display and search.

step5: 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 point 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.

step 7.1 Check if front == -1 and rear == -1 then print queue is empty.

step 7.2. Else repeate the step 7.3 while i != rear.

step 7.3 print queue [i] and set i = i+1 mod max.

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. stop.

Output:

:: Menu ::

1. Enqueue
2. Dequeue
3. Display
4. Search
5. Exit

Enter any option: 1
Enter a number to insert: 56

:: Menu ::

1. Enqueue
2. Dequeue
3. Display
4. Search
5. Exit

Enter any option: 2
56 was deleted.

:: Menu ::

1. Enqueue
2. Dequeue.
3. Display

4. Search
5. Exit

Enter any option: 3
The circular queue is empty! nothing to display.

:: Menu::

1. Enqueue
2. Dequeue
3. Display
4. Search
5. Exit

Enter any option:
4
Enter the element to be searched 43
Item not found.

Prgm4: Doubly Linked List

step1: Start

step2: Declare a structure and related structure variables.

step3. Declare function to create a node. insert a node in the begining, 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.

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 choise 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.

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

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 newnode 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 ~~and~~ cannot insert other than 1" 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 ~~to~~ ~~to~~ 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 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 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 ~~set~~ 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 ~~insert~~ count by 1.

step 11.7 Print element not found in the list.

Step 12. St...

Output

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

Enter choice: 1
Enter value to node: 2

Enter choice: 2
Enter value to node: 5

Enter choice: 3
Enter position to be inserted: 4
Position out of range to insert.
Enter choice: 4
Enter position to be deleted: 2
Node deleted from list.
Enter choice: 5
Linked list elements from begining: 3
Enter choice 6.
Enter value to search: 3

Data found in 1 position

Enter choice : 7

[program finished]

Aim:

Prgm 5: Set Data Structure and Set Operations.

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

step 4.1 Read the cardinality of two 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 chose from the user to perform instruction.

step 5.1 Read the cardinality of two sets.

~~step 5.2 Else read the elements in both the sets.~~

step 5.2 Else Check if m!=n then print cannot perform instruction step 5.3 Else read the elements in 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 two sets.

step 6.2 Check if m! = n then print cannot perform set different operation.

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 & 7.2 until i<m

step 7.1 print C[i]

step 7.2 Increment i by 1

step 8. Stop

Output

1. Input choice to perform:

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

Choice : 1

Enter cardinality of first set: 4

Enter cardinality of second set: 4

Enter elements of first set: (0/1) 1001

Enter elements of second set: (0/1) 0110

Elements of set 1 union set 2 : 1111

Input choice to perform:

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

Choice: 2

Enter cardinality of first set: 3

Enter cardinality of second set: 3

Enter elements of first set: (0/1) 110

Enter elements of second set: (0/1) 100

Elements of set 1 intersection set 2: 100

Input choice to perform:

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

Choice: 3

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: (0/1) 1 1 1
Element of set - set 2 : 0 0 0

prgm6: Aim: Binary Search trees.

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 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 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 then 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.

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 choice : 1

Enter new element : 65

root is 65

Inorder traversal of binary tree is : 65

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

Enter choice : 1

Enter new elements : 55

Root is 65

Inorder traversal of binary tree is : 55 65

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 elements: 46

Root is 65.

Inorder traversal of binary tree is 46 55 65

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

Enter choice: 4

Search operation in binary tree
Enter the element to be searched: 4 6
Element 48 which was searched is found.

Pgm 7: Aim : Disjoint sets and Associated operations

Step1: Start

Step 2: Declare the structure and related structure variable

Step 3. Declare a function make set()

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 step 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 [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 pass 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 == y set then return
Step 6.4	Check if dis.rank [x set] < dis.rank [y set] 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 [yset]
Step 6.8	Set x set to disparent [y set]
Step 6.9	Set -1 to dis rank [y set]
Step 6.10	Else dis.parent [y set] = x set.
Step 6.11	Set dis.rank[x set]+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 makeset.
Step 9.	Read the choise 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 operation.

Step 11: If the user choose to perform final 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 operations 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 union: 3

5

Do you wish to continue? (1/0)

1.

MENU

******

1. Union
2. Find
3. Display

Enter choice: 2

Enter elements to check if connected components: 3

5

Connected components

Do you wish to continue? (1/0)

1.

MENU

****

1. Union
2. Find
3. Display

Enter choice: 3

Parent Array:

0 1 2 3 4 3 6

Rank array:

0 0 0 1 0 -1 0

Do you wish to continue? (1/0)

0

(Program finished)