MERGING

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 order.

Step 5: Read the size of second array.

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

Step 7: Repeat the step 8 and 9 while i<m j<n.

Step 8: Check if a[i] >= b[j] then c[k++] = b[j++]

Step 9: Else c[k++] = a[i++]

Step 10: Repeat the step 11 while i<m

Step 11: a[k++] = b[j++]

Step 14: Print the first Array.

Step 15: Print the second Array.

Step 16: Print the Merged Array.

Step 17: End.

Output

Size of first array:

2

Size of Second array

3

6

Enter 2 value in the sorted order

3

6

Size of second array

4

Enter 4 values in Sorted Order

4

5

8

9

A list is:

3 6

PAGE NO.
DATE / /

STACK OPERATION

Step 1: Start.

Step 2: Declare the node and required variables.

Step 3: Declare the function 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 and call the function to push the element by passing the value to the function.

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

Step 5.2: Set new -> data = Value

Step 5.3: Check if top == NULL then set newNode -> next = null.

Step 5.4: Set newNode->data = ~~value~~ top

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

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

Step 6.1: Check if top == NULL then print stack is empty.

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

Step 6.3: Print the element that being deleted.

Step 6.4: Set temp = temp -> next.

Step 6.5: Free the temp

Step 7: If the user choose the display then call the function to display 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 initialize it to top.

Step 7.3: Repeat the steps below 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 pointer 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.

Step 9: End.

Output:

1. Insert at beginning
2. Insert at end.
3. Insert at position
4. Delete at
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 beginning : 3 2 1 7

PAGE NO.
DATE / /

Circular - Enqueue 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 passing the value.

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 % max == front or front = rear + 1 the print Queue is overflown

Step 5.3 : Else set rear = rear + 1 % max and set queue [rear] = element.

Step 6 : If the user choice is 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 dequed set front = front + 1 % max.

Step 7 : If the user choice is to display the Queue then call the function display.

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 ~~found~~ queue[i] and set f=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 its position and increment i by 1

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

Step 9: End.

Output:

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
Enter your choice
Press 4: Search the element
Enter your choice 1
Enter the elements 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 elemnt

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 2
The dequed 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 queue are : 2, 3, 4

Press 1: Insert an element
Press 2 Delete an element
Press 3 Display the elemet

Enter your choice : 4

Enter the element which is to be searched 4 Item found at location 3.

Doubly linked list

Step 1: Start

Step 2: Declare a structure and add 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 into the node.

Step 4.3: Set temp -> n = data and increment (count by)

Step 5: Read the choice from 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 insertion.

Step 6.1: Check if head == null then call a function to create a node, perform step 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 temp1 -> next = head, set 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

Step 8.6: While i < pos then set temp2 = temp2 -> next and increment i by 1

Step 8.7: Call the function to create a new node and then set temp -> prev = temp2 and then set temp2 -> next = temp2 -> next -> prev = temp.

temp2 -> next = temp.

Step 9: If the user choose to perform deletion operation in the list then all 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 if 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 = next = 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 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 choose to perform the display operation then call the function to display the list.

Set 10.1 : Set temp2 = n.

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 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 print the list is empty

Step 11.5: While temp2 != null and 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

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

Enter choice: 1
Enter values 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: 4
Enter choice: 5
Linked list elements from beginning: 3 2 1 4

PAGE NO.
DATE / /

Set Operations

Step 1: Start

Step 2: Declare the necessary variables.

Step 3: Read the choice from the user to perform set operations.

Step 4: If the user choose to perform union.

Step 4.1: Read the cardinality 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

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 m|=n then print cannot perform intersection.

Step 5.3: Else read the elements in both the sets.

Step 5.4: Repeat step 5.5 to 5.7 until i<m.

Step 5.5: C[p] = A[i] AND B[j]

Step 5.6: Print C[p]

Step 5.7: Increment p 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 elements in both sets.

Step 6.4: Repeat the step 6.5 to 6.8 until i<m.

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

.. 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 elements to first set : (0/1)1
0
1
Enter elements of second set : 111

Binary Search Tree

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 required variable.

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 ~~pointer~~ 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~~ 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 7: 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 function by passing root→left.

Step 8.2: ~~print~~ print root→info.

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

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

TRIVENI
PAGE NO.
DATE / /

Disjoint

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 step 4.5 and 4.6 until p<n.

Step 4.5 : Print disjunkset[p]

Step 4.6 : Increment p by 1.

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

Step 5.1 : Check if disjunkset(x) != x then set the ~~return~~ return value to disjunkset[x].

Step 5.2 : Return disjunkset[x].

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

Step 6.1 : Set a 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 disjunkset[x set] < disjunkset[y set]

then

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

Step 6.6: Set rl to dis.rank[xset]

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

Step 6.8: Set x set to dis.parent[y set]

Step 6.9: Set rl to dis.rank[yset]

Step 6.10: Else dis.parent[yset] = x set.

Step 6.11: Set dis.parent[yset] = x set.

Step 6.12: Set rl to dis.rank[yset].

Step 7: Read the number of elements.

Step 8: Call the functions 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 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 elements 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

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 (Y/N)
1