Program: 01

MERGING

Step 1: Start
Step 2: Declare the variables
Step 3: Read the size of first array
Step 4: Read elements of first array in stored order.
Step 5: Read the size of second array
Step 6: Read the elements of second array in sorted order.
Step 7: Repeat step 8 and 9 while i<m & j<n
Step 8: check if a[i] >= b[j] then c[k++]
Step 9: Else c[k++]=a[i++] =b[j++]
Step 10: Repeat step 11 while i<m
Step 11: c[k++] = a[i++]
Step 12: Repeat 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.

Program: 02

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 new node & allocate memory for new node.
step 5.2 : Set new node → data = value ~~next = top~~
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.

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 stak is empty
step 6.2 : Else declare a pointer variable temp and initialize it 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 the elements in the stack.
step 7.1 : check if top == NULL then print stack is empty.
step 7.2 : Else declare a pointer variable temp & initialize it to top.
step 7.3 : Repeat 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 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.

print 9 : 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

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

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

Menu
1. Push
2. Pop
3. Display
4. Search
5. Exit
Enter your choice : 4

enter item which is to be searched: 2
item found at location 2

Program: 03

Circular Queue

Step 1: start
Step 2: Declare the que 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 enque.
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 que is overflow
step 5.3: Else set rear = rear + 1 % max and set
queue [rear] = element
step 6: if the user choice is 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 % 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 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 as the que
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.

Press1 : Insert an element
Press2 : Delete an element
Press3 : Display the element
Press4 : Search the element
Enter your choice : 1
Enter the element which is to be inserted :

Press1 : Insert an element
Press2 : 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
Press4 : 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

Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Press 4: Search the element
Enter your choice: 2
Entered Deleted 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

Program: 04

Doubly Linked List

Step 1: Start
Step 2: Declare structure and 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 to the node
Step 4.3: Set temp → n = data and increment count by 1
Step 5: Read the choice from the user to perform diffrent operation on the list.
Step 6: If the user choose to perform insertion operation at the begeinning then call the ~~operation~~ function to perform the insertion.
Step 6.1: Check if head == null then call the function to create a node, perform step 4 to 4.3

step 6.2 : set head = temp and temp 1 = head
step 6.3 : else call the function to create a node perform step 4 to 4.3 then set temp → 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 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 = temp 1.
step 7.2 : Else call the function to create a new node then set temp 1 → next = temp, temp → prev = temp 1 and temp 1 = temp.
step 8 : if the user choose to perform insertion in the list at any position then call the function to perform the insertion operation.
step 8.1 : Delare the necessary variable.
step 8.2 : Read the position where the node need to be inserted, set temp 2 = head.
step 8.3 : 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 = temp 1.
Step 8.6: While i < pos then set temp2 = temp 2
→ next then increment i by 1
Step 8.7: Call the functions to create a new
node and then set temp → prev = temp 2
temp → next - temp 2 → next → prev = temp
temp 2 → next = temp.
Step 9: If the user choose to perform deletion
operation in the list then all the functions
to perform the deletion operations.
Step 9.1: Declare the necessary variables.
Step 9.2: Read the position where node need
to be deleted set temp 2 = 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 temp 2 = temp 2 → 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 temp 2 = head = null

step 9.7: check if temp 2 → next == null then
temp2 → prev → next = null then free
(temp 2) then print node deleted.

step 9.8: temp 2 → next → prev = temp 2 → prev then
check if != 1 then temp 2 → prev → next
= temp 2 → next

step 9.9: check if i == 1 then head = temp 2 →
next then print node deleted then
free temp 2 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 temp 2 = n

step 10.2: check if temp 2 = null then print list
is empty

step 10.3: while temp 2 → next 1 = null then
print temp 2 → n then temp 2 = temp 2
→ next.

step 11: if the user choose to perform the
search operation the call the function.
to perform search operations.

step 11.1: Declare the necessary variables

step 11.2: set temp 2 = head

step 11.3: check if temp 2 == 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+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.

1. Insert at beginning
2. Insert at end
3. Insert at position
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 begening : 3 2 1 7

Program: 05

SET OPERATIONS

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
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 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 the step 5.5 to 5.7 until i<m
step 5.5: C[i] = A[i] & B[j]
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 m1=n then print cannot
perform set difference operation
step 6.3: Else read the element in both sides.
step 6.4: Repeat the step 6.5 to 6.8 until i<n
step 6.5: check if A[i]==0 then B[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

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 element to first set (0/1) : 1
0
1
Enter element to second set (0/1) : 0
1
0
Element of set 1 union set 2 : 111

Program: 06

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 the required variable

Step 4: Read the choice from the user to perform insertion, deletion, ~~sel~~ 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 the right of the root

step 5.6: Returns 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 going 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 returns 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 pointers.

Step 8.1: If root not equals to null
recursively call functions by passing
root → left

Step 8.2: print root → info

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

1. Insert in Buiary tree
2. Delete from Briary tree
2. Inorder traversed of buiary tree.
4. Search
5. Exit
Enter choice: 1
Enter neue 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 neue elemed: 25
root is 50
Inorder traversal of binary tree is : 25 50

Program : 07.

Disjoint set

Step 1 : start
step 2 : Declare the structure and related structure variable
step 3 : Declare a function makeset ()
step 3.1 : dis.parent[i] is set to i
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 : Repeate 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 returns
step 6.4 : check if dis.rank [x set] < dis rank [y set] then
step 6.5 : Set Y set = 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 [y set]
step 6.8 : set x set to dis.parent [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 choice from user to perform union, find and display
step 8 : Call the function make set
step 9 : Read the choice from user to perform union find and display operations.
step 10. If the user choose to perform unions, read the element to perform union then call the functions 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 functions display set.
step 13. End

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