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 sorted 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++] = b[j++]

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

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

STACK OPERATIONS

Step 1 : Start

Step 2 : Declare the node and the required variabl

Step 3 : Declare the functions for push, pop, display and search an element.

Step 4 : Read the ~~functions~~ & choice from the user

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

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

Step 8.3 - check if ptr = null the 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 point the element not

Step 9 : end.

CIRCULAR QUEUE 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 choise from the user

Step 5 : If the user choose the choise enqueue 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 queue is overflow

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

Step 6 : If the user choise is the option dequeue then call the function dequeue

Step 6.1 : Check if front ==1 and rear ==-1 then print Queue is overflow

Step 6.2 : else check if front == rear then print the element is to the be deleted then set front = -1 and rear = -1

Step 6:3 : else point the element to be deleted Set front = front + 1 % max

Step 7 : If the user choise is to display the queue then call the function display.

Step 7.1 : check if front = -1 and rear == -1 then put 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+i % 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 c by 1

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

Step 9 : End

Doubly Linked list operation

Step 1: Start

Step 2: Declare a structure and related variables

Step 3: Declare functions to create a nodes 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 insert count by 1

Step 5: Read the choice from the user to perform diff. 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 6.1: 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 temp1 = 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 the end.

Step 7.1: Check if head == null then call the function to create a newnode then set temp = head and then set head = temp1

Step 7.2: Else call the function to create a new node then set temp → next = temp, temp → prev = temp1 and temp1 = 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: Declare 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 pos < 1 or pos >= count + 1 then print the position is out

page: 10

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 newNode 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 = temp 2. 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 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 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 temp 2 → next
== null then print node deleted free (temp 2)
Set temp 2 = head = null

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

Step 9.8: temp 2 → next → prev = temp 2 → prev
then check if i != 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 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 temp 2 = n

Step 10.2: Check if temp 2 = null then print list
is empty

Step 10.3: while temp 2 → next != null then print
temp 2 → n then temp 2 = temp 2 → 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 +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

SET OPERATIONS

Step 1: Start

Step 2: Declare the necessary variables

Step 3: Read the ~~d~~ choise 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 : insrement 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 is 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 : insement i by 1

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

Step 6.2 : Check if $m != n$ then print Cannot perform set diff an operator

Step 6.3 : Else read the elemt in both sets.

Step 6.4: Repeat the Step 6.5 to 6.8 until

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 : insement 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 : insement i by 1

Binary Search Tree

Step 1 : Start

Step 2 : Declare a structre 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, 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: 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 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 left 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 free 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 then 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 if ptr → right == null then set p1 = ptr → left and free ptr, return p1

Step 6.8: 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

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 → 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 the functions by passing root → left.

Step 8.2: point root → info

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