Crystal DATE: PAGE: 1

Program No: 1

Aim: program to perform merge operation of two sorted array.

Step 1: Start
Step 2: Declare the variables
Step 3: Read size of first array
Step 4: Read elements of first array in sorted order
Step 5: Read size of second array
Step 6: Read the elements of second array in sorted order.
Step 7: Rpt 8 & 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[j++]
Step 12: Repeat step 13 while j<n
Step 13: c[k++] = b[j++]
Step 14: print the first array
Step 15: print the ~~Sorted~~ Second array
Step 16: print the Merged array.
Step 17: End.

Program No:-

Aim:

Program to perform operations in Singly linked list as stack

Steps:

1> Start

2> Declare node & required variables.

3> Declare the functions for push, pop, display & search an element.

4> Read the choice from the user.

5> If the user choose PUSH operation & call PUSH()

5.1> Declare the newnode & allocate memory for new node.

5.2> Set newnode -> data = value.

5.3> Check If top == NULL then

5.4> Set newnode -> next = ~~top~~ NULL

5.5> Newnode -> next = top

5.6> Set top = newnode & print "Insertion Success"

6> If user POP, then call POP()

6.1> If top == NULL, print "Stack is empty

6.2> Else, declare a pointer variable temp & initialize to top

6.3) print "element top is deleted"
6.4) temp = temp -> next
6.5) free(temp);
7) If user choose display call the the display()
7.1) check if top == NULL, print "stack is empty"
7.2) Else, declare a pointer temp & intialise to top;
7.3) Repeat steps below while temp -> next != NULL
7.4) print temp -> next
7.5) set temp = temp -> next
8) If user choose SEARCH, call the Search()
8.1) Declare a pointer variable ptr & other necessary variable.
8.2) Intialise ~~top~~ = ptr = top.
8.3) check ptr == null, then print "stack empty";
8.4) Else, read the element from the user.
8.5) Repeat steps 8.6 & 8.8 while (ptr != null)

8.6) check if ptr → data == item,
then print "element found"
set flag = 1

8.7) Else flag = 0

8.8) increment i. by 1 & set ptr = ptr → next

8.9) check if flag = 0 then :
print "Element Not found

9) END.

Program No: 3

Aim:

Program to perform operation in circular Queue.

Steps:

1) Start
2) Declare the queue & other variable.
3) Read the choice from the user.
4) If the user select ENQUEUE then, read element to be inserted. & call enqueue()
   4.1) check if front == -1 && rear == -1 then
   Set front = 0 & rear = 0 &
   Set queue[rear] = element.
   4.2) Else if rear + 1 % max == front or
   front == rear + 1
   then, print "Queue is overflow"
   4.3) Else,
   Set rear = rear + 1 % max & set que[rear] = element,
5) If user choose dequeue, then call dequeue()
   5.1) check if front == -1 & rear == -1
   print "Queue is underflow.
   5.2) Else check if front == rear then,

print "element is deleted"
set front = -1 & rear = -1
5.3> Else print dequeued.
set front = front + 1 % max
6> If the user choose is display
display()
6.1> If front = -1 & rear = -1 then,
print Queue is empty"
6.2> Else repeat the step 6.3 while i <= rear
6.3> print queue[i] & set i = i + 1 % max.
7> If the user choose the search then
call search()
7.1> Read element to search
7.2> If item == queue[i] then print.
"item found & its position"
8.3> check if c == 0 then,
print "NOT FOUND"
8> END.

Program no: 4

AIM:

Program to perform operations in doubly linked list

Steps

1) start
2) Declare a structure and related variables.
3) Declare functions to create node, insert a node in the beginning, at end, and on given position, display & search the list.
4) Define a function to create a node & the variables
   4.1) Set memory allocated to the node;
   node = temp then
   Set temp → prev = null
   temp → next = null
   4.2) Read the value to inserted to the node
   4.3) set temp → n = data & increment count by 1
5) Read the choice from user to perform different operation on the list.
6) If user choose to perform insertion at the beginning then call the function to perform inseetion
   6.1) Check if head == null, then create a node. St - 4 to 4.3

6.2> Set head = temp & temp1 = head.
6.3> Else, call function to creat a node perform step 4 to 4.3 then,
temp -> next = head;
head -> prev = temp
7> If the user choose Insertion at end calls that function.
7.1> check if head == NULL then, creat a new node then, set temp = head
set head = temp1
7.2> Else, create a new Node
set temp1 -> next = temp
temp -> prev = temp1 &
temp1 = temp
8> If user choose to perform Insertion list at any position then call the function
8.1> Declare the variables.
8.2> Read the position where the node head to be inserted, set temp2 = head
8.3> check if pos < 1 or pos >= count+1 then, "position Out of Range"
8.4> check if head == NULL & pos = 1 Then, print "Empty list"

8.5) check if head == NULL & pos = 1 then create newnode then,
Set temp = head & head = temp 1
8.6) While i < pos then,
Set temp 2 = temp 2 → next then,
increment i by 1
8.7) Call the function to create a new node
set temp → prev = temp 2
temp → next = temp 2 → next prev = temp
temp 2 → next = temp.
9) If user choose a deletion operation the list then perform the deletion opeiation
9.1) Declare nessicary variables.
9.2) Read the position where node need to be deleted set temp 2 = head.
9.3) If pos < 1 or pos > = count + 1 then print "position Out of range"
9.4) check if head == NULL, then print "list is Empty"
9.5) While i < pos then temp 2 = temp 2 → next & increment i by 1
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

9.7) check if temp2 → next == NULL then
temp2 → prev → next = NULL
free (temp2)
print "Node deleted"

9.8) temp2 → next → prev = temp2 → prev then check if i != 1 then,
temp2 → prev → next = temp2 → next

9.9) check if i == 1 then head = temp2 → next then print node deleted then free (temp2) & decrement count by 1.

10) If user choose to perform display operation then call display()

10.1) Set temp2 = h

10.2) check if temp2 = NULL then, print "list is empty"

10.3) while temp2 → next1 = NULL then print temp2 → n then temp2 = temp2 → next

11) If user choose to perform the search operation call search()

11.1) Declare the necessary variables.

11.2) Set temp2 = head.

11.3) check if temp2 == NULL then print the list is empty

11.4) Read the value to be searched

11.5> While temp2 != NULL then, check temp2 -> n == data then print "Element found at position count + 1

11.6> Else.
Set temp2 = temp2 -> next & increment count by 1.

11.7> print "Element Not found"

12> END.

Program: 5

AIM: Program to perform set operations.

Steps:

1) Start
2) Declare necessary variables
3) Read the choice from the user to perform set operation
4) If user choose Union call union()
   4.1) Read cardinality of two sets
   4.2) check if m != n then,
        print "cannot perform Union"
   4.3) Else,
        Read both sets.
   4.4) Repeat the step 4.5 to 4.7 until i<m
   4.5) C[i] = A[i] | B[i]
   4.6) print C[i]
   4.7) increment i by 1
5) Read the choice from the user to perform intersection
   5.1) Read cardinality of two sets.
   5.2) check if m != n print not able to perform intersection
   5.3) Else, read elements in both the sets.

5.4> Repeat 3.5 to 5.7 until i<m.
5.5> C[i] = A[i] | B[i]
5.6> print C[i]
5.7> increment i by 1
6> If the user choose to perform union.
6.1> Read cardinality of 2 sets.
6.2> check if m!=n then print "cannot perform difference operation"
6.3> Else. read element in both sets.
6.4> Repeat the 6.5 to 6.8 until i<m
6.5> check if A[i]==0 then,
C[i]=0.
6.6> Else if B[i]==1 then C[i]=0
6.7> Else C[i]=1
6.8> Increment by 1.
7> Repeat Step 7.1 & 7.2 until i<m
7.1> print C[i]
7.2> Increment i by 1.

Program : 6

Aim: Program to perform binary search tree operation.

Steps :-

1) Start
2) Declare a structure & pointer for insertion deletion & search. also declare a function to perform inorder traversal.
3) Declare a pointer as root & also the required variable.
4) Read the choice from the user
5) If user choose to perform insertion then call Insert ().
5.1) pass the value to Insert pointer & also the root pointer
5.2) Check If ! root then, allocate memory for root.
5.3) Set the value to the info part of the root & then set left & right part of the root to NULL return root
5.4) Check If root → info > x then call the insert pointer to insert to left of the root.
5.5) Check if root → info < x then call

pointer to insert to the right of the root.

5.6) Return the root.

6) If the user choose to perform deletion operation then read the element to be deleted from the tree pass the root pointer & item to the delete pointer.

6.1) check if not ptr then,
print "node not found"

6.2) Else if ptr → info < x then call delete pointer by passing the right pointer & the item

6.3) Else if ptr → info > x then call delete pointer by passing the left pointer and the item

6.4) check if ptr → info == item then,
if ptr → left == ptr → right then
free ptr & return null

6.5) Else if ptr → left == NULL then
set ptr → right & free ptr, return p1

6.6) Else if ptr → right == NULL then,
set p1 = ptr → left & free ptr. return p1

6.7) Else set p1 = ptr → right &
p2 = ptr → right

6.8> While p1 -> left node equal to null, set p1 -> left ptr -> left & free ptr, return p2

6.9> Return ptr.

7> If the user choose to perform Search Operation then call the pointer to perform search operation

7.1> Declare the necessary pointer & variables.

7.2> Read the element to be searched

7.3> While ptr check if item < ptr -> info then ptr = ptr -> right

7.4> Else if item < ptr -> info then, ptr = ptr -> left.

7.5> Else break.

7.6> Check if ptr then print that the element is found.

7.7> Else print element not found in tree & return root.

8> If the user choose to perform traversal then call the traversal & pass the root pointer

8.1> If root not equals to NULL, recursively call the function

by passing root → left

2.) print root → info.

3.) call the traversal function recursively by passing root → right.

Crystal DATE:
PAGE: 16

Program No: 7

AIM:

program to perform operations on disjoint set.

Steps:

1) Start
2) Declare the structure & related structure variable
3) Declare a function makeset()
3.1) Repeat 3.2 to 3.4 until i<n
3.2) dis.parent[i] is set to i
3.3) Set dis.rank[i] is equal to 0
3.4) increment i by 1
4) Declare a function display set.
4.1) Repeat Step 4.2 & 4.3 until i<n
4.2) print dis.parent[i]
4.3) increment by 1
4.4) Repeat Step 4.5 & 4.6 until i<n
4.5) print dis.rank[i]
4.6) increment i by 1
5) Declare a function find & pass x to the function
5.1) checks if dis.parent(x) != x then, set return value to dispacent(x)

5.2) return dis.parent(x)
6) Declare a function Union & pass two variables x & y
6.1) Set X Set to find (x)
6.2) Set Y Set to find (y)
6.3) Check if XSet == YSet then return
6.4) Check if dis.rank[XSet] < dis.rank[Y
then,
6.5) Set y set = dis.parent [ySet]
6.6) Set -1 to dis.rank [xSet]
6.7) Else if Check dis.rank[XSet] > dis.rank[ySet]
6.8) Set X Set to dis.parent [ySet]
6.9) Set -1 to dis.rank [ySet]
6.10) Else dis.parent[ySet] = XSet.
6.11) Set dis.rank[XSet]+1 to dis.rank[x
6.12) Set -1 to dis.rank[ySet]
7) Read the number of elements.
8) Call function make set
9) Read the choice from the user to perform Union, find & display operation
10) If user choose Union call Union()

11.4> if user choose Find operation
- call find()
11.1> check if find(x) == find(y)
then print "connected component.
11.2> Else print Not connected component
12> If the user choose to perform
display operation call function
display set.
13> End.