Merge Two Sorted Arrays and stored in a Third Array.

step 1: start

step 2: Declare the Variable [int a1[10], a2, a3]

step 3: Read the Size of the First array [scanf("%d", &n)]

step 4: Read elements of first array in Sorted Order.

[printf("\n enter the sorted elements of array 1: \n"); for(i=0; i<n; i++) scanf("%d", &a1[i]);]

step 5: Read the Size of the Second array. [n2]

step 6: Read the elements of ~~first~~ Second array in sorted Order

step 7: Repeat step 8 and 9 while i<n1 & i<n2.

step 8: check if a1[i] ~~>~~ < a[j] [while(i<n1 && j<n2) a[i++] = a[j++]

set merge[k] := a1 a1[i].

set k := k+1 and i := i+1.

else

set Merge[k] := a2[j]

set k := k+1 and j := j+1

[End of if structure]

[End of loop].

3. Repeat while I<M, then:
SET Merge[K] := array 1[I]
SET I := I+1 (I++)
SET K: K+1 (K++)
[End of loop].

[while (I<M)
a3[K] = a1[I];
I++;
K++;]

4. Repeat while J<N, then:
SET Merge[K] := array 2[J]
SET J: J+1 and K: = K+1
[End of loop]

[if while (J<N)
{
a3[K] = a2[J];
J++; K++;]

5 Exit:

Enter size of array 1: 2
Enter sorted elements of array 1:
1
3
enter size of array 2: 3
enter sorted element of array 2:
2
4
6
After Merging:
1
2
3
4
6

Singly linked stack.

Push, Pop, linear Search

---

step 1: start.

step 2: Declare the node and the required Variable.
(struct Node)

step 3: Declare the function for Push, Pop, Display, and search an element.

```
[Void push(int)
 Void pop();
 Void display();
 Void search ]
```

step 4: Request the element choice from the user (push)
~~Create a New Node with given data.~~

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

5.1 [If the user select push operation then create a New Node with given data (If top == Null then; Check whether the stack is empty).
Set Top == New Node Newnode -> Next = Null
Set New node -> Next = top top = Newnode.]

5.c : Set Top = newNode & then Push inside to Stack.

step 6. If user choose to pop an element from the Stack then call the function to pop the element.

[If top == Null then: check whether stack is empty, display "stack is empty"

else:
set Temp = Top
(Create a Temporary node and set it Top)
display Temp → data.
set Top = Temp → Next
(make Top point to the Next node)

Free (temp).
(delete the Temporary Node)].

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

7.1 : declare a pointer node ptr

7.2 set node ptr = top

9.3 · Repeat while temp ≠ NULL

If Temp → data == key then,

display "element found"

Set flag = 1 (if not)

9.4 Go to Step 10

else

Set Temp = Temp → Next

8.5 IF flag == 0 Then

display "element not found"

(= Goto)

[set flag = 0 increment one by one until

ple. pk → Next]

Circular Queue Add, Delete, Search

1 : Start

2 : Declare queue and required Variable

3 : declare the function for enqueue, dequeue display and Search [Void insert()
Void not delete()
Void display()
Void search()]

4 : Read the choice from the user to enqueue dequeue, display & Search.

[printf ("In enter any opt
scanf ("%d" & a);

5 : If the user choose the option enqueue then read the element to be inserted from the user. To call the function enqueue and pass the value to the functions

(ref Void insert)

5.1 : Check if front == -1 and rear == max-1 then set front = 0 rear = 0 and set queue [rear] = elem

5.2 : If front == 0 and rear == max-1 and front == rear+1 Print a it Overflow

[front==0 && rear == max-1 || front
== rear+1]
Print C'ln Circular Que is Full !!no";

5:3 : else rear = rear+1 and max and set
queue [rear] = element.

[rear = rear+1]
{ Queue [rear] = NO;

6s.4 : if the user choose the option dequeue the
call the function dequeue (or delete)

5.1 : check if front == -1 and rear == -1 Print queue
empty. {

6.α : else check if front == rear then Print the ele
to be deleted then set front = -1, rear = -1

[front == rear. { Print ("\n the
front = -1; deleted
rear = -1; }

6.3 : else Print the element to be dequeued, set
front = front + 1 Print max

[else front = front + 1;]

7 : If the user choose the option to display the Queue, then call Search display.

7.1 : Check if front == -1 and rear == -1 then Print Queue is empty.

[if front == -1
Print ("The C.Q. is empty! Not done"].

7.2 : else Repeat the Step 7.3 while it rem.

7.3 : Print Queue [i] and set i = (i+1) [while (i != rear) mod max.

8 : If the user choose to Search an element in the Queue then call the function to Search an element in queue.

8.1 : Read the element to be Searched in the Queue

8.2 : Check item == Queue[i] then Print item Found. and Position increment by 1

[if item == Queue[i]:
Print ("item found at location")]

8.3 : If (c == 0) Print Item Not Found

[If (c == 0)
Print ("Item Not Found")]

S2 : exit.

Set Operation

Aim: Program to perform Set Operation

step 1: Start

step 2: declare the necessary Variable [int ch, n, a[10], b[10], c[20], i, j, m, n;]

step 3: Read the choice from the user to perform set Operation [Print ("Input choice to perform set operation"); scanf("%d", &ch);]

step 4: If the user choose to perform union.

step 4a: Read the cardinality of [Print ("Input cardinality of set A"); scanf("%d", &m); Print ("Input cardinality of set B"); scanf("%d", &n);]

step 4b: check wether they Print Cannot Perform union [if (m!=n) Print ("Cannot perform union");]

step 4.3: else Read the element in both the Set

step 4.4: Repeat the step 4.3 to 4.7 until i<m

c[i] = a[i] / a[i]

step 4.6: Print c[i] Print("%d", c[i]);

4.7 : Increment i by 1

Step 5 : Read the choice from the user to Perform intersection. [case 2. Print()]

Step 5.1 : Read the Cardinality of 2 Sets.

[Print("In enter Cardinality of first Set
Scanf("%d", &m);
Print("In Enter Cardinality of Second Set")
Scanf("%d", &n)]

Step 5.2 : Check if m!=n then Print Cannot Perform intersection.

[If (m!=n)
{
Print("In 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[i]

Step 5.6 : Print C[i].

[Scanf("%d", &A[i]
" " &B[i]
C[i] = A[i] & B[i]
Print("%d", C[i]);

Step 5.7 : Increment i by 1.

step 6 : If the User Choice to perform Set diff. Operation.

step 6.1 : Read the Cardinality of 2 Sets

[Case 3.]

step 6.2 : check if m != n then Print Cannot Perform set diff. operation.

[If (m != n)
{
Print ("\n Cannot Perform Diff.!");]

step 6.3 : else read element 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] = i

step 6.8 : ID Incremented i by 1.

[If A[i] == 0)
C[i] = 0;
else.
{ If B[i] == 1)
C[i] = 0.

else
C[i] = i;
}
}
]

Step 3 : Repeat the step 2.1 and 2.2 until i <= n.

step 2.1 : Read c[i]

step 2.2 : IPC(num, 1, &y);

Aim: Doubly linked list.

Aim: Program to perform operate in doubly linked list

Step 1: Start.

Step 2: Declare the structure and related structure Variable.

Step 3: Declare functs to Create a node, insert a node in the beginning, insertion at the end, insertion at the given position, display the list and search an element in the list.

Step 4: Define a functs to Count a node

Step 5: declare the required Variable.

Step 5.1: Set memory allocate to the Create a node, declare the de Variable.

Step 4.2: Read

step 4.1 : Set memory allocated to the node = Temp, Then Set temp→n = data 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 next (cast by p.

Step 5 : Read the choice from the user to Perform diff. operate on the list.

step 6 : If the user choose to Perform insertion operate 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 temp 2 = 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 choose 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 new node. then set temp = head and then set temp = head->next.

Step 7.2 : else call the function to create a new node then set temp1->next = temp 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 need to be inserted. Set temp = head

Step 8.3 : check if pos < 1 or pos > count+1 then print the position is out of range

step 5.4 : check if head = Null and pos = 1 then Print "Empty list Cannot insert other than 1st Position"

step 5.5 : check head == Null and pos = 1 then call the function to create new node, then set temp = head and head = temp

step 5.6 : while i < pos then set temp2 = temp2 → next. Then increment i by 1

step 5.7 : Call the function to check a new node and then set temp → next = temp2. Temp → next = Temp2 → next. Temp2 → next → prev = temp. temp2 → next = temp. ... 4

step 6 : If the user choose Position delete operation in the list the call the function to perform the delete operation.

step 6.1 : Declare the necessary Variables.

step 6.2 : 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. Temp2 -> next and incremented in by 1.

Step 9.6 : Check if i == 1 then check if temp2 -> next == NULL temp Print node deleted . Free (temp2) set temp2. head = NULL.

Step 9.7 : Check if temp2 -> 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. temp2 -> next

Step 9.9 : Check if i == 1 the head = temp2 -> next then Print node deleted then free temp2 and decremented count by 1

step 10 : If the choose to perform the display operation then call the funct? to display the list

step 10.1 : Set temp