EXPERIMENT - 10

QUEUE USING LINKED LIST

AIM

Implementation of Queue using linked list.

Input :- Set of integers
Output :- Operation performed in Queue.

ALGORITHM

1. Start
2. Create a structure 'node' containing datatype int to store the integers and struct node *next.
3. struct node *head, *rear.
4. Choose from the options 1- Insert 2- Delete 3- Display 4 - Exit.
   4.1. If (choice = 1) then
   Insert()
   4.2. If (choice = 2) then
   delete()
   4.3. If (choice = 3) then
   display()
   4.4. If (choice = 4) then
   exit.
   4.5. else.
   Display "Invalid Choice"

5. Repeat Step 4 till option = 4.
6. Stop.

Insert()

1. struct node *ptr
2. Input value, item
3. ptr.data = item
4. if(head == NULL) then
   head = ptr
   rear = ptr
   head.next = NULL
   rear.next = NULL
5. else
   rear.next = ptr
   rear = ptr
   rear.next = NULL.

Delete()

1. Struct node *ptr.
2. If(head == NULL) then
   Display "Underflow"
3. else
   ptr = head
   head = head.next.
   free(ptr)

Display()

1. struct node *ptr
2. ptr = head
3. if (head == NULL) then
   Display "Empty Queue"

else

while (ptr != NULL)
   Display ptr.data
   ptr = ptr->next

RESULT

Program ran succesfully and output obtained.

EXPERIMENT - 11

POLYNOMIAL ADDITION USING A LINKED LIST

AIM

Implementation of Polynomial addition using Linked list.

Input :- 2 polynomials

Output: Sum of the two polynomials

ALGORITHM

1. Start
2. Declare a structure 'node' with datatype 'int' for the coefficient and exponent and struct node* next.
3. Struct node* p1 = NULL, *p2 = NULL, *SUM = NULL
4. int i = 1
5. Choose from the options 1 - Enter polynomial 1 2 - Enter polynomial 2 3 - Add Polynomial 4 - Exit.
   5.1. If (option = 1) then
      5.1.1. Input the coefficient.
      5.1.2. Input the Exponent.
      5.1.3. p1 = create (p1, co, exp)
   5.2. If (option = 2) then
      5.2.1. Input the coefficient.
      5.2.2. Input the Exponent.
      5.2.3. p2 = create (p2, co, exp)

5.3. If (option = 3) then
    5.3.1 sum = polyAdd (p1, p2, sum)
    5.3.2. Display Sum.
5.4. If (option = 4) then
    i = 0
6. Repeat step 5 till i=1
7. stop.

Struct node* Create (struct node* head, int co, interp)

1. Struct node *temp, *ptr
2. If (head == NULL) then
    temp.co = co
    temp.exp = exp
    temp.next = NULL
    head = temp.
3. else.
    temp = head
    while (temp.next != NULL)
        temp = temp.next
        ptr.co = co
        ptr.exp = exp
        ptr.next = NULL
        temp.next = ptr
4. Return head.

struct node * polyAdd (struct node *p1, struct node *p2,
struct node *sum)

1. struct node *poly1 = p1, *poly2 = p2, *res
2. If (poly1 != NULL and poly2 == NULL) then
   2.1. sum = poly1
   2.2. return sum
3. else if (poly1 == NULL and poly2 != NULL) then
   3.1. sum = poly2
   3.2. return sum.
4. while (poly1 != NULL and poly2 != NULL)
   4.1. If (sum == NULL) then
        res = sum
   4.2. else
        res = res.next.
   4.3. If (poly1.exp > poly2.exp2 then
        resp.co = poly1.co
        res.exp = poly1.exp
        poly1 = poly1.next.
   4.4. else if (poly1.exp < poly2.exp2 then
        res.co = poly2.co
        res.exp = poly2.exp
        poly2 = poly2.next.
   4.5. else if (poly1.exp == poly2.exp) then
        res.co = poly1.co + poly2.co
        res.exp = poly1.exp
        poly1 = poly1.next
        poly2 = poly2.next.

5. while (poly1 != NULL)
    res = res.next
    res.co = poly1.co
    res.exp = poly1.exp
    poly1 = poly1.next
6. while (poly2 != NULL)
    res = res.next
    res.co = poly2.co
    res.exp = poly2.exp
    poly2 = poly2.next
7. res.next = NULL
8. Return sum

Void display (struct node * head)
1. struct node * temp = head
2. While (temp != NULL)
    Display temp.co, temp.exp
    temp = temp.next.

RESULT

Program ran successfully and sum obtained

EXPERIMENT - 12
SPARSE ADDITION

AIM

Implementation of Addition of Sparse Matrix

Input :- 2 sparse matrix

Output :- Sum of 2 sparse matrix

ALGORITHM

1. Start
2. Input 2 matrix of size m x n
3. Convert sparse matrix to Triplet form
   Triplet();
4. Addition of 2 matrix.
   Add();
5. Display the sum, S
6. Stop

Triplet()

1. a[0][0] = m
2. a[0][1] = n
3. k = 1
4. for i = 0 to m-1
5. for j = 0 to n-1
   if (A[i][j] ≠ 0)
   a[k][0] = i
   a[k][1] = j
   a[k][2] = Aij

k = k+1

5. a[0][2] = k-1

6. Return a[][]

Add()

1. Input 2 matrix a, b in triplet form.
2. If a[0][0] ≠ b[0][0] || a[0][1] ≠ b[0][1] then
   Display "Addition not possible"
3. r = a[0][0]
4. c = a[0][1]
5. s[0][0] = r
6. s[0][1] = c
7. t_1 = a[0][2]
8. t_2 = b[0][2]
9. i = j = 1 ; k = 1
10. while (i <= t_1) && (j <= t_2)
    if (a[i][0] < b[j][0])
        s[k][0] = a[i][0]
        s[k][1] = a[i][1]
        s[k][2] = a[i][2]
        i = i+1 ; k = k+1
    else if (a[i][0] > b[j][0]) then
        s[k][0] = b[j][0]
        s[k][1] = b[j][1]
        s[k][2] = b[j][2]
        k = k+1 ; j = j+1

```
else if (a[i][1] < b[j][1]) then
    s[k][0] = a[i][0]
    s[k][1] = a[i][1]
    s[k][2] = a[i][2]
    i=i+1 , k=k+1;
else if (a[i][1] > b[j][1]) then
    s[k][0] = b[j][0]
    s[k][0] = b[j][1]
    s[k][0] = b[j][2]
    j=j+1; k=k+1;
else.{
    s[k][0] = a[i][0]
    s[k][1] = a[i][1]
    s[k][2] = a[i][2]
    k=k+1 ; i=i+1;
12. while (i<+2)
    s[k][0] = b[j][0]
    s[k][1] = b[j][1]
    s[k][2] = b[j][2]
    k=k+1; j=j+1;
13. s[0][2] = k-1
```

EXPERIMENT- 13

TRANSPOSE OF SPARSE MATRIX

AIM

Implementation of transpose of sparse matrix.

Input :- A sparse matrix

Output :- Transpose of matrix.

ALGORITHM

1. Start
2. Input the sparse matrix, A with size m×n
3. Convert to triplet form, a
   Triplet();
4. Transpose();
5. Display Transpose of sparse matrix.

Triplet()

1. a[0][0] = m
2. a[0][1] = n
3. k = 1
4. for i = 0 to m-1
   for j = 0 to n-1
   if (A[i][j] ≠ 0)
   a[k][0] = i
   a[k][1] = j
   a[k][2] = A[i][j]; k = k+1;

5. a[0][2] = k-1
6. Return a[][]

Transpose ()

1. T[0][0] = a[0][1]
2. T[0][1] = a[0][0]
3. T[0][2] = a[0][2]
4. t = a[0][2]
5. n = a[0][1]
6. k=1
7. for i=0 to n-1, i++
   for j=1 to t, ++t;
   if (a[j][1] == i)
   T[k][0] = a[j][1]
   T[k][1] = a[j][0]
   T[k][2] = a[j][2]
   k = k+1

RESULT

Program ran succesfully and transpose of matrix obtained.

EXPERIMENT-14

INFIX TO POSTFIX

AIM

Convert an infix expression to postfix and evaluate it.

Input:- Expression in infix form.
Output:- Postfix form and result.

ALGORITHM

1. Scan the I/p expression from left to right.
2. If the scanned character is an operand, then place it to o/p
3. else.
   3.1. if the precedance of the scanned operator is greater than the precedence of stack 'Top' operator, or if the stack is empty, then push the operator.
   3.2. else
      i) Repeatedly pop from stack & add to o/p, each operator which has the same or higher precedence than the scanned operator.
      ii) Push the scanned operator onto stack

4) If the scanned character is opening bracket, push it to stack.

5) If the scanned character is closing bracket, pop the contents from the stack & o/p it until "(" is found. Discard '(' & ')'

6) Repeat steps 2 to 5 until the expression is scanned completely.

7) If the stack is not empty, pop the contents from the stack & add it to o/p.

8) Display the o/p

9) To Evaluate the postfix expression push the operand.

10) pop the operand according to the operator present and find the result.

RESULT

Program ran succesfully and postfix expression and result obtained.

EXPERIMENT - 15
BINARY TREE TRAVERSAL

AIM

Write a program to implement traversal in a binary tree (recursive traversal)

Input :- A binary tree
Output :- post order, preorder and Inorder traversal results.

ALGORITHM

1. Start
2. Declare a structure 'node' having datatype 'int' for data and struct node *LC & *RC
3. Input data of different nodes of the tree from user.
4. Display the options, get input and call corresponding function.
5. Stop.

Create (ptr, item)

1. If (ptr ≠ NULL) then
   1.1. ptr.data = item
   1.2. Display "Node ptr has LST (Y/N) ?"
   1.3. If (option == 'Y') then
      1.3.1. Allocate memory and let lcptr point to it.
      1.3.2. ptr.LC = lcptr

1.3.3. Creat (lcptr, l.data)

1.4. else

ptr.LC = NULL.

1.5. Display "Node ptr has RST (Y/N)?"

1.6. If (option == 'Y') then

1.6.1. Allocate memory and let rcptr point to it

1.6.2. ptr.RC = rcptr

1.6.3. Creat (rcptr, r.data)

1.7. else

ptr.RC = NULL.

Inorder (root)

1. Inorder (root.LC)
2. visit root
3. Inorder (root.RC)

Preorder (root)

1. visit (root)
2. Preorder (root.LC)
3. preorder (root.RC)

Postorder (root)

1. Postorder (root.LC)
2. Postorder (root.RC)
3. visit root

Result

Program ran successfully and traversal through tree performed.

EXPERIMENT-16

BINARY TREE OPERATIONS

AIM

Perform Binary tree operations like search, insertion and deletion.

Input :- A binary tree

Output :- Operations on binary tree.

ALGORITHM

1. Start
2. Create a new node 'new'
3. Input the value of root node.
4. Create a binary tree.
5. Display Menu 1. Insert 2- Search 3- Delete
   4- Display 5- Exit.
6. do
   6.1. Display "Enter the choice"
   6.2. Read the choice
   6.3. switch (choice)
   case 1 : Insert()
   break;
   case 2 :- Search()
   break;
   case 3 :- Delete()
   break

case 4:- Display the tree.
break
case 5:- Exit(0)
break
default:- display "Enter a valid choice"
while (choice != 4)

7. Stop.

Search (ptr, key)
1. If ptr->data ≠ key
2. i) if ptr->LC ≠ NULL
Search (ptr->LC, key)
ii) else
return 0
iii) If ptr->RC ≠ NULL
Search (ptr->RC, key)
iv) else
return 0
3. else
Display element found at index ptr.

Insert (item)
1. If search (ptr, key)
i) Input choice whether to insert at LC or RC
ii) if choice = LC
if ptr->LC = NULL
ptr->LC = item

else
display "LC exists"
iii) else
if ptr.RC = NULL
ptr.RC = item
else
display "RL exists"
2. else
Display "parent not found."

Delete (Item)
1. ptr = root
2. parent = search (ptr., Item)
3. if parent ≠ NULL
i) ptr1 = parent.LC
ii) ptr2 = parent.RC
iii) if ptr.data = item
if (ptr1.LC = NULL & ptr1.RC = NULL)
parent.LC = NULL
else
display "Node is not leaf node"
else if (ptr2.LC = NULL & ptr.RC = NULL)
parent.RC = NULL
else
Display "Node not leaf node"

RESULT
Program ran successfully and operations done on binary tree.

EXPERIMENT - 17
HEAP SORT

AIM

Implementation of heap sort.

Input :- Array (binary tree) to be sorted
Output :- sorted tree (array representation)

ALGORITHM

1. Start
2. Input the array representation of a binary tree
3. Call Heap_sort (a[], n)
4. Stop.

Heap_sort (a[], n)

1. for (i = n/2 - 1) to 0
   Heapify (a, n, i)
2. for (i = n - 1 to 0)
   2.1. swap (a[0], a[i])
   2.2. heapify (a, i, 0)
3. Display the sorted tree
4. Stop

heapify (a[], n, i)

1. largest = i
2. le = 2*i + 1

3) rc = 2*i + 2
4) if (lc < n if a[lc] > a[largest])
largest = lc
5) if (rc < n & a[rc] > a[largest])
largest = rc
6) if (largest ≠ i)
6.1) swap (a[i] & a[largest])
6.2) heapify (a, n, largest)
7) return.

RESULT

Program ran successfully & sorted tree obtained.

EXPERIMENT-18
MERGE SORT

AIM
Implementation of Merge sort

Input:- Array representation of a tree
Output:- Sorted array.

ALGORITHM

Merge_sort (a[], beg, end)

1. if (beg < end)
   (i) mid = ⌊(beg + end)/2⌋
   (ii) Merge_sort (a, beg, mid)
   (iii) Merge_Sort (a, mid+1, end)
   iv) merge (a, beg, mid, end)

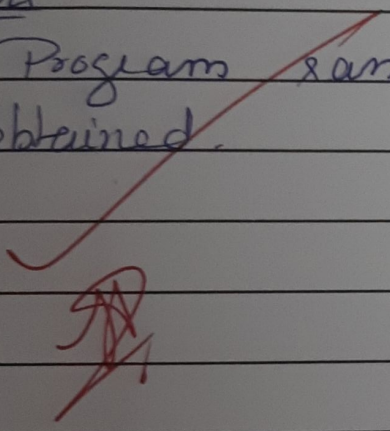
merge (a[], beg, mid, end)

1. i = beg, j = mid + 1, k = beg
2. while (i <= mid and j <= end)
   (i) if (a[i] <= a[j])
   b[k] = a[i]
   k++, i++
   (ii) else
   b[k] = a[j]
   k++, j++

3) while (i<=mid)
   (i) b[k] = a[i]
   (ii) k++, i++
4) while (j<=end)
   (i) b[k] = a[j]
   (ii) k++, j++
5) for (i = beg to end)
   (i) a[i] = b[i]
   (ii) i++

RESULT

Program ran succesfully and sorted array obtained.



EXPERIMENT-19
QUICK SORT

AIM

Implementation of quick sort

Input:- Array representation of tree

Output:- Sorted array.

ALGORITHM

Quick sort (a[], lb, ub)

1) if (lb<ub)
   (i) loc = partition (a, lb, ub)
   ii) Quick sort (a, lb, loc-1)
   iii) Quick sort (a, loc+1, ub)

Partition (a[], lb, ub)

1) pivot = a[lb], start = lb, end = ub
2) while (start < end)
   (i) while (a[start] <= pivot)
       start++
   (ii) while (a[end] > pivot)
       end--
   (iii) if (start < end)
       swap (a[start], a[end])
3) swap (a[lb], a[end])
4) return end