Experiment :

Practical - 1

Date

Page No. 1

```
/* Program For Simple Selection Sort */
/* SELSORT.C */

#include <stdio.h>
#include <conio.h>
# define maxn 200
 /* maximum size of array */

main ()
{
    int n, i, j, k, min, max ;
    int a[maxn], b[maxn];
/* a is unsorted list and b is sorted list*/
    printf("\n Enter the elements in the list");
    scanf("%d", &max);
/* read array elements */
   for (i=0; i<max; i++)
   {
   printf("\n enter %d th element:", (i+1));
   scanf("%d", &a[i]);
   }
/* start sorting the list */

 n=max;
 for(j=0; j<max; j++)
 {
```

CLASSTIME

```
min = 9999 ;
/* find smallest element */
for (i=0 ; i<m ; i++)
{
    if (a[i] < min)
    { min = a [i] ;
      k = i ;
    }
}
b [j] = min ;
/* remove the element entered in sorted list*/

for (i=k ; i<m-1 ; i++)
   a[i] = a [i+1] ;
for (i=m ; i<max ; i++)
   a[i] = 0 ;
 -m ;
}

/* print the sorted list */

for (i=0 ; i<max ; i++)
   printf ("\n %d \t %d", i, b [i]) ;
}
```

Experiment :

Practical - 2

Date

Page No. 3

```
/* Program for Bubble Sort */
/* BUBBLESORT.C */

# include <stdio.h>
# include <conio.h>
# define maxn 200
/* define maximum length of array */

main()
{
   int i,j, temp, max;
   int a[maxn]; /* the unsorted list */
   printf("\n enter number of elements in the list");
   scanf("%d", &max);

   /* read array elements */

   for (i=0; i<max; i++)
   {
     printf("\n enter %d the element:", (i+1));
     scanf("%d", &a[i]);
   }

   /* start sorting of list */

   for (j=0; j<max; j++)
```

CLASSTIME

```
{
  for (i=(max-1); i>j; i--)
  {
    if (a[i] < a[i-1])
    {
      temp = a[i];
      a[i] = a[i-1];
      a[i-1] = temp;
    }
  }
}

/* print sorted list */

for (i=0; i<max; i++)
    printf("\n %d \t %d", i, a[i]);
}
```

Experiment :

Practical - 3

Date

Page No. 5

```
/* Program for Quick Sort */

# include <stdio.h>
# include <conio.h>
# define max 200

int a[max], n, i, l, h;

/* l for low, h for high, n for total elements,
   i counter */

void inpu(void);

void main()
{
    inpu();
    getch();
}

void inpu(void)
{
   void out_p(int a[], int n);
   void qsort(int a[], int l, int h);
   printf("Enter total number of elements
    in the array:");
   scanf("%d", &n);
```

CLASSTIME

```
/* read the array elements */

for (i=0; i<=n; i++)
{
   printf("\n Enter %d th element :", i);
   scanf("%d", &a[i]);
}
l=0;
h = n-1;

/* get sorted array */

qsort(a, l, h);

/* print sorted array */

printf("\n The Sorted Array is :");
out-p(a, n);
}

/* function quicksort starts here */

void qsort(int a[], int l, int h)
{
   int temp, key, low, high;
   low = l;
   high=h;
```

```
key = a[(low + high) / 2];
do
{
  while (key > a[low])
  {
    low++;
  }
  while (key < a[high])
  {
    high--;
  }
  if (low <= high)
  {
    temp = a[low];
    a[low++] = a[high];
    a[high--] = temp;
  }
} while (low <= high);
if (l < high)
qsort (a, l, high);
if (low < h)
qsort (a, low, h); }
/* function output for display starts here */
void out_p (int a[], int n)
{ for (i=0; i <= n-1; i++)
  { printf ("\n %d", a[i]); }
}
```

Experiment : Practical -4 Date _____ Page No. 8

```
/* Program for Simple, Sort */
                    Merge
#include <stdio.h>
#include <conio.h>

int msort (int, int *, int, int*, int*);
int bsort (int, int *);
/* function main */

void main ()
{
   int a [100], b [100];
   int res [200];
   int i, k, n, m;

   printf ("\n Enter number of elements in
   first list : ");
   scanf ("%d", &n);

   for (i=0; i<n; i++)
   {
      printf ("\n Enter the %d th element:", i+1);
      scanf ("%d", &a[i]);
   }
/* Sort the elements of first list using
   bubble sort */
```

CLASSTIME

```
    bsort (n, a);
/* End of sorting */

    printf ("\n Enter number of elements in
      second list :");
    scanf ("%d", &m);
    for (i=0; i<m ; i++)
    {
      printf ("\n Enter %d the element:", i+1);
      scanf ("%d", &b[i]);
    }
/* Sort the elements of second list using
      bubble sort */
      bsort (m, b);

/* End of sorting */

/* perform the merging with sorting */
    k = msort (n, a, m, b, res);
    printf ("\n %d Duplicates found in lists",
          n+m-k);
    printf ("\n");

/* print the sorted list */

    printf ("\n The Sorted list is as follows: \n");
    for (i=0; i<k; i++){
```

```
    printf("%d", res[i]);
}
printf("\n");
}

/* function merge sort starts here */

int msort(int n, int l_a[], int m,
int l_b[], int s_list[])
{
    int i = 0;
    int j = 0;
    int k = 0;
    int ch, l;

/* merge elements recursively until both
        lists are exhausted */

while((i<n) && (j<m))
{
    if(l_a[i] < l_b[j])
    {
        s_list[k] = l_a[i];
        i++;
        k++;
    }
```

```
else
    if (l_a[i] > l_b[j])
    {
      r_list [k] = l_b [j];
      j++;
      k++;
    }
    else
    {
      r_list [k] = l_a [i];
      i++;
      j++;
      k++;
    }
  printf ("\n Next Pass is \t");
    for (ch=0; ch<k; ch++)
      printf ("%d \t", r_list [ch]);
}
  /* If size of a>size of b then copy rest
of the elements of a into r */
    if (i<n)
    {
      for (l=i; l<n; l++)
      {
        r_list [k] = l_a [i];
        i++;
        k++;
```

```
        printf ("\n");
        for (ch=0 ; ch<k; ch++)
            printf(" %d", r_list [ch]);
        }
    }
    else
        /* If size of b > size of a then copy
    rest elements of b into r */

    if (j<m)
    {
        for (l=j; l<m ; l++)
        {
            r_list [k] = l_b [j];
            j++;
            k++;
            printf ("\n");
            for(ch=0 ; ch<k ; ch++)
                printf ("%d", r_list [ch]);
        }
    }
    return (k);
}

/* function Bubble sort starts here */

int bsort (int n, int l[])
```

Experiment :

Date

Page No. 13

```
{
    int flag = 1;
    int i, j, k, temp;
    for (j=0; j<n-1; j++)
    {
        for (k=0; k<n-j-1; k++)
        {
            if (a[k] > a[k+1])
            {
                temp = a[k];
                a[k] = a[k+1];
                a[k+1] = temp;
                flag = 0;
            }
        }
        if (flag)
            break;
        else
            flag = 1;
    }
    printf("\n The list in Ascending order is:");
    for(i=0; i<n; i++)
        printf("%d ", a[i]);
    return 0;
}
```

Experiment :

Practical - 5

Date

Page No. 14

The Binary search procedure to search element is given list.

```
# include <stdio.h>
# include < conio.h>

void main()
{
    int a [100] ;
    int ead, mid, start, last ;
    int i, j, n, t, elem ;
    int flag = 0 ;
    printf("\n Enter number of elements in
       array") ;
    scanf ("%d", &n) ;
    for (i=0 ; i<=n-2 ; i++)
{
    for (j = i+1 ; j <= n-1 ; j++)
{
      if (a[i] > a[j])
      {
         t = a[i] ;
         a[i] = a[j] ;
         a[j] = t ;
      }
    }
}
```

Experiment :

Date

Page No. 15

```
printf("\n Enter the element to be searched");
scanf("%d", &elem);
mid = 0;
start = 0;
last = n-1;
/* start searching the element */
while ((start <= last) && (elem != a[mid]))
{
    mid = ((start + last) / 2);
    if (elem == a[mid])
    {
        printf("\n The search is successful");
        ead = mid;
        printf("\n The element is found at
%d the position", ead+1);
        flag = 1;
    }
    if (elem < a[mid])
        last = mid - 1;
    else
        start = mid + 1;
}
if (flag = 0)
{
    printf("\n Search unsuccessful. Element
not found in list."); }
    printf("\n");
}
```

CLASSTIME

Experiment :

Practical - 6

Date

Page No. 18

The linear search procedure to search element

The 'C' implementation of above algorithm is ;

```
/* Program Sequential Search */
# include <stdio.h>
main ()
{
  int mks [100] ; /* table of marks */
  int flg , x , i ;
/* Read data table in memory */
  for ( i=1 ; i<=100 ; i++)
  { printf (" \n Enter Marks : ") ;
    scanf ("%d", & mks [i] ) ;
  }
  printf (" \n Enter the marks to search ") ;
  scanf (" %d ", & x ) ;
  flg = 0 ;
  for ( i=1 ; i<= n ; i++)
{
  if (mks [i] == x ) {
  printf (" \n The element is found at %d
position ", i )
  flg = 1 ;
  break ;
  } }
if ( flg == 0 )
  printf (" \n The element is not present in table ") ;
}
```

CLASSTIME

Experiment :

Practical - 7

Date

Page No. 17

```
/* Program to add two matrices */

# include <stdio.h>
# include <conio.h>
# define max 10

void main()
{
  int i, j, m, n, r, p;
  int a[max][max], b[max][max],
   c[max][max];

  /* mXn is order of matrix a */
  /* rXp is order of matrix b */

  printf("\n Enter order of matrix A");
  scanf("%d %d", &m, &n);

  printf("\n Enter order of matrix B");
  scanf("%d %d", &r, &p);

  if(m != r || n != p)
  {
    printf("\n Order mismatch. Matrices
     cannot be added");
    return;
  }
```

CLASSTIME

Experiment :

Date

Page No. 18

```
/* read the matrix A */

for (i=0; i<m; i++)
{
   for (j=0; j<n; j++)
   {
      printf("\n Enter %d, %d th element of
        matrix A", i, j);
      scanf("%d", &a[i][j]);
   }
}
/* read the matrix B */
   for (i=0 ; i<m; i++)
   {
      for (j=0; j<n; j++)
      {
         printf("\n Enter %d, %d th element
            of matrix B", i, j);
         scanf("%d", &b[i][j]);
      }
   }
/* add matrix A and matrix B to give
      matrix C */
   for (i=0; i<m; i++)
   {
      for (j=0; j<n; j++)
      {
```

CLASSTIME

```
            c[i][j] = a[i][j] + b[i][j];
        }
    }
/* Print the resultant matrix c */
    printf("\n\n The sum of matrices A and
     B is as follows \n\n");
    for(i=0; i<m; i++)
    {
        printf("\n\t\t|\t");
        for(j=0; j<n; j++)
        {
            printf("%d \t", c[i][j]);
        }
        printf("|");
    }
    printf("\n\n\n");
    getch();
}
```

Experiment :

Practical - 8

Date

Page No. 20

Multiplication of two matrices using pointers.

```
#include<stdio.h>
#include<malloc.h>
#include<stdio.h>
main()
{
   int r1, r2, c1, c2;
int i, m, k;
printf("Enter the number of rows of matrix I=");
scanf("%d", &r1);
printf("Enter the number of columns of
   matrix - I = ");
scanf("%d", &c1);
printf("Enter the number of row of matrix - II
   =");
scanf("%d", &r2);
printf("Enter the columns of matrix - II");
scanf("%d", &c2);
if ( c1 != r2)
{
printf("Multiplication is not possible");
      exit(0);
}
   int *m1 = malloc(r1* c1* size of (int));
   int* m2 = malloc(r2*c2 * size of (int));
   int * m3 = malloc [r2 * c1 * size of int));
```

Experiment :

Date

Page No. 21

```
/* creates memory dynamically for first, second
and resultant matrices */
printf("Enter the element of matrix-I");
for (i=0; i<r1; i++)
{
for (j=0; j<<1; j++)
scan("%", m1 [i*<1+j]);
}
printf("Enter the elements of matrix-II");
for (i=0; i<r2; i++)
{
for (j=0; j<<2; j++)
scanf("%d", m2 [i*c2+j]);
}
for (i=0; i<r1; i++)
{
for(j=0; j<c2; j++)
}
m3 [i* c2+j] =0
for (k=0; k<c1; k++)
m3[ i*c2+j] + = m1 [i* c1*k]* m2[k* c2+j];
}
}
printf ("Multiplication of two matrices is \n");
for (i=0; i<r1; i++) {
printf ("\n");
for (j=0; j<c2; j++)
printf (m3 [i* c2+j]); }
}
```

CLASSTIME

Experiment :

Practical - 9

Date

Page No. 1

```
/* Implementation of Stack Using Linked List */
/* PROGRAM SL.C */

# include <stdio.h>
# include <malloc.h>
# include <ctype.h>
# include <string.h>

Struct node
{
   int info;
   struct node *next;
};
typedef struct node *stack;
stack start;

 void disp (struct node *);
 struct node * push (struct node *);
 struct node * pop (struct node *);
 int main menu();
/* main function */
   void main()
   {
     struct node * start;
     int cho;
     int f=0;
```

```
start = NULL;
do
{
    printf ("\n \t \t MAIN-MENU ");
    printf ("\n 1. Push");
    printf ("\n 2. Pop ");
    printf ("\n 3. Exit to system");
    printf ("\n Enter your choice (1-3): ");
    scanf ("%d", &cho);
    switch (cho)
    {
    case 1:
        start = push (start);
        printf ("\n After push operation stack is
          as follows : \n");
        disp (start);
        break;
    case 2:
        start = pop (start);
        printf ("\n After pop operation stack
          is as follows : \n");
        disp (start);
        break;
    case 3: f = 1;
        printf ("\n\t\t END OF SESSION \n \n");
        break;
```

```
        default:
            printf("\n You have entered wrong
            choice. Enter 1-3 only");
            break;
        }
    } while (f!=1);
}
/* function disp starts here */
void disp (struct node *item)
{
    while (item != NULL)
    {
        printf ("%d", item -> info);
        item = item -> next;
    }
}
/* function push starts here */
struct node * push (struct node item)
{
    struct node *new_item
    printf ("\n Input the new value for next
        location of the stack :");
    new_item = (struct node*) malloc (size of
            (struct node));
    scanf ("%d", &new_item -> info);
    new_item -> next = item;
    item = new_item
```

```
    return (item);
}
/* function pop starts here */
struct node* pop(struct node * item)
{
    struct node * temp ;
    if (item == NULL)
    {
        printf ("\n Stack is empty");
    }
    else {
        temp = item -> next ;
        free (item);
        item = temp;
        if (item == NULL)
            printf("\n Stack is now empty");
    }
    return (item);
}

/* end of the program */
```

Experiment :

Practical - 10

Date

Page No. 5

/* Program for Implementing Circular Queue */

```
# include <stdio.h>
# include <conio.h>
# include <ctype.h>
# define max 20

void cqins (void);
void cqdisp(void);
int cqdel (void);

int cq [10];
int front=-1, rear=0;
int cho;
char ch;
void main ()
{
  do
  {
    printf("\n ----------");
    printf("\n          Main Menu ");
    printf("\n 1. Insert ");
    printf("\n 2. Delete");
    printf("\n 3. Display");
    printf("\n 4. Exit to system");
    printf("\n ---------- ");
```

CLASSTIME

```
printf ("\n\n\t Enter your choice (1-4) ");
scanf ("%d", &cho);
switch (cho)
{
  case 1 : cqins();
           printf ("\n The queue after insertion
             is : \n");
           cqdisp();
           break;
  case 2 : cqdel();
           printf ("\n The queue after deletion is :
             \n");
           cqdisp();
           break;
  case 3 : printf ("\n The final status of queue
             is : ");
           cqdisp();
           break;
  case 4 : return;
  }
  fflush (stdin);
}
while (cho) (cho != 4);
}
/* Function insert starts here */
void cqins (void)
{
```

```
int num;
if (front == (rear + 1) % max)
{
    printf ("Queue is full \n");
    return;
}
else
{
    printf ("Enter the element to be inserted \n");
    scanf ("%d", &num);
    if (front == -1)
        front = rear = 0;
    else
        rear = (rear + 1) % max;
        q[rear] = num;
    }
    return;
}
/* Function delete starts here */
int cqdel (void)
{
    int num;
    if (front == -1)
    {
        printf ("Queue is empty \n");
        return (0);
    }
```

```
    else
    {
      num = cq[front];
      printf("Deleted element is = %d \n", cq[front]);
        if (front == rear)
      front = rear = -1;
      else
        front = (front+1) % max;
    }
    return (num);
  }
/* Function display starts here */
  void cqdisp (void)
  {
    int i;
    if (front == -1)
    {
      printf("Queue is empty \n");
      return;
    }
    else
    {
      for (i=front; i<=rear, i++)
      {
        printf("\n %d", cq[i]);
      }
    }
```

```
if (front > rear)
{
  for (i=front ; i<max ; i++)
  {
    printf ("\n %d", cq[i]);
  }
  for (i=0; i<=rear ; i++)
  {
    printf ("\n %d ", cq[i] );
  }
}
printf ("\n");
}

/* ------- end of program------- */
```

Experiment :

Practical - 11

Date

Page No. 10

Write a program for insertion and deletion of elements in a linked list.

```
# include <stdio.h>
# include <stdlib.h>
# include <allock.h>
 struct node
    {
      int info;
      struct node* link;
    };
      typed of struct node n;
     n* get node ();
main()
}
   n* first ;
   int ch , data , pos , n ;
   initialize (first);
printf ("creating a singly linked list \n");
printf ("How many nodes you wanted to
    create \n");
     scanf ("%d", &n);
     create ( & sirst , n);
   while (1)
   {
    printf ("Main menu \n");
```

CLASSTIME

```
printf("\n\t 1. Insertion into a linked list");
printf("\n\t 2. Deletion from a linked list");
printf("\n\t 3. display");
printf("\n\t 4. exit");
printf("\n\t Enter your choice <1-4>:");
scanf("%d", *ch);
switch (ch)
 {
case 1:
   printf("Enter data of information =");
   scanf("%d", & data);
   printf("Enter the position where you want to
      in sett");
   scanf("%d", & pos);
   insert list (& first, data, pos);
   break;
case 2:
   printf("\n Enter item you want to delete");
   scanf("%d", & data);
   delete list (& first, data);
   break;
case 3:
   traverse (first);
   break;
case 4:
   exit (0);
 }
```

```
    }
}
initialize (n** first)
    {
        (*first) = NULL;
    }
create (n** f, int n)
    n* temp, * current)
        int i, data
for (i=1, i<=n, i++)
    {
    printf ("Enter data field of node %d", i);
    scanf ("%d", &data);
    temp = getnode ();
    temp -> info = data;
    temp -> link = NULL;
If (*f == NULL)
    *f = temp;
    else
      current -> link = temp;
      current = temp;
    }
}
traverse (n** first)
}
  printf ("\n Linked list is as...\n\n");
  while (first != NULL)
```

CLASSTIME

Experiment :

Date

Page No. 13

```
    }
    printf("%d" → ", first → info);
    first = first → link;
  }
  link ("End \n");
}
Insert (n** first, int data, int pos)
{
  n* current, *t;
    int i;
    If (*first == NULL) || (pos == 1))
  }
  t = get node ()
  t → info = data;
      t → link = (* first);
      (* first) = t;
  }
  else
    }
    current = (* first);
      i = 1
    while (i > pos - 1) && current → link != NULL))
      {
      current = current → link;
      i++
    }
    t = get node ();
```

CLASSTIME

Experiment :

Date
Page No.

```
        t -> info = data;
        t -> link = current -> link;
        current -> link = t;
    }
}
delete (int **first, int data)
{
    int current, *prev;
    if (*first == NULL)
    {
        printf("List is empty");
    else
        prev = NULL;
        current = (*first);
    while ((current != NULL) && (current -> info !=
                                 data));
    {
        prev = current;
        current = current -> link;
    if (current == NULL)
    {
    printf("item not found in the list");
        return;
    }
    else if (prev == NULL)
        (*first) = (*first) -> link
    else
```

CLASSMATE

```
        else
        prev -> link = current -> link
        free (current);
    }
}
n* get node ()
{
    n* p;
    p = (n*) malloc (size of (n));
    return (p);
```

Experiment :

Practical - 12

Date

Page No. 16

Insertion and deletion of elements in doubly linked list.

Insertion -

```
Void INSINPLACE (node * FIRST, int intem)
{
  node * ptr, * temp, * prev ;
  ptr = (node*) malloc (size of (node));
  ptr -> info = item;
  IF ((FIRST == (node *) NULL) //(item >= FIRST ->
        Infor))
  {
    ptr -> link = first;
    FIRST = ptr;
    return;
  }
  else
  {  temp = first;
  }
If (temp -> link == (node*) NULL
  {
    ptr -> link = temp -> link;
    temp -> link = ptr;
    return;
  }
  prev = temp
  While ((item <= temp -> link) &&
```

```
(temp -> link != NULL )) do
{
    prev = temp;
    temp = temp -> link;
}
If (item > temp -> info)
{
    ptr -> link = temp;
    prev -> link = ptr;
}
else
{
    ptr -> link = temp -> link;
    temp -> link = ptr;
}
    return;
}
```

Deletion -

```
delete_first (Node *f)
}
    Node *t;
    int item,
If (*f == NULL);
{
    Proof ("\n List empty");
    getch ();
```

Assignment:

Date
Page No. 12

```
        return;
    }
    else
    {
      t = (*f); /* any temporary variable t points to
                   pointer first f*/
      item = t -> data; /* data field of first node
                           is accessed */
      (*f) = t -> link; /* the first pointer
       variable is update to next value)*/
       free (t); /* free the memory occupied by t
                    temporary)*/
       return (item);
    }
}
```

Experiment :

Practical-13

Date

Page No. 19

Find factorial of a given number using recursion.

```
#include <stdio.h>
main()
{
  int n, num;
  printf("\n Enter the number");
  scanf("%d", &sum);
     n= fact (num);
  printf("The factorial of a given number
   %d = %d", num, n);
  }
  fact (int n1)
       {
       if (n1==0)
          return(1);
  else
     return (n1* fact (n1-1);
  }
```

Experiment :

Practical - 14

Date

Page No. 20

Inserting and deleting elements in an array.

```
#include <stdio.h>
#include <conio.h>
#define size 30
 main()
{
   int A [size];
int i, n, pos, num;
printf ("Enter number of element <30 --");
   scanf ("%d", &n);
printf ("Enter array element.");
   scanf ("%d", A [i]);
printf ("Enter new element and position");
   scanf ("%d %d", &num, &pas);
   Insert (A, &n, pos, num);
   printf ("Array after insertion of new element
        --");
for (i=0; i<n; i++)
   printf ("%d", A [0]);
printf ("Enter position of element you want to
     delete");
   scanf ("%od", &pos);
   delete (A, sn, pos); /* delete function */
printf ("Array after deletion");
for (i=0; i<m; i++)
printf ("%d", A [i]);
```

CLASSTIME

```
}
void insert (int A[], int *n, int pos, int num)
{
int j;
j = *n
if (pos >= j)
{
  a[j] = num
printf ("item is inserted at end");
}
else
{
  while (j >= pos)
  {
    A[i] = a(j-1);
    j--;
  }
  A[pos-1] = num;
}
(*n)++;
}
delete [int A[], int *n, int pos]
{
int x, j
if (pos < *n)
{
  x = A[pos-1)
```

Experiment :

Date

Page No. 22

```
for (j = pos - 1 ; i<*n , i++)
  a[j] = a[j+1];
printf ("deleted item is %d", x);
(*n)--
}
```

CLASSTIME

Experiment :

Practical - 15

Date

Page No. 23

```
/* Conversion from infix to postfix notation */

# include <stdio.h>
# include <string.h>
# include <ctype.h>
# include <stdlib.h>

main()
void intopost (void)
{
  int i, p, len, type, precedence;
  char next;

  /* i for infix, p for postfix */

  i = p = 0;
  len = strlen (infix);
  while (i < len)
  {
    if (!white_space (infix [i]))
    {
      type = get_type (infix [i]);
      switch (type)
      {
        /* push left paren onto stack */
      case LP:
```

CLASSTIME

```
        push (infix[i]);
        break;

/* pop stack until matching left paren */
    case RP:
        while ((next = pop()) != '(')
            postfix[p++] = next;
        break;
    /* RP stack until first opt of higher
       precedence and then stack this opt */
    case opt:
        precedence = get_prec (infix[i]);
    /* Anything on stack to pop */
    while (top > emp && precedence <= get_prec
        (stack[top]))
            postfix[p++] = pop();
    push (infix[i]);
    break;
    }
}
    i++; /* next symbol in infix expression */
}
/* pop any remaining opts */
    while (top > emp)
        postfix[p++] = pop();
    postfix[p] = '\0'; /* ensure a string */
}
```

CLASSTIME

Experiment :

Date

Page No. 25

```
/* function to find opt type */

int get_type (char symbol)
{
   switch (symbol)
   {
   case '(' :
      return(LP) ;
   case ')' :
      return (RP);
   case '+':
   case '-':
   case '%':
   case '*':
   case '/':
      return (opt);
   default :
      return (opd);
   }
}
```

// -------- FINISHED ----------//

CLASSTIME