DSA MidSem (2020) Solution and Evaluation Scheme
Section A

1. Answer the following questions.

a. What is ADT? Implement push() operation for stack ADT using array data structure.

Solution: Abstract Datatype is the logical representation of data combined with some well defined functions.

```
push(int val)
{
 if(top==max-1)
        printf("stack overflow");
 else
        stack[++top]=val;
}
```

b. What is ADT? Implement pop() operation for stack ADT using array data structure.

Solution: Abstract Datatype is the logical representation of data combined with some well defined functions.

```
int pop()
{
 if(top==-1)
        printf("stack underflow");
 else
        val = stack[top--];
}
```

c. What is ADT? Implement push() operation for stack ADT using linked list data structure.

Solution: Abstract Datatype is the logical representation of data combined with some well defined functions.

```
push(int val)
{
 struct node *temp=(struct node *)malloc(sizeof(struct node));
 temp->info=val;
 temp->next=top;
 top=temp;
}
```

d. What is ADT? Implement pop() operation for stack ADT using linked list data structure.

Solution: Abstract Datatype is the logical representation of data combined with some well defined functions.

```
int pop()
{
 struct node *temp;
 if(top==NULL)
        printf("Overflow");
 else
 {      temp=top;
        val=top->info;
```

```
        top=top-link;
        free(temp);
        return(val);
}}
```

2.

a. A matrix B[10][20] is stored in the memory with each element requiring 4 bytes of storage. If the address of B[2][1] is 2140, write the formula to calculate the address of B[5][4] and find the address of B[5][4] when the matrix is stored in Column Major Order.

Solution: Loc(B[5][4])=2140+4((5-2)+10(4-1)) and 2272

b. A matrix B[10][20] is stored in the memory with each element requiring 2 bytes of storage. If the base address at B[1][2] is 2140, write the formula to calculate the address of B[5][4] and find the address of B[5][4] when the matrix is stored in Row Major Order.

Solution: Loc(B[5][4])=2140+2(20(5-1)+(4-2)) and 2304

c. A matrix B[20][10] is stored in the memory with each element requiring 2 bytes of storage. If the base address at B[1][2] is 2140, write the formula to calculate the address of B[5][4] and find the address of B[5][4] when the matrix is stored in Column Major Order.

Solution: Loc(B[5][4])=2140+2((5-1)+20(4-2)) and 2228

d. A matrix B[20][10] is stored in the memory with each element requiring 4 bytes of storage. If the base address at B[2][1] is 2140, write the formula to calculate the address of B[5][4] and find the address of B[5][4] when the matrix is stored in Row Major Wise.

Solution: Loc(B[5][4])=2140+4(10(5-2)+(4-1)) and 2272

3.

a. Suppose one head pointer pointing to the head node of circular double linked list. Write a C function/pseudo-code to delete the last node without traversing the linked list.

Solution:
```
delete(struct node *head)
        {
          struct node *ptr=head->prev;
          ptr->prev->next=head;
          head->prev=ptr->prev;
          free(ptr);
        }
```

b. Suppose one head pointer pointing to the head node of circular single linked list. Write a C function/pseudo-code to delete the immediate previous node of the last node (last but one node).

Solution: delete(struct node *head)

```
{
  Struct node *ptr=head, *prev;
  while(ptr->next->next!=head)
  {
     prev=ptr;
     ptr=ptr->next;
  }
  if(ptr == head)
        head=head→next
  else
        prev->next=ptr->next;
  free(ptr);
}
```

c. Suppose one pointer pointer(*ptr*) is pointing to any intermediate node of circular double linked list. Write a C function/pseudo-code to delete the previous node of *ptr* without traversing and without taking any extra pointer.

Solution: delete(struct node *ptr)

```
{
  ptr->prev=ptr->prev->prev;
   free(ptr→prev→next);
   ptr->prev->next=ptr;
}
```

d. Suppose one pointer (ptr) is pointing to any intermediate node of circular double linked list. Write a C function/pseudo-code to delete the next node of *ptr* without traversing and without taking any extra pointer.

Solution: delete(struct node *ptr)

```
{
   ptr->next=ptr->next->next;
   free(ptr→next->prev);
   ptr->next->prev=ptr;
}
```

4.

a. Following is the infix expression. Let the infix to postfix conversion algorithm is applied on this. Write the stack[] and postfix[] content when number of elements in the stack will be maximum. Infix[]={A+(B*(D/E^F)-D)+P}.

Solution: Stack: (+(*(/^ and postfix[]=ABDE

or Stack:(+(*/^ postfix[]=ABDEF

b. Following is the infix expression. Let the infix to postfix conversion algorithm is applied on this. Write the corresponding stack[] and postfix[] content when number of elements in the stack will be maximum. Infix[]= {A^(B/A*(C+D)-E)+F}

Solution: Stack: (^(*(+ and Postfix[]= ABA/C

Or Stack: (^(*(+ and Postfix[]= ABA/CD

c. Following is the infix expression. Let the infix to prefix conversion algorithm is applied on this. Write the stack[] and prefix[] content when number of elements in the stack will be maximum. Infix[]={A+(B*(D/E^F)-D)+P}.

Solution: stack[]=)+)-)^ and postfix[]=PDF

Or stack[]=)+)-)^ and postfix[]=PDFE

d. Following is the infix expression. Let the infix to prefix conversion algorithm is applied on this. Write the corresponding stack[] and prefix[] content when number of elements in the stack will be maximum. Infix[]={ A^(B/A*(C+D)-E)+F}

Solution: Stack []=)+)-)+ and postfix[]= F E D

or Stack []=)+)-)+ and postfix[]= F E D C

Or Stack []=)+)-*/ and postfix[]= F E D C + A

5.

a. The time complexity of the following code is:

```
int i, m, l = 0;
for (i = m / 2; i <= m; i++) {
      for (k = 2; k <= n; k = k * 2) {
            l = l + m / 2;
      }
}
```

i. O(n^2)
ii. O(nlogm)
iii. O(mlogn)
iv. O(logn^2)

Ans: iii

b. The time complexity of the following code is:

```
int abc(int n)
{
   int I, j, count = 0;
   for (int j = 0; j < n; j++)
        xyz(j);
   return count;
}
xyz(int p)
{
   for (int i = p; i > 0; i--)
            count = count + 1;
```

}

i. O(n^2)
ii. O(nlogn)
iii. O(n)
iv. O(log^n)

Ans: i

c. The time complexity of the following code is:

```
void abc(int n, int a[])
    {
        int k = 0, l = 0;
        for(; k < n; ++k)
            while(l < n && a[k] < a[l])
                l++;
    }
```

i. O(n)
ii. O(n^2)
iii. O(nlogn)
iv. O(n(logn)^2)

Ans: i

d. Let the following function reverses the length n number num=$K_1K_2K_3..K_n$ [K_i=value at i^{th} position of the number]

```
int num, reverse;
//Take input for num
reverse = 0;
while (num > 0)
{
    rev = rev*10 + num%10;
    num = num/10;
}
```

i. num = $K_1K_2K_3..K_{n-i}$ and reverse = $K_mK_{m-1}..K_{m-i+1}$
ii. num = $K_1K_2K_3..K_{n-i}$ and reverse = $K_mK_{m-1}..K_{m-i-1}$
iii. num = $K_mK_{m-1}..K_{m-i-1}$ and reverse = $K_1K_2K_3..K_{n-i}$
iv. None

Ans: i

Section B

1. Let the individual school of the university is maintaining the B.Tech. student information using the linked list data structure with attributes as name (string), roll-number (int), semester(int), section(char), and CGPA (float). Write the structure to define the node of the linked list.

Suppose for the class/semester promotion, the university wants to apply the following changes based on student's CGPA. Let after every semester result the student database is updated as follows.

a. For the 1^{st} to 7^{th} semester let following changes have to make

For CGPA	Semester	Section
CGPA < 6	Delete the node from the master linked list and add to another linked list.	
6<=CGPA<7	Semester promotion	Section will change to 'A'

7<=CGPA<8	Semester promotion	Section will change to ‘B’
8<=CGPA<9	Semester promotion	Section will change to ‘C’
9<=CGPA<10	Semester promotion	Section will change to ‘D’

b. For 8th semester, delete the student information or delete the node from the master linked list.

Write a program to implement the above.

Evaluation Scheme – Full mark for the correct answer and partial marks to be awarded depending on the correctness of the steps.

Solution:

```
strcut student                                          [Mark-1]
{
  char name[10];
  int roll;
  int sem;
  char sec;
float CGPA;
}*start, *start_new;

update()                                                [Mark-4]
{
   struct student *ptr;
   while(ptr!=NULL)
  {
     if ((ptr->CGPA<6) && (ptr->sem!=8))
    {       delete(ptr);
               insert(ptr);
    }
    else if((6<=ptr->CGPA<7) && (ptr->sem!=8))
    {
       ptr->sem=ptr->sem+1;
       ptr->sec=’A’;
    }
    else if((7<=ptr->CGPA<8) && (ptr->sem!=8))
    {
       ptr->sem=ptr->sem+1;
       ptr->sec=’B’;
    }
     else if((8<=ptr->CGPA<9) && (ptr->sem!=8))
    {
       ptr->sem=ptr->sem+1;
       ptr->sec=’C’;
    }
    else if((9<=ptr->CGPA<=10) && (ptr->sem!=8))
    {
       ptr->sem=ptr->sem+1;
       ptr->sec=’D’;
```

```
    }
    else if(ptr->sem==8)
        delete(ptr);
}

 delete(struct student *ptr)                                    [Mark-2.5]
{
   if(ptr==start)
          start=start->next;
 else
 {
      struct student *prev, *q=start;
      while(q!=ptr)
      {
          prev=q;
          q=q->next;
      }
      prev->next=q->next;
    }
   free(ptr);
}

insert(struct student *ptr)                                     [Mark-2.5]
{
   struct student *q;
   ptr->next=NULL;
   if(start_new==NULL)
      start=ptr;
 else
 {
  for(q=start_new; q->next !=NULL; q=q->next)
  q->next=ptr;
 }
}
```

2. Suppose one stack is given, where the elements are stored in sorted order with their number of occurrences i.e each stack element contains two part: element and the number of time it is occurring . Using the basic stack push() and pop() operations, implement the following insert() and delete() functions.
   a. ***Insert()***:- insert one new element in the stack using the push() or/and pop() operation(s) such that if the element exist then it will just increase the number of occurrences or if element doesn't exist then it will insert the element with its occurrence as 1. After insertion of the element the stack must be sorted.
   b. ***Delete()***:- delete the existing element from the stack using the push() or/and pop() operation(s) such that if the element exist then it will just decrease the number of occurrences or if element doesn't exist then it will give the underflow message.

Example: Let the stack contains the following **elements along with its occurrences in sorted order.**

Element	Occurrences
15	2
13	1
9	5
5	3
4	1

Insert(4):-

Element	Occurrences
15	2
13	1
9	5
5	3
4	1+1=2

Insert(3):-

Element	Occurrences
15	2
13	1
9	5
5	3
4	2
3	1

Delete(5):-

Element	Occurrences
15	2
13	1
9	5
5	3-1=2
4	2
3	1

Delete(13):-

Element	Occurrences
15	2
9	5
5	2
4	2
3	1

[Note: As 13 is having a single occurance, it is completely deleted from the stack]

Evaluation Scheme – Full mark for the correct answer and partial marks to be awarded depending on the correctness of the steps.

Solution:

```
struct node
{
   int num;
   int count;
}stack[50];

top=-1;

void insert(int val)                                    [Mark-4]
{
   struct node temp;
   if((top == -1) || (stack[top].num < val)
         push( val, 1);
   else if(stack[top].num == val)
   {
     temp = pop();
     temp.count++;
     push(temp.num, temp.count);
```

```
    }
    else
    {
       temp = pop();
       insert(val);
       push(temp.num, temp.count);
    }
}
```

[Mark-4]

```
void delete(in val)
{
   sturct node temp;
   if(top == -1) && (stack[top].num < val)
        return;
   else if(stack[top].num == val)
   {    temp=pop();
        if(temp.count > 1)
        {    temp.count--;
             push(temp.num, temp.count);
        }
   }
   else
   {
      temp=pop()
      delete(val);
      push(temp.num, temp.count);
   }
}
```

[Mark-1]

```
void push(int n, int c)
{
     top++;
     stack[top].num = n;
     stack[top].count = c;
}
```

[Mark-1]

```
struct node * pop()
{
 struct node *temp;
 if (top == -1)
     temp=NULL;
  else
  {
  temp=(struct node *) mallocc(sizeof(struct node));
  temp.num=stack[top].num;
  temp.count=stack[top].count;
  top--;
  }
  return(temp);
}
```

3.

a. What is the requirement to convert any infix expression to it's corresponding prefix or postfix notation? Compare the postfix and prefix conversion process/logic. Using the conversion algorithm, stepwise translate the following infix expression to its corresponding postfix expression. And then do the evaluation of the postfix expression using the stack data structure.
Infix[]= (A-(B+C)/F)^C+G/E where A=10, B=5, C=3, F=2, G=16, E=4

b. Write an algorithm/ C-function to identify the position of an unmatched parenthesis in a given mathematical expression using stack. For Example: (x+y)*((z-w), Output: if the expression starts from 0 position, in 6th position the left parenthesis '(' is an unmatched parenthesis.

Evaluation Scheme – Full mark for the correct answer and partial marks to be awarded depending on the correctness of the steps.

Solution:

a. [1]The requirement to convert any infix expression to its corresponding prefix or postfix notation is to maintain an expression without parenthesis and to make free the expression from the rule of operator precedence. **[Mark-1]**

[2] **[Mark-1]**

Postfix Conversion	Prefix Conversion
Processing input array is done from left to right	Processing input array is done from right to left
When any new operator encountered in the input array, higher or equal precedence operators are popped to postfix[]	When any new operator encountered in the input array, higher precedence operators are popped to postfix[]
After the conversion process, it is not required to reverse the resultant array to get postfix[]	After the conversion process, it is required to reverse the resultant array to get prefix[]

[3] Infix [ ]=(A-(B+C)/F)^C+G/E) //One close parenthesis added to the end of expression and open parenthesis to the stack **[Mark-2]**

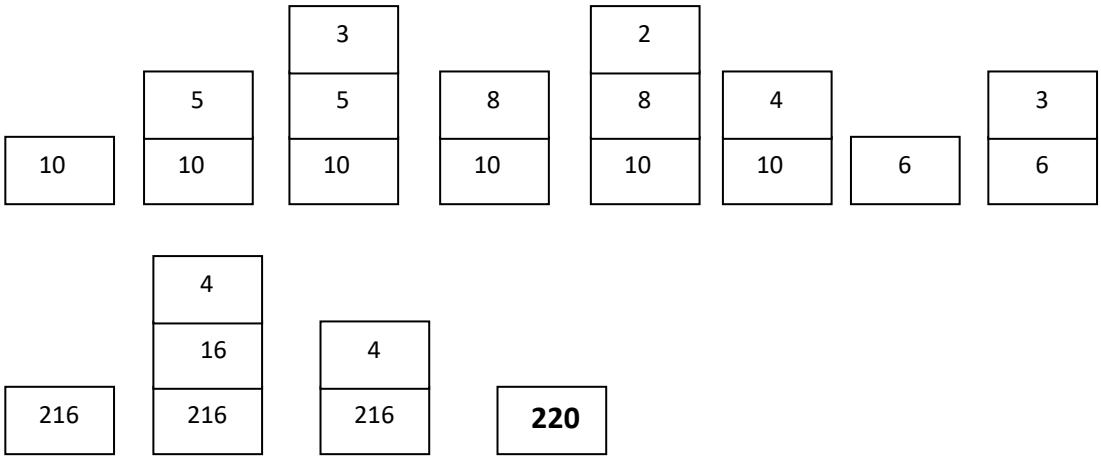
Conversion into postfix expression

Symbol Scanned	Stack	Postfix Expression
(	((	
A	((	A
-	((-	A
(	((-(	A
B	((-(	AB
+	((-(+	A
C	((-(+	ABC
)	((-	ABC+
/	((-/	ABC+
F	((-/	ABC+F
)	(	ABC+F/-
^	(^	ABC+F/-

C	(^	ABC+F/-C
+	(+	ABC+F/-C^
G	(+	ABC+F/-C^G
/	(+/	ABC+F/-C^G
E	(+/	ABC+F/-C^GE
)	Empty	**ABC+F/-C^GE/+**

[4] Evaluation of the postfix expression: **[Mark-2]**

ABC+F/-C^GE/+ [A=10, B=5, C=3, F=2, G=16, E=4]



b. Write an algorithm/ C-function to identify the position of an unmatched parenthesis in a given mathematical expression using stack. **[Mark-4]**

Let stack1[]: Character array meant for storing '(' (with its top as stack1_top) and stack2[]: int array meant for storing position of '(' (with its top as stack2_top).

```
void check_find_unbalanced_parenthesis(char *s)
{
      int pos, p;
      Char ch;
      for (pos = 0; pos < s[pos]!='\0'; pos++) {

      // if s[pos] is opening bracket then push '(' onto stack1 and position onto stack2
      if (s[pos] == '(')
          {
                push(Stack1,s[pos]);// push():insert an element into the stack
                 push(Stack2,pos); //keep track of position of '('
          }
      // if s[pos] is closing bracket ')'
       else if (s[pos] == ')')
          {
              if(stack1_top != -1)
               {
                    ch=pop(Stack1); // pop(): delete one '(' from the stack1
                    p=pop(Stack2);
               }
```

```
                else
                    printf("The expression is unbalanced at %d\n",pos);
            }//end of else if
  }// end of for
  if(stack2_top != -1)
  {
        p=pop(Stack2);
        printf("The expression is unbalanced at %d\n",p);
    }
}
```

4. Write a C function to perform the following operations on the data stored in a triplet format for a sparse matrix.

(a) The number of rows present in the header node/ first row in the triplet format should be reduced by the number of rows with complete zero value present from the bottoms of the sparse matrix.[Note: If two number of rows present in the bottom of the sparse matrix are completely zero, then the number of rows in the triplet format should be reduced by two]

(b) The number of columns present in the header node/ first row in the triplet format should be reduced by the number of columns with complete zero value present from the right of the sparse matrix.[Note: If two number of columns present from the right of the sparse matrix are completely zero, then the number of columns in the triplet format should be reduced by two]

(c) After reducing the number of rows and number of columns for two such sparse matrix in their respective triplet format, add two sparse matrix (converted into triplet format) with reduced rows and columns.

Example:

Suppose a given sparse matrix is :

0	1	0	0	0	**0**
0	0	1	0	1	**0**
1	0	0	0	0	**0**
0	1	1	1	0	**0**
0	**0**	**0**	**0**	**0**	**0**
0	**0**	**0**	**0**	**0**	**0**

Evaluation Scheme – Full mark for the correct answer and partial marks to be awarded depending on the correctness of the steps.

Solution:

```
struct node                 struct head
{                           {
   int row, col, ele;          int trow, tcol, tele;
   struct node *next;          struct node *next;
};                          };
```

Note: Reduction of size of Sparse Matrix can be implemented using linked list [or array] depending upon number of non-zero rows and columns. Sample answer contains linked list implementation. **[Mark-4]**

/*reduse_matrix() is finding the maximum row and maximum column value present in the sparse matrix */

```
reduce_matrix(struct head *h){
      if(h->next==NULL) return;
      int max_r, max_c;
            max_r = h->next->row;
            max_c = h->next->col;
      struct node *ptr;
      for(ptr=h->next->next; ptr!=NULL; ptr=ptr->next){
            if(ptr->row > max_r)
                  max_r = ptr->row;
            if(ptr->col > max_c)
                  max_c=ptr->col;
      }
      h->trow=max_r+1;
      h->tcol=max_c+1;
}

main()
{
   struct head *head1=NULL, *head2=NULL, *head3=NULL;
   // Take input for sparse matrix-1: head1;
   // Take input for sparse matrix-2: head2;
   reduce_martix(head1);
   reduce_martix(head2);
   head3 = add_sparse(head1, head2);
   //display resultant sparse matrix
}
```

Note : Sparse Matrix Addition can be done using linked list [or array]. **[Mark-6]**

5. Let there is an one dimensional integer array where each value stored that many number of times in a continuous manner. For example, the following array, where 4 is stored 4 times, 3 is stored 3 times, 6 is stored 6 times, 2 is stored 2 times and again another 2 is stored 2 times, and so on.

   4,4,4,4,3,3,3,6,6,6,6,6,6,2,2,2,2,5,5,5,5,5

   You need to write an insert function which takes an integer value along with a position where that integer value is going to be inserted. But no integer value should not be inserted in the middle of any other continuous value sequence.

   If the position given by the user is in the beginning position of any continuous sequence then the new value will be inserted at that position only, otherwise the new value will be inserted at the end of that continuous sequence.

For example, in the given array(**index of the array starts with 0**), if the user wants to insert an element 5 at position 2, then the resultant array will be as follows.

Input Array : 4,4,4,4,3,3,3,6,6,6,6,6,6,2,2,2,2,5,5,5,5,5
Output Array : 4,4,4,4,5,5,5,5,5,3,3,3,3,6,6,6,6,6,6,2,2,2,2,5,5,5,5,5

If the user wants to insert an element 5 at position 7, then the resultant array will be as follows.

Input Array : 4,4,4,4,3,3,3,6,6,6,6,6,6,2,2,2,2,5,5,5,5,5
Output Array : 4,4,4,4,3,3,3,5,5,5,5,5,6,6,6,6,6,6,2,2,2,2,5,5,5,5,5

If the user wants to insert an element 5 at position 14, then the resultant array will be as follows.

Input Array : 4,4,4,4,3,3,3,6,6,6,6,6,6,2,2,2,2,5,5,5,5,5
Output Array : 4,4,4,4,3,3,3,6,6,6,6,6,6,2,2,5,5,5,5,5,2,2,5,5,5,5,5

Evaluation Scheme – Full mark for the correct answer and partial marks to be awarded depending on the correctness of the steps.

Solution: **[Mark-10]**

```
//count: number of elements currently present in array
//val, p: value val to be inserted at p position of array arr
//Assume that count>0 and -1<p<=count

int arr[MAX],count;
void insert(int val, int p){
    int exact_pos=p, i, j;
    i = p;

   //find the beginning of the element to whom p is pointing
   while(i>0 && arr[i-1]==arr[i])
        i--;

    //j is the number of occurrence of arr[i]; place the j at right occurrence
    j=0;
    while(arr[i] * j + i < p)
        j++;
    exact_pos=arr[i] * j + i;

    //shifting the element for insertion purpose
    for(i=count-1; i>=exact_pos; i--)
        arr[i+val]=arr[i];

    //inserting element at right position
    for(i=0; i<val; i++)
        arr[exact_pos+i]=val;
    count +=val;
}
```