1a. What is a data structure? What is a linear and non-linear data structure. What is an algorithm? Explain the criteria an algorithm must satisfy.

Definition (Data Structure) :—
A logical or mathematical model of a particular organisation of data. Choice is made based on
(i) Relationships of Data and (Nature of Data)
(ii) Effectively processing the Data.

Data Structure is a particular way of organisation (organising) data in computer so that it can be used effectively. Any datastructure is designed to organise the data to suite a specific purpose so that it can be accessed and worked in appropriate way.

Linear DS:
Traverses the data elements sequencially. One one element can directly be reached. Eg: Arrays, linked list.

Non-linear DS:
A data is connected to several other data items so tha a given data structure has the possibility to reach one or more data items. Eg: Trees, Graphs.

Algorithm Specification

Defination: An algorithm is a finite set of instruction that if followed accomplishes a particular task.

All the Algorithms must satisfy the following criteria:

1. Input : There are zero or more quantities that are externally supplied.
2. Output: At least one quantity is produced.
3. Definiteness: Each instruction is clear and unambiguous.
4. Finiteness: For all cases of the problem, the algorithm terminates after a finite number of steps.
5. Effectiveness: Every instruction must be basic enough to be carried out and must be feasible.

1b. Write about dynamic memory allocation functions.

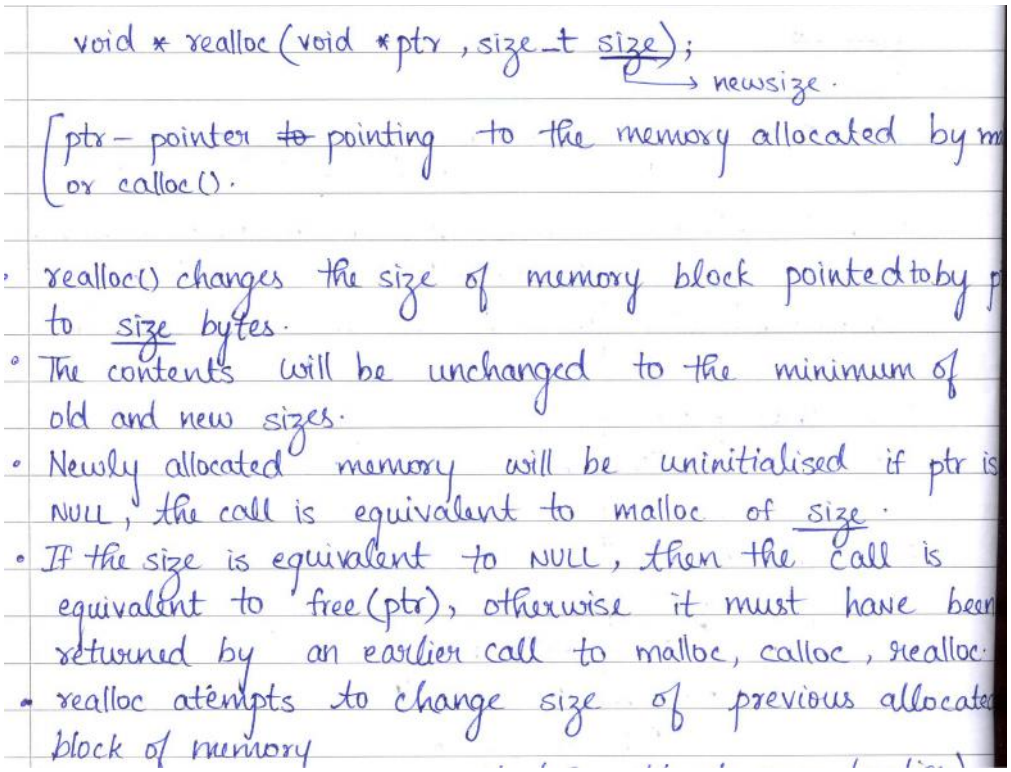
malloc(): allocates a continuous block of memory which is not cleared. If enough continuous memory is available and returns a pointer (void *) to the block of memory otherwise returns NULL

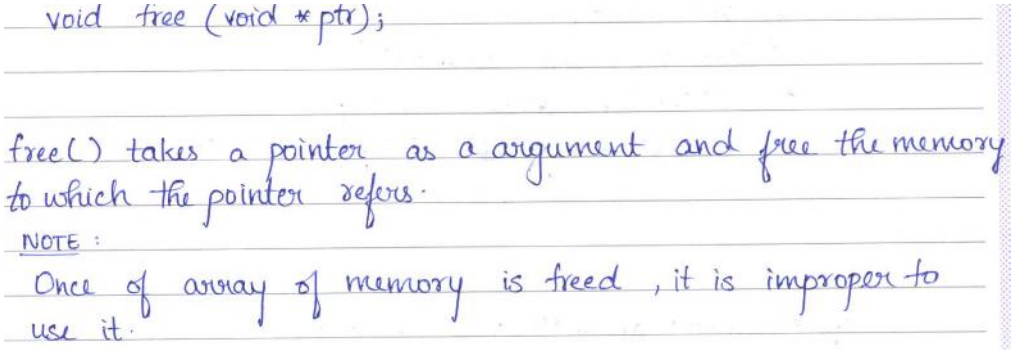
Eg: p holds the address of the first element of the block of 2n bytes.

```
void *calloc(size_t nmemb, size_t size);
                  → elements.   → size bytes
```

calloc() allocates memory for an array of n members (nmemb) elements of size bytes each and returns a pointer to the allocated memory and the memory is set to zero.

```
Eg:- char *cp;
     cp = calloc(100, sizeof(char));
```





3. a. Consider two polynomials, A(x)=4x15+3x4+5 and B(x)=x4+10x2+1

Show diagrammatically how these two polynomials can be stored in a 1-D array. Also give its C representation

3.b. Write a function to add two polynomials

Since each terms has two values — one coefficient, other expo

we use structures:

```
#define MAX_TERMS 100
typedef struct
{
    float coef;
    int expon;
} polynomial;
```

(polynomial → user defined type)

```
polynomial terms[MAX_TERMS];
int avail = 0;
```

(type: polynomial, name: terms, size: MAX_TERMS)

Eg:- $A(x) = 4x^5 + 3x^2 - 2$

	term[0]	term[1]	term[2]
coef	4	3	-2
expon	5	2	0

Adding two polynomials:

Eg: $A(x) = 4x^5 + 3x^3 + 5x^2 + 6$

$B(x) = 2x^5 + 8x$

$D(x) = A(x) + B(x)$

$= 6x^5 + 3x^3 + 5x^2 + 8x + 6$

$A(x) = 4x^5 + 3x^3 + 8x + 6$

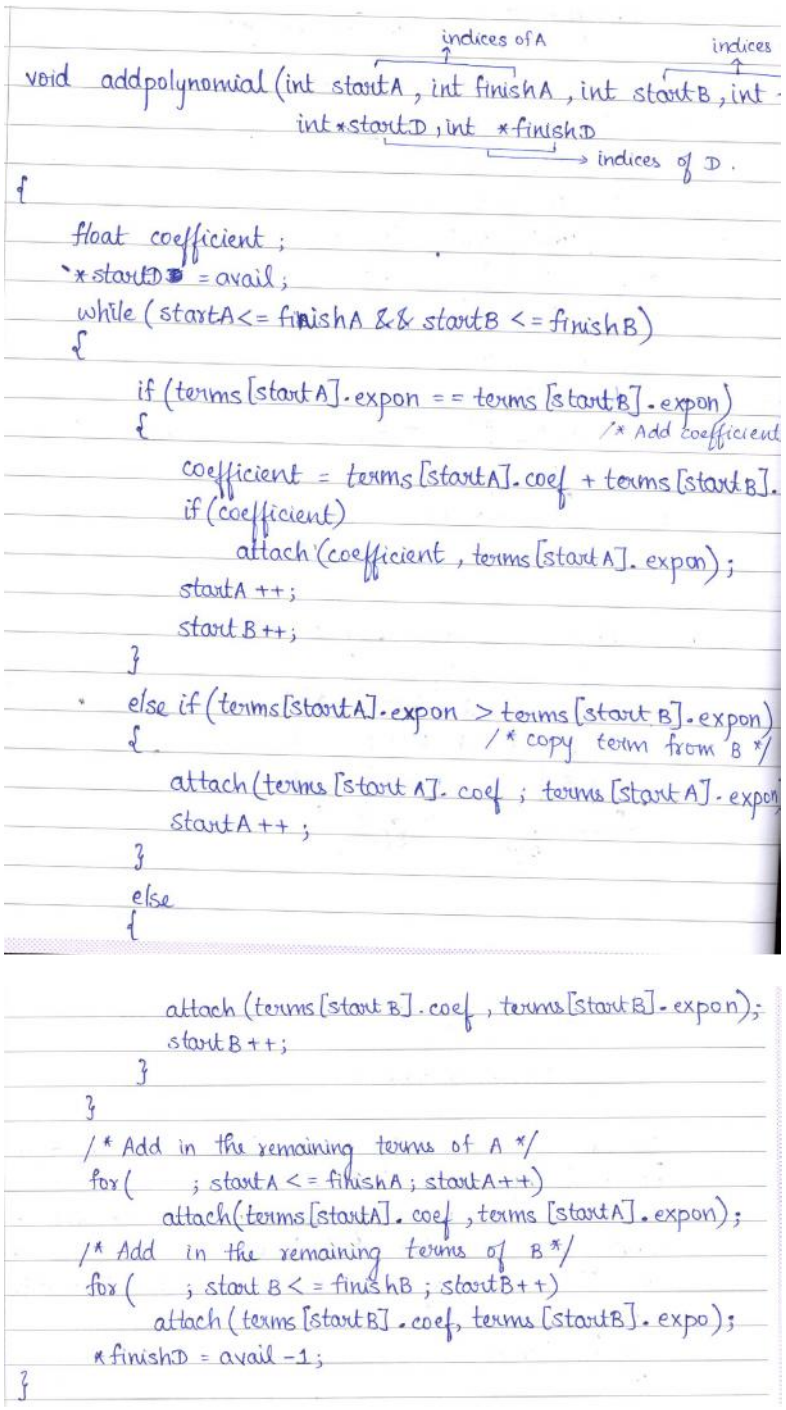
$B(x) = 2x^5 + 5x^2$

Representing in 1-D Array :-

	startA			finishA	startB	finishB	avail
coef	4	3	8	6	2	5	
expon	5	3	1	0	5	2	
	0	1	2	3	4	5	6

A(x) B(x)

```
void attach(float coefficient, int exponent)
{
    if (avail >= MAX_TERMS)
        exit(0);
    terms[avail].coef = coefficient;
    terms[avail].expon = exponent;
    avail++; // increment the index
}
```



```
                                    indices of A                    indices
void addpolynomial(int startA, int finishA, int startB, int
                   int *startD, int *finishD
                                                  → indices of D.
{
    float coefficient;
    *startD = avail;
    while (startA <= finishA && startB <= finishB)
    {
        if (terms[startA].expon == terms[startB].expon)
        {                                          /* Add coefficient
            coefficient = terms[startA].coef + terms[startB].
            if (coefficient)
                attach(coefficient, terms[startA].expon);
            startA++;
            startB++;
        }
        else if (terms[startA].expon > terms[startB].expon)
        {                                          /* copy term from B */
            attach(terms[startA].coef; terms[startA].expon
            startA++;
        }
        else
        {
            attach(terms[startB].coef, terms[startB].expon);
            startB++;
        }
    }
    /* Add in the remaining terms of A */
    for(    ; startA <= finishA; startA++)
        attach(terms[startA].coef, terms[startA].expon);
    /* Add in the remaining terms of B */
    for(    ; startB <= finishB; startB++)
        attach(terms[startB].coef, terms[startB].expo);
    *finishD = avail-1;
}
```

4a. Write Knuth Morris pattern matching algorithm

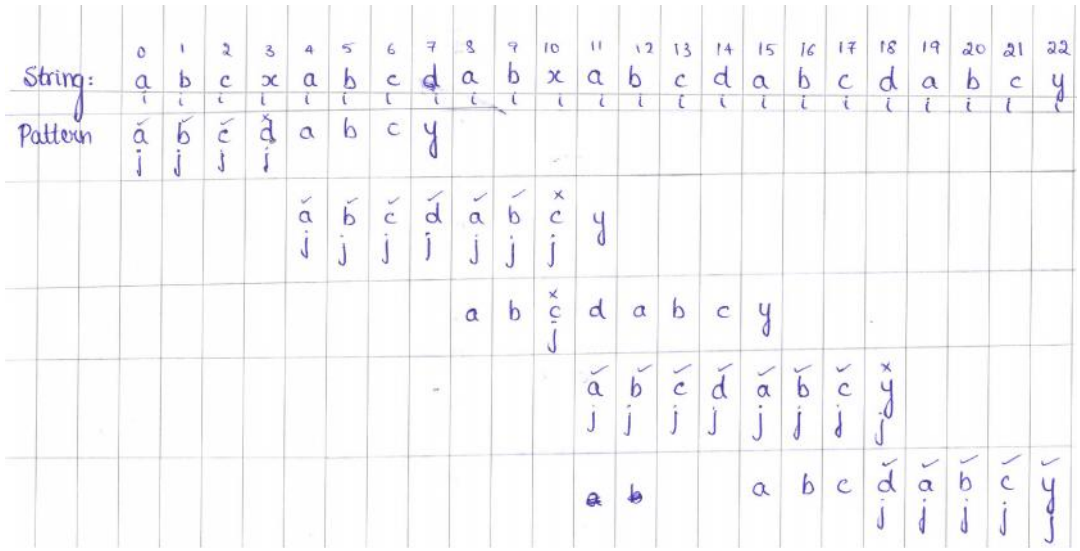
4b. Apply Knuth Morris pattern matching algorithm the same to search pattern ‘abcdabcy’ in the text ‘abcxabcdabxabcdabcdabcy’.

```
int kmpm(char *string, char *pattern)
{
    int i=0, j=0;
    int sl, pl;
    sl = strlen(s);
    pl = strlen(p);
    while (i<sl && j<pl)
    {
        if (s[i] == p[j])
            i++, j++;
        else
        {
            j = FF(j-1);
            if(j == 0)
                i++;
        }
    }
    return ((j== pl) ? (i-pl) : -1);
}
```

not found

found.

Pattern	a	b	c	d	a	b	c	y
F.F	0	0	0	0	1	2	3	0



5a. Write a fast transpose algorithm to transpose the given sparse matrix

```
void fastTranspose (term a[], term b[])
{ /* the transpose of a is placed in b */
  int rowTerms [MAX_COL], startingPos [MAX_COL];
  int i,j, numCols = a[0].col , numTerms = a[0].value;
  b[0].row = numCols ; b[0].col = a[0].row;
  b[0].value = numTerms;
  if (numTerms > 0)        /* nonzero matrix */
  {
      for(i=0; i<numCols; i++)
          rowTerms[i] = 0;
```

```
      for(i=1 ; i<= numTerms ; i++)
          rowTerms [a[i].col]++;
      startingPos[0] = 1;
      for (i=1; i<numCols ; i++)
          startingPos[i] = startingPos[i-1] + rowTerms [i-1];
      for (i=1; i<= numTerms; i++) {
          j = startingPos[a[i].col]++;  b[j].row = a[i].col;
          b[j].col = a[i].row; b[j].value = a[i].value;
      }
  }
```

5b. Express the given sparse matrix as triplets and find its transpose.

sparse matrix :- triple

	row	col	value
a[0]	6	6	8
a[1]	0	0	15
a[2]	0	3	22
a[3]	0	5	-15
a[4]	1	1	11
a[5]	1	2	3
a[6]	2	3	-6
a[7]	4	0	91
a[8]	5	2	28

Transpose of sparse matrix :- tri

	row	col	value
b[0]	6	6	8
b[1]	0	0	15
b[2]	0	4	91
b[3]	1	1	11
b[4]	2	1	3
b[5]	2	5	28
b[6]	3	0	22
b[7]	3	2	-6
b[8]	5	0	-15

6a. Write a recursive algorithm for tower of Hanoi and ackerman's function

```
void main()
{
    int n=3; //Read n
    toh(n,'a','b','c');
}
void toh(int n, char a, char b, char c)
{
    if(n==0)
        return;
    else
    {
        toh(n-1,'a','c','b');
        printf("Move the disks from %c to %c ,a,b);
        toh(n-1,'c','b','a');
    }
}
```

```
int ackerman(int m, int n)
{
    if (m==0) return n+1;
    else if(n==0) return ackerman(m-1, 1);
    else return ackerman(m-1,ackerman(m,n-1));
}
void main()
{
    int m,n;
    // Read m and n
    printf("A(%d,%d) = %d ; m, n, ackerman(m,n));
}
```

6b. Convert the infix expression to postfix form

i) (a+b)*d+e/(f+a*d)+c, ii)(((a/b)+(d*c))-(a*c))

ab+d*e+fad*+c/

ab/dc*+ac*-

7. Write an algorithm to implement a stack using dynamic array whose initial capacity is 1 and array doubling is used to increase the stack's capacity(that is dynamically relocate twice the memory) whenever an element is added to a full stack. Implement the operations-push, pop and display

```
typedef struct
{
    int ele;
}STACK;

STACK *stack;                                   top
stack = malloc(sizeof(*stack)
int capacity = 1;
int top = -1;
void push(int item)
{
    if(top >= capacity-1)
            stackFull();
    else
        stack[++top] = item;
}
void stackFull()
{
    stack = realloc(stack, 2*capacity, *sizeof(*stack));
    capacity *= 2;                              size
}
int pop()
{
    if(top == -1)
        stackEmpty();
    else
    {
        return stack[top--];
    }
}
void stackEmpty()
{
    printf("Stack Underflow");
    exit();
}
```

8. List the disadvantages of linear queue and explain how it is solved in circular queue. Give the algorithm to implement a circular queue with suitable example.

```
#define SIZE 8
typedef struct
{
    int ele;
} QUEUE;
QUEUE cqueue[SIZE];
```
⇒ int cqueue[SIZE]

```
int front = rear = -1;
void addcq (QUEUE item)
{
    if (front == (rear + 1) % SIZE)
        cqueueFull();
    else
    {
        rear = (rear + 1) % SIZE;
        cqueue[rear] = item;
        if (front == -1)    /* First insert */
            front = front + 1;
    }
}
```

```
QUEUE deletecq()
{
    QUEUE item;
    if (front == -1)
        queueEmpty()
    else
    {
        item = cqueue[front];
        if (front == rear)  /* Only one element */
            front = rear = -1;
        else
            front = (front + 1) % SIZE;
    }   return item;
}
```

```
void queueFull()
{
    printf("Queue is Full");
    exit();
}
void queueEmpty()
{
    printf("Queue is Empty");
    exit();
}
```