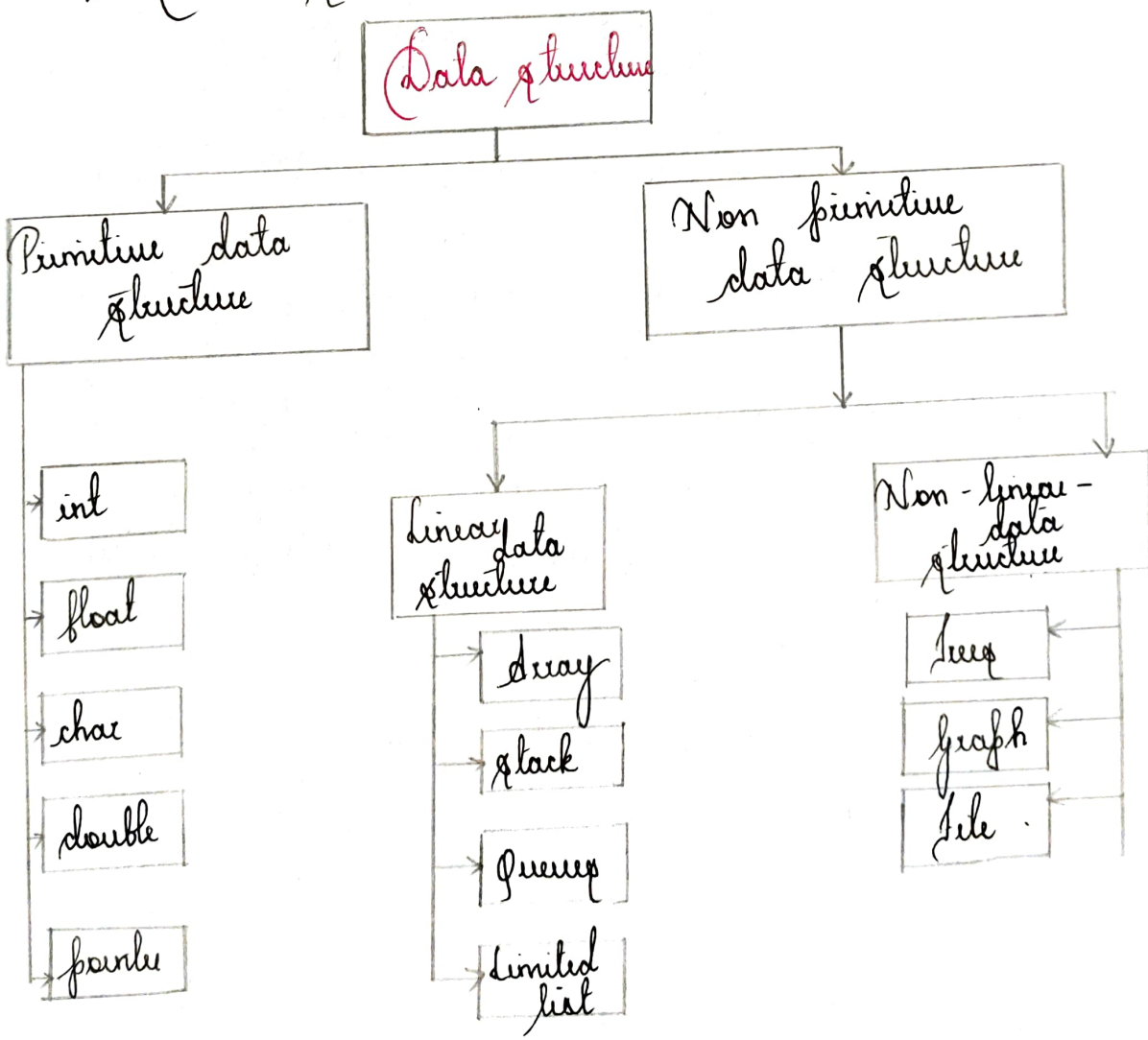
Module - 1

Q1] Define data structure. List and explain the classification of data structure [17CS33 Aug | Sep 2020] [15CS33 Aug | Sept 2019]

Data structure is a logical or mathematical model of storing data and organizing data in a particular way in a computer required for designing and implementing efficient algorithm and program development.

★ Data structure can be classified as



* Primitive data type
    * These are fundamentaly standard data-type
    * These are used to represent single value.
    * These are again classified as

* int
* float
* char
* double
* pointer

* Non-primitive data type
    * They are derived from primitive-data type.
    * They are used to store group of values.
    * They are again classified as

* Linear data structure
    A data structure is said to be linear, if its elements are classified in sequential order.
    * They are again classified as

→ Arrays
→ Stacks
→ Queues
→ Linked list.

* Non-linear data structure

A data structure is said to be non-linear data structure if its elements are not classified in sequential order.

* They can be again classified as

→ Trees
→ Graph
→ File.

Q2] Write Bubble sort program with algorithm. (17CS33 Dec 2018/Jan 2019)

Program:

```
#include <stdio.h>
void main()
{
        int a[50], i, j, n, temp;
        printf("enter the number of elements in array\n");
        scanf("%d", &n);
        printf("Enter %d elements\n");
        for(i=0; i<n; i++)
                scanf("%d", &a[i]);
        printf("Array elements before sorting are: \n");
        printf("%d\t", a[i]);
```

```
for(i=0; i<n ; i++)
{
        for(j=0; j<n-1; j++)
        {
                if (a[j] > a[j+1])
                {
                    temp = a[i];
                    a[i] = a[i+1];
                    a[i+1] = temp;
                }
        }
}

for(i=0;i<n; i++)
{
        printf("Array after sorted \n");

}
```

Algorithm

step 1: set pointer to 1 [Initialise pointer]

step 2: If Data [PTR] > Data [PTR+1]
Interchange Data [PTR] and Data [PTR+1]
End of If structure

step 3: PTR = PTR+1 [end of inner loop]

step 4: Exit.

3. Define Data structure. Give its applications. What are the basic operation that can be performed on data structure. [17CS33 June/July 2019]

Data may be organised in many different ways; the logical or mathematical model of a particular organisation of data is called data structure.

Basic operation that can be performed on data structure

* Data appearing in our data structure are processed by means of certain operations

* Traversing: accessing each record exactly once so that certain items may be processed.

* Searching: finding the location of the record with a given key value or finding the location of all records which satisfy one / more condition

* Insertion: Adding a new record into the data structure.

* Deletion: Removing a record from the structure

Other:

* sorting: arranging the records in ascending / descending order.

* merging: combining two data structure.

Application of data structure are:

* The type of operation on a certain data structure makes it useful for specific task.

→ Arrays

* Arrays are used in storing list of data elements belonging to same data type.
* It act as storage for other data structure, binary tree element of fixed count and matrices

→ Linked list

* Implements stacks, queues, binary trees and graph of predefined size.
* Implement dynamic memory management function of operating system
* Polynomial implementation for mathematical operation.
* circular linked list is used to implement OS and used in a slide show where a user wants to go back to first slide after last slide is displayed.
* circular queue is used to maintain the playing sequence of multiple players in a game.

→ Stacks

★ Temporary storage structure for recursive operations

★ Evaluation of infix expression to postfix expression, arithmatic expression, infix to prefix

★ Recursive function call.

→ Queues

★ It is used in breadth search operation in graph.

★ priority queues are used in file downloading operations in browser.

★ Call handled by customers in BPO.

→ Trees

★ Implementing the hierarchical structure in computer system like directory and file system.

★ Implementing the navigation structure of a website.

★ Hash tree.

→ Graph

★ Representing network and routes in communication, transportation and travel application.

★ Mapping applications

★ Robotics motion and neural network.

:- Pointers :-

Define pointers ? List the advantages of pointers over array ? How to declare and initialize pointers, Explain with example ?

[18CS32 Aug/Sep 2020] [18CS32 Dec 2019/Jan 2020] [17CS33 Aug/Sep 2020]

→

Pointer :

A variable which holds address of another varieable or a memory location is called "pointer".

Advantages :

* Pointers allow to use dynamic memory allocation.
* Pointers are good for handling big arrays as arguments to functions.
* It allows you to implement sharing without copying. i.e pass by reference.
* Pointers allow modifications, by a function that is not the creator of memory, without using global varieables.

* Pointers have great applications while implementing TREE, LINKED LIST, STACK etc

To declare & intialize the pointer :

Syntax :

datatype * pointervariable

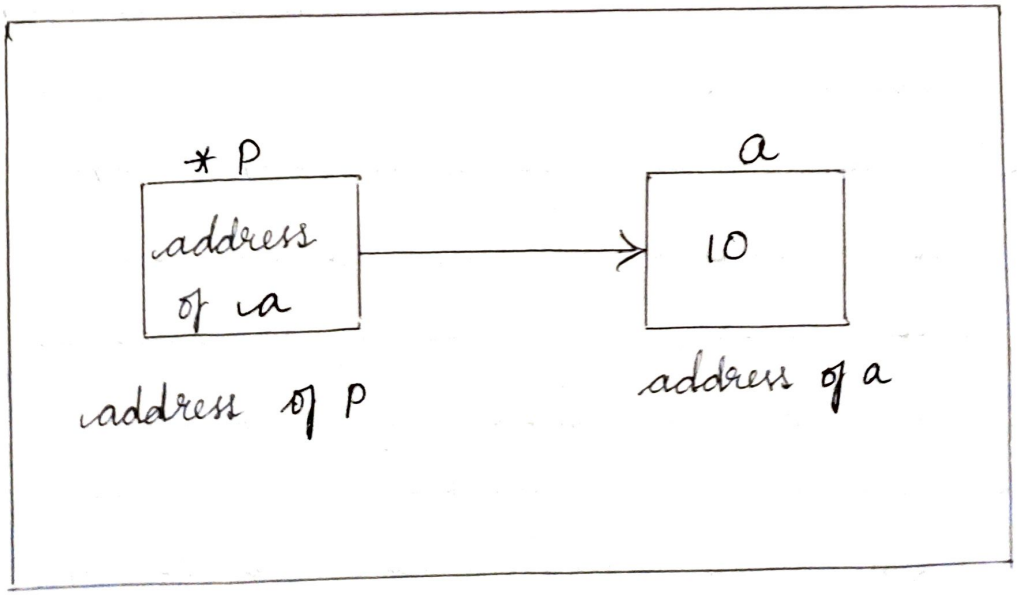
Ex :

```
int * p
float * q
```

Intialization :

```
int a = 10 ;
p = &a        // Initializing point.
```

" point 'p' contains the address of Integer variable 'a' as shown in diagram "



Dynamic memory allocation :

* Explain the four dynamic memory allocation ? [17CS33 - JAN / FEB 2021] [18CS32 JAN / FEB 2021] [16CS33 AUG - SEPT 2020] [18CS32 DEC 2019 / JAN 2020] [17CS33 JUNE - JULY 2019] [17CS33 DEC 2018 / JAN 2019] [18CS32 AUG - SEPT 2020]

→

Dynamic memory allocation :

- DMA is the process of allocating memory space during execution time. i.e Run time.

- Dynamic allocation will be used when we create dynamic arrays, linked lists, trees.

- There are four types of memory allocation

i) Malloc
ii) Calloc
iii) Realloc
iv) Free.

is Malloc :

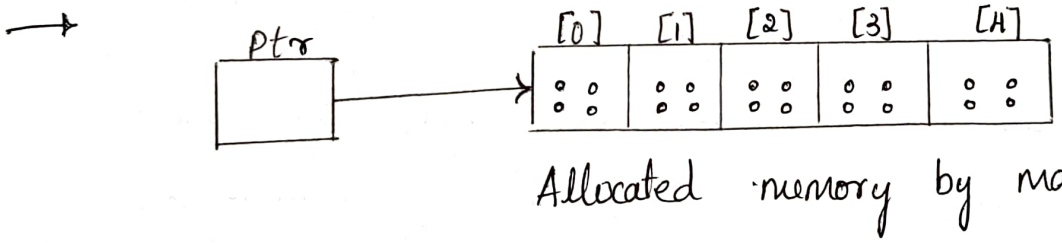
"Memory allocation"

Syntax :

```
int * ptr
ptr = (datatype *) malloc (size);
```

Ex :

```
ptr = (int *) malloc (5);
if (ptr == NULL)
{
    printf ("Insufficient memory \n");
    exit (0);
}
```



Allocated memory by malloc.

ptr → is a point that contains starting address of allocated memory.

* If memory is not allocated succesfully malloc returns . NULL
* It allocates a block of memory.
* It allocates memory space as per requirement.

Calloc :

"Clear and allocation".

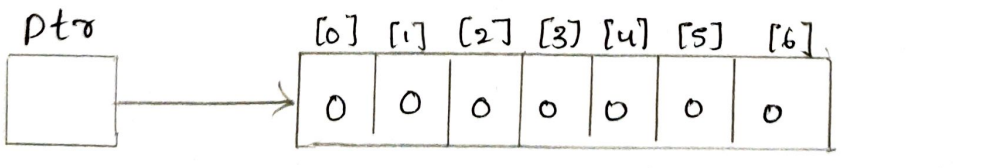
Syntax :

```
int * ptr
ptr = (datatype *) calloc (n, size);
```

Ex :

```
ptr = (int *) calloc (10, 2);
if (Ptr == NULL)
{
        printf ("Insufficient memory \n");
        exit (0);
}
```

* It allocate multiple blocks of memory.
* calloc is similar to malloc, But it initialize the allocated memory to zero.



Allocated memory by calloc

Realloc :

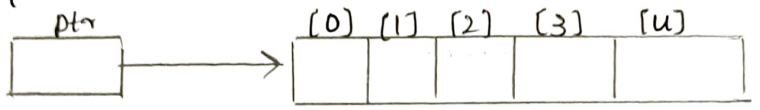
"Re-allocate"

Syntax :

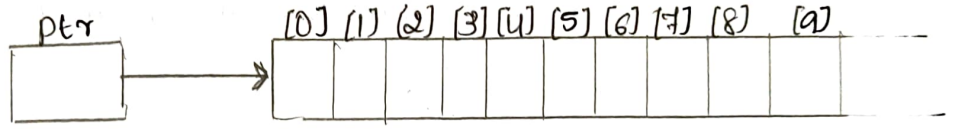
```
ptr = (datatype *) realloc (ptr, newsize);
```

Ex :

ptr = (int *) realloc (ptr ; 50);



ptr = (int *) realloc (ptr, 10); → new size



* It is used to modify the size of allocated block by malloc(), calloc() to new size

* The addition (or) deletion of allocated memory can be done by using 're-allocate'.

Free :

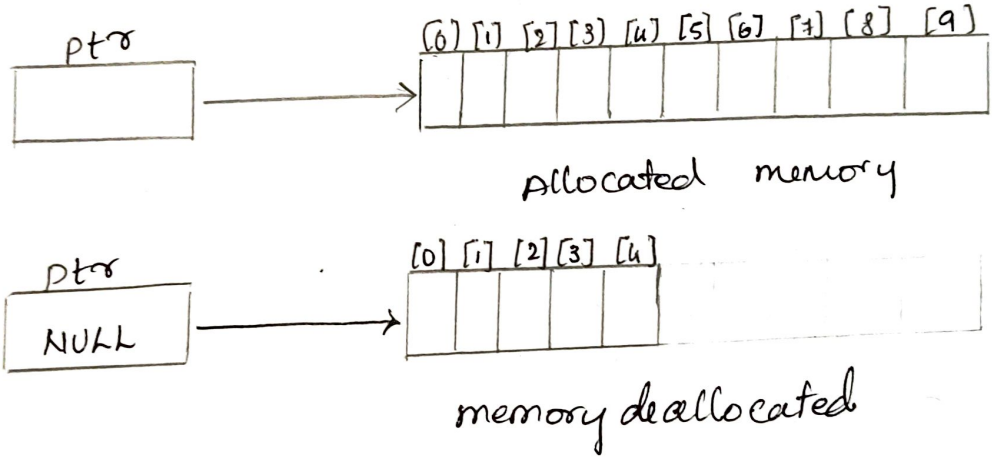
" Dealocate memory "

Syntax :

```
free (ptr);
```

Ex :

```
int * ptr
ptr = (int *) malloc (10);
frec (ptr)
ptr = NULL
```



→ It deallocate the allocated memory which was done using malloc(), calloc() & realloc

Write the algorithm for inserting for Element into a linear array and deleting an Element from linear array. [17CS33 Aug/Sep 2020]

Algorithm for Inserting Element in K^{th} position in an linear array with N Elements, and using J as a Counter

Insert (LA, N, K, Item)

Step 1: [Initialize] set J := N

Step 2: Repeat steps 3 and 4 while J ≥ K

Step 3: [Move J^{th} Element downward] set LA[J+1] := LA[J]

Step 4: [Decrease counter] set J := J - 1

[End of step 2 loop]

Step 5: [Insert Element] Set LA[K] := Item

Step 6: [Reset N] Set N := N + 1

Step 7: Exit.

Algorithm for delete the K^{th} Element from a linear array LA and assigns it to a variable ITEM.

DELETE (LA, N, K, ITEM)

Here LA is a linear array with N Elements and K is a positive integer such that K ≤ N. This algorithm delete the K^{th} element from LA.

1. Set ITEM := LA[K]
2. Repeat for J = K to N - 1:
   [Move J + 1st Element upward] Set LA[J] := LA[J+1].
   [End of loop.]
3. [Reset the number N of Elements in LA]. Set N := N - 1.
4. Exit.

what are arrays ? list and Explain any different operation that can be carried out in arrays [18CS32 Aug / Sept 2020]

An array is collection of similar data items. the Elements of an array are of same type and Each element can be accessed using same name, but with different index values.

Traversing linear arrays

let A be a collection of data Elements stored in the memory of the computer. Suppose we want to print the contents of Each Element of A or Suppose we want to count the number of Element of A with a given property. this can be accomplished by traversing A, that is by accessing and processing (frequently called visiting) Each Element of A Exactly once.

Algorithm :- (Traversing a linear array) Here LA is a linear array with lower band LB and upper bound UB. This algorithm transverse LA applying an operation PROCESS to Each Element of LA.

1. [Intialize Counter.] Set K := LB.
2. Repeat Steps 3 and 4 while K ≤ UB.
3. [Visit Element.] Apply PROCESS to LA[K].
4. [Increase Counter.] Set K := K+1.

   [End of Step 2 loop.]

5. Exit.

Inserting and deleting

Inserting refers to a operation of adding another Element to the collection.

Deleting refers to the Operation of removing one of the Elements from collection.

Algorithm :- (Inserting into a linear array)

INSERT (LA, N, K, ITEM)

Here LA is a linear array with N Elements and K is a position integer such that K ≤ N, this algorithm inserts an element ITEM into the k^{th} position in LA.

1. [Initialize Counter] Set J := N
2. Repeat Steps 3 and 4 while J ≥ K
3. [move J^{th} element downward.] Set LA[J+1] := LA[J]
4. [Decrease Counter.] Set J := J-1

   [End of Step & loop.]

5. [Insert Element.] set LA[K] := ITEM.

6. [Reset N.] set N := N+1

7. Exit

Algorithm: (Deleting from a linear array) DELETE(LA, N, K, ITEM)

Here LA is a linear array with N elements and K is a position integer such that K ≤ N. This algorithm deletes the K^{th} Element from LA.

1. Set ITEM := LA[K].

2. Repeat for J = K to N-1:

   [Move J+1st Element upward.] Set LA[J] := LA[J+1].

   [End of loop.]

3. [Reset the number N of Elements in LA.] Set N := N-1.

4. Exit.

Sorting :- A sorting techniques that is typically used for sequencing small lists. It starts by comparing the first item to the second, the second to the third and so on until it finds one item out of order. It then swaps the two items and starts over

Bubble sort :- Bubble sort in C is a sorting algorithm where we repeatedly iterate through the array and swap adjacent Elements that are unordered. we repeat this until the array is sorted.

Algorithim :- (Bubble Sort) BUBBLE (DATA. N)

Here DATA is an array with N elements. This algorithim Sorts the elements in DATA.

1. Repeat Steps 2 and 3 for K = 1 to N - 1.
2. Set PTR := 1. [Initializes pass pointer PTR.]
3. Repeat while PTR ≤ N - K : [Executes pass.]
   a) If DATA[PTR] > DATA[PTR + 1], then:
      Interchange DATA[PTR] and DATA[PTR + 1].
      [End of If Structure].
   b) Set PTR = PTR + 1.
   [End of inner loop.]
   [End of Step 1 outer loop.]
4. Exit.

Searing :-

Searching has to check for an element from any data structure where the data is stored.

Linear Search

linear Search is a method for finding an element within a list. It sequentially checks each element of the list until a match is found or the whole list has been searched.

Algorithm :- (Linear Search) LINEAR (DATA, N, ITEM, LOC)

Here DATA is a linear array with N elements, and ITEM is a given item of information. this algorithm finds the location LOC of ITEM in DATA, or sets LOC := 0 if the search is unsuccessful.

1. [Insert ITEM at the End of DATA]. Set DATA[N+1] := ITEM
2. [Initialize counter.] set LOC := 1.
3. [Search for ITEM.]
   Repeat while DATA [LOC] ≠ ITEM:
   Set LOC := LOC + 1.
   [End of loop]
4. [Successful?] If LOC = N+1. then : set LOC = 0
5. Exit.

Binary Search:

Binary Search is a Search algorithm that is used to find the position of an Element in a Sorted array.

Algorithm :-

(Binary Search) BINARY (DATA, LB, UB, ITEM, LOC)

Here DATA is Sorted array with lower bound LB and upper bound UB. and ITEM is a given item of information the variables BEG, END and MID denote. respectively, the beginning End and middle locations of a Segment of Elements of DATA. This algorithm finds the location LOC of ITEM in DATA or Sets LOC = NULL.

1. [Initialize Segment Variables.]
   Set BEG := LB. END := UB and MID = INT((BEG + END)/2).
2. Repeat Steps 3 and 4 while BEG ≤ END and DATA[MID] ≠ ITEM
3. If ITEM < DATA[MID], then:
   set END := MID - 1.
   Else.
   Set BEG := MID + 1.
   [End of If structure].
4. Set MID := INT((BEG + END)/2).
   [End of step 2 loop.]
5. If DATA[MID] = ITEM. then:
   Set LOC := MID.
   Else :-
   Set LOC := NULL.
   [End of If Structure.]
6. Exit.

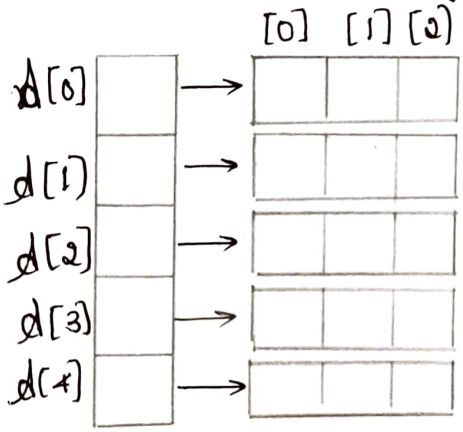
12] Explain the representation of two-dimensional array in memory. [ITCS33 Dec 2018/2019]

In two dimensional array representation, normally array of arrays representation is represented as one dimensional array, in which each element is, itself, one dimensional array.

eg: int d[5][3];

we actually create one-dimensional array d whose length is 5; each element of x is one dimensional array whose length is 3.

* Two dimensional array can be represented as



C finds the element x[i][j] by first accessing the pointer in x[i]. This pointer gives the address, in memory, of the zeroth element of row i of the array. Then by adding j * size of (int) to this pointer, the address of [j]th element of row is determined.

C provides two additional memory allocation function calloc and realloc that are useful in the context of dynamically allocated arrays. The function calloc allocates a user-specified amount of memory and initializes the allocated memory to 0; a pointer to start of allocated memory is returned. In case there is insufficient memory to make allocation, the returned value is Null.

13 What is structure? What is difference between Structure and array? [18CS32 DEC 2019/JAN 2020]

Structure is defined as a group of dissimilar or heterogenous data items, where items can be of a different data item type.

Array	Structure
• Array refers to a collection consisting of elements of homogeneous data type	• Structure refers to a collection consisting of elements of heterogeneous data type.
• Array uses subscript or "[]" for element access	• Structure uses "." for element access
• Array is pointer as it points to the first element of the collection.	• Structure is not a pointer
• Array size is fixed	• Structure size is not fixed
• Array declaration is done simply using [] and not any keyword	• structure declaration is done with the help of "struct" keyword.
• Array is a non primitive datatype	• Structure is user defined datatype
• Array traversal and searching is easy and fast	• Structure traversal and searching is complex and slow.
• Array elements are stored in continuous memory locations	• Structure elements may or may not be stored in a continuous memory location.
• data-type array-name [size];	• Struct struct-name { data-type 1 ele1; data-type 2 ele 2; };

Explain different type of structure declaration with Example [18CS32 DEC 2019 / JAN 2020]

Types of structure

1) Tagged structure

The structure associated with a tag name are called tagged structures

Syntax:-

```
Struct tagname
{
  data type 1. member 1;
  data type 2. member 2;
  :
  :
};
Struct tagname variable;
```

Eg:-

```
Struct person
{
  char Name [20];
  int age;
  char gender;
  float salary;
};
Struct person P;
```

2. Without tag

The structures associated without any tag name are called without tag structures

Eg:-

```
Struct
{
  int Roll;
  float percent;
  char grade;
} x, y, z;
```

Here x, y, z are the Structure variable

Syntax:-

```
Struct
{
  datatype member;
  .
  .
  .
} variable;
```

3. Type defined

The structure associated with key word typedef is called type defined Structure

Syntax:-

```
type def Struct
{
  data type member 1;
  .
  .
  .
} type ID;
```

Eg:-

```
type - def struct
{
    char Name [20];
    int Roll;
} student;
```

Polynomials

Represent the following two polynomial diagrammatically using array ? (oo)

Show diagrammatically how these two polynomial can be stored in 1-D array?

[17CS33 JAN/FEB 2021] [17CS33 AUG/SEPT 2020] [15CS33 AUG/SEPT 2020] [17CS33 JUNE/JULY 2019].

→

* $A(x) = 5x^{14} + 3x^3 + 5$

  $B(x) = x^3 + 10x^2 + 2$

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
Coefficient	5	3	5	1	10	2			
exponent	14	3	0	3	2	0			

START END/FINISH START END AVAIL [AVAILABLE]

* $A(x) = 2x^{1000} + 1$

  $B(x) = x^4 + 10x^3 + 3x^2 + 1$

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
Co-efficient	2	1	1	10	3	1		
exponent	1000	0	4	3	2	0		

START END START END AVAIL

$A(x) = 4x^{15} + 13x^4 + 5$

$B(x) = x^4 + 10x^2 + 1$

	[0]	[1]	[2]	[3]	[4]	[5]	[6]
coeficient	4	13	5	1	10	1	
exponent	15	4	0	4	2	0	

START START

END END

Design develop and execute a program in C to accept two polynomials add them and then print the resulting polynomial.

→

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

# define COMPARE (x, y) ((x)<(y)) ? -1 : ((x)==(y)) ? 0 : 1)

# define MAX_TERMS 100

Typedef struct
{
        int coef ;
        int expon ;
} polynomial ;

polynomial term [MAX_TERMS];

int avail = 0 ;

void attach ( int, int ) ;
void padd ( int, int, int, int, int*, int* );
void print ( polynomial terms1 [20], int, int);
void readpoly ( polynomial terms1 [20], int, int );
```

```
Void main()
{
    int m, n, i, start A, start B, start D, finish A,
            finish B, finish D;
    clrscr();

    printf("Enter the no of terms in first poly\n");
    scanf("%d", &m);
    start A = 0;
    finish A = start A + m - 1;

    readpoly(terms, start A, finish A);
    printf("THE POLYNOMIAL A(x) is :\n");
    print(terms, start A, finish A);
    printf("Enter the no of terms in second poly\n");
    scanf("%d", &n);
    Start B = finish A + 1;
    finish B = start B + n - 1;

    readpoly(terms, start B, finish B);
    printf("THE POLYNOMIAL B(x) is \n");
    print(terms, Start B, finish B);
    avail = finish B + 1;

    padd(start A, finish A, Start B, finish B, &
            start D, &finish D);
```

```
    for ( ; startA <= finishA ; startA++)
    attach(terms[startA].coef, terms[startA].expon);
    for ( ; startB <= finishB ; startB++)
    attach(terms[startB].coef, terms[startB].expon);
    * finishD = avail - 1;
}  //end of padd function.

void attach(int coefficient, int exponent)

{
    if (avail >= MAX_TERMS)
    {
      printf("too many terms in the polynomial \n");
      exit(EXIT_FAILURE);
    }
    terms[avail].coef = coefficient;
    terms[avail++].expon = exponent;
}

void print(polynomial terms1[20], int k, int term)
{
  int i;
  printf("\n ------ \n");
  for(i=k; i<=term-1; i++)
  printf("%d x^%d + ", terms1[i].coef, terms1[i].expon);
  printf("%d x^%d \n", terms1[i].coef, terms1[i].expon);
  printf("\n");
}
```

```
    printf ("THE RESULTANT POLYNOMIAL D(x) is :\n");
    print ( terms , start D , finish D );
    getch ();
}
void padd ( int start A , int finish A , int start B , int
            finish B , int* start D, int * finish D)
{
    int coefficient ;
    * start D = avail ;
    while (start A <= finish A && start B <= finish B)
    switch ( COMPARE (terms [start A]. expon,
              terms [start B] . expon)).
    {
        case -1 : attach ( terms [start B]: coef,
                  terms [start B]. expon );
                  start B ++ ;
                  break ;
        Case 0 : Coefficient = terms [start A ].coef+
                  terms [ start B]. coef ;
                if ( coefficient ) ;
                  attach (coefficent , terms[start A].expon);
                  startA++ ;
                  start B ++ ;
                  break ;
        case 1 : attack (terms [start A]. coef, terms [startA].
                  expon);
                  start A++ ;
    }
```

```
void readpoly (Polynomial terms1[80], int start, int end)
{
    int i;
    Printf(" Enter the coeff and expon of term \n");
    for (i= start ; i<= end ; i++)
    scanf ("%d %d", &terms1[i].coef, &terms1[i].expon);
}
```

20 Define Sparse matrix and Explain the triplet representation of Sparse matrix with an Example [17CS33 Jan/Feb 2021]

A Sparse matrix is a matrix in which most of the Elements are zero.

Ex :-

	0	1	2	3	4
0	7	0	0	1	0
1	0	0	2	0	0
2	0	5	0	0	3
3	6	0	0	4	0

In triplet form

R	C	V
4	5	7
0	0	7
0	3	1
1	2	2
2	1	5
2	4	3
3	0	6
3	3	4

21 Define Sparse matrix. Express the following matrix in triplet form and find form and find its transpose.

Matrix A =

15	0	0	22
0	11	3	0
0	0	0	-6
0	0	0	0
91	0	0	0
0	0	28	0

A sparse matrix is matrix in which most of the elements are zero.

Triplet form :-

R	C	V
6	4	7
0	0	15
0	3	22
1	1	11
1	2	3
2	3	-6
4	0	91
5	2	28

Transpose of Sparse matrix

A =

15	0	0	0	91	0
0	11	0	0	0	0
0	3	0	0	0	28
22	0	-6	0	0	0

Triplet form of transpose of Sparse matrix

R	C	V
4	6	7
0	0	15
0	4	91
1	1	11
2	1	3
2	5	28
3	0	22
3	2	-6

Give ADT of Sparse matrix and show with suitable Example. Sparse matrix representation starting as triplet [ISCS33 dug | sept 2020]

ADT of Sparse matrix is

objects :- a set of triples, < row, column, value >. where row and column are integers and form a unique combination, and value comes from the set item.

function :-

for all a, b ∈ Sparse Matrix, x ∈ item, i, j, maxCol, maxROW ∈ index.

Sparse Matrix Create (maxROW, maxCol) ::=

return a Sparse matrix that can hold up to maxItems = maxROW x maxCol and whose maximum row size is maxROW and whose maximum column size is maxCol.

Sparse Matrix Transpose :: - return the matrix produced by interchanging the row and column value of every triple.

Sparse matrix Add (a.b) ::=

if the dimensions of a and b are the same return the matrix produced by adding corresponding items, namely those with identical row and column values. Else return Error.

Sparse matrix multiply (a.b) ::=

if number of columns in a Equals number of rows in b

return the matrix d produced by multiplying a by b according to the formula : d[i][j] = Σ(a[i][k]. b[k][j]) where d(i,j) is the (i,j)th Element.

Else return Error.

Ex:-

	0	1	2	3
0	0	0	2	0
1	0	1	0	1
2	0	7	0	0
3	0	0	0	-9

Triplet form

R	C	V
4	4	5
0	2	2
1	1	1
1	3	1
2	1	7
3	3	-9

Express the Sparse matrix in triplet form [IT CS 33 June/July-19]

Matrix A =

$$\begin{bmatrix} 10 & 0 & 0 & 25 & 0 \\ 0 & 23 & 0 & 0 & 45 \\ 0 & 0 & 0 & 0 & 82 \\ 42 & 0 & 0 & 31 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 30 & 0 & 0 \end{bmatrix}$$

Triplet form of Sparse matrix

R	C	V
6	5	30
0	0	10
0	3	25
1	1	23
1	4	45
2	4	82
3	0	42
3	3	31
5	2	30

Difference between Structure and union. [18CS32 Dec/Jan20]

Structure	Union
* Structure declaration starts with keyword struct	* union declaration starts with keyword union.
* Structure reserves memory for each data member separately	* union reserves memory i.e Equal to maximum data member size amongst all.
* Any data member value can be accessed at any time	* Only one data member can be accessed at a time
* Every field in structure's offset is in increasing order	* The offset of field is same i.e 0.
* All fields are active at a time	* Only one field is active at a time
Ex:- Struct Book { int isbn; float price; char title[20]; } book;	Ex:- union book { int isbn; float price; char title[20]; } book;
* Total memory reserved will be Size of(int) + Size of(float) + (20 * Size of(char)	* Total memory reserved will be Max (Size of(int) + Size of(float) + (20 * Size of (char))

Q 24] Write the pattern matching algorithm to match pattern P with each of substring of T.
[ITCS33 jan/Feb 2021]

Let us assume T be a string and P be the pattern string

Let us consider text string T as

i	0	1	2	3	4	5	6	7	8
T	F	U	N		U	N	C	L	E

consider pattern string as

j	0	1	2	3	4
	U	N	C	L	E

we should allign pattern string to beginning of text string and compare the character of pattern string with character of text string from left to right. If all character are compared and found same, algorithm terminate.

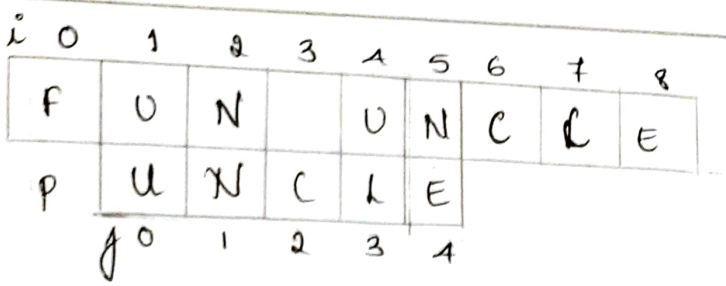
step 1 :

i	0	1	2	3	4	5	6	7	8
T	F	U	N		U	N	C	L	E
P	U	N	C	L	E				
j	0	1	2	3	4				

character 'F' and 'U' are compared and they are not equal.

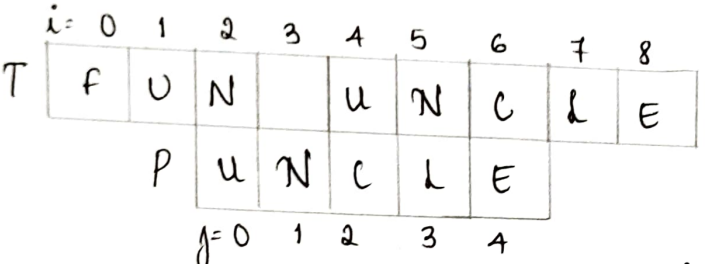
Slide the pattern one step towards right

step 2 :



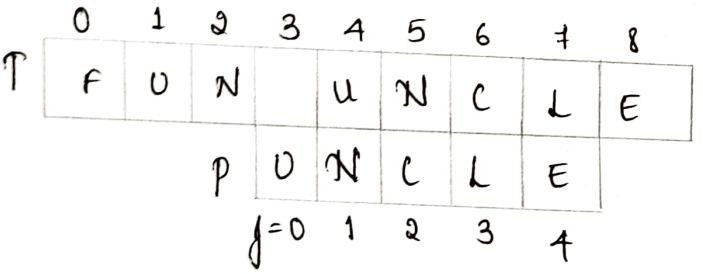
The characters 'U' and 'U', 'N' and 'N' matches but there is mismatch in '-' and 'L' are comparision slide the pattern string towards right.

step 3:



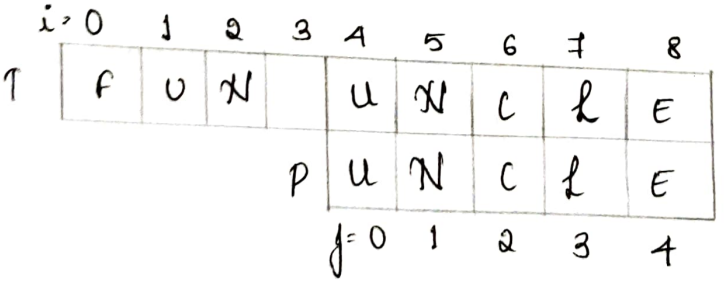
character 'N' and 'U' are compared and not equal. so slide pattern string towards right.

step 4:



character 'U' and '-' are compared and not equal. slide pattern string towards right.

step 5:



Pattern string is found and position i has to be returned.

Q 28] Define strings. List and explain five functions in strings with example [18CS32 Aug/Sep 2020]

Strings can be defined as array of character. The difference between a character array and array of character [string] is that it can be terminated with a special character '\0'-[null character]

*Five operations in strings are [with inbuilt function]
→ string length
→ string concatenate
→ string number copy
→ string copy
→ string compare
→ string number concatenate
→ string number compare

1] string length

*Syntax: strlen(str);

eg: char str[7] = "Beautiful";
int n;
n = strlen(str);

```
#include <stdio.h>
#include <string.h>
void main()
{
    char str[] = "Beautiful";
    int n;
    n = strlen(str);
    printf("string length is :%d", n);

}
```

This function returns length of string str i.e it counts all character upto '\0' but not '\0'.

2] string concatenate

syntax : strcat (s_1, s_2);

eg : char s_1[] = "good";
char s_2[] = "NATURE";
str ((s_1, s_2);
puts (s_1);

working : s_1 is first string
s_2 is second string
The function copies all characters of s_2 to end of s_1. Only condition is size of s_1 must be large enough to store a string whose length is $s_1 + s_2$.

program :

```
# include <stdio.h>
# include <string.h>
void main ()
{
        char s1[30] = "good";
        char s2[7] = "Nature";
        if (strlen(s1) + strlen(s2) < sizeof s1)
                strcat (s1, s2)
        else
                printf ("Error : Invalid concatenate \n");
}
```

output : good nature

③ string copy

syntax : strcpy (dest , src);

eg: char s1[ ] = "good";
char s2[ ];
strcpy (s2 , s1);
puts (s2);

working : destination [dest] → where string has to copied
src → source string

Basic function is to copy n character from source string to destination string.

program :

```
# include <stdio.h>
# include <string.h>
void main()
{
        char s1[5] = "GOOD";
        char s2[5];
        strcpy(s2 , s1);
        printf (" Destination string = %s\n", s2 );
}
```

output : Destination string = GOOD.

④ string number copy

syntax : strncpy (dest , src , n)

where * dest is destination
* src is source
* n is number of bytes to be copied to destination string.

working: Based on value of n, copying activity can be performed as

* If source string is less than n, the entire string is copied from source to destination

* If source string is greater than n, only n characters are copied to destination

* eg: strncpy (dest, src, n);

```
char s1[ ] = "good";
char s2[ ];
strncpy (s2, s1, 2);
puts s2;
```

program:

```
# include <stdio.h>
# include <string.h>
void main()
{
        char s1[5] = "good";
        char s2[5];
        strncpy (s2, s1, 2);
        puts s2;
}
```

output : 60

// Here since sizeof s_1 is greater than n, only n characters are copied.

5. String compare

syntax : strcmp (char s_1[], s_2[]);

eg: char s_1[] = "good";
char s_2[] = "NATURE";
strcmp(s_1[], s_2[])

working : function is used to compare two strings. comparision starts with first character of each string. comparision continues till the corresponding character differ.

program :

```
#include <stdio.h>
#include <string.h>
int main()
{
        char str1[] = "add";
        char str2[] = "sub";
        int result;
        result = strcmp(str1, str2);
        printf("strcmp(str1, str2) = %d\n", result);
        return 0;
}
```

output

strcmp (str 1, str 2) = 0.

⑥ string number compare

syntax : strncmp (char s1[], char s2[], int n)

working : This function is used to compare first n number of character in two strings.

program :

```
# include <stdio.h>
# include <string.h>
void main()
{
        char s1[20] = "strong";
        char s2[10] = "strange";
        if (strncmp (s1, s2, 3) == 0)
                printf ("first three characters are equal\n");
        else if (strncmp (s1, s2, 3) < 0)
                printf ("first three characters of first str is less\n");
        else
                printf ("first three characters of first str is
                                greater\n");
}
```

output : first three characters are equal.

⑦ string number concatinate

syntax strncat (char $s_1[\,]$, char $s_2[\,]$, int n);

working : If length of s_2 is less than n, function copies all character of s_2 to s_1.

If s_2 is greater than n, then first n character of s_2 are copied to end of s_1.

program.

```
# include <stdio.h>
# include <string.h>
void main()
{
        char s1[30], s2[30];
        int n;
        printf("Enter string 1 : \n");
        scanf ("%s", s1[]);
        printf (" Enter string 2 : \n");
        scanf (" %s", s2[]);
        if ( s2[] < n)
            printf (" strncat ( s1[], s2[]) = s1[] ");

        else ( s2[] > n)
            printf (" strncat ( s1[], s2[], n) = s1[]n");

}
```

output

```
Enter string 1: Good
Enter string 2: Phone
                Good Phone.
```

without using inbuilt functions

- string length
- string concatenate
- string copy
- string number copy
- string compare
- string number compare
- string number concatenate

string length

```
# include <stdio.h>
# include <string.h>
void main()
{
        char str[] = "Beautiful";
        int i=0;
        while (s[i]!='\0')
                i++;
        printf ("The string length is : %d\n", i);
}
```

output

string length is : 9.

③ string copy

```
# include <stdio.h>
void main()

{
        char s1[] = "good";
        char s2[];
        int i = 0;
        while (s1[i] != '\0')
        {
                s2[i] = s1[i];
                i++;
        }
        puts(s2);
}
```

output good good.

③ Program on concatinate of two strings without using inbuilt function

```
#include <stdio.h>
#include <string.h>
void main()
{
        char s1[20] = "GOOD";
        char s2[20] = "NATURE";
        int i=0, j=0;
        while (s[i]!= '\0')
                i++;
        while (s2[j] != '\0')
        {
                s1[i] = s2[j];
                j++;
                i++;
        }
        s[j] = '\0';
        puts(s1);
}
```

output

GOOD NATURE

④ string number copy

```
# include < stdio.h >
void main ()
{
        char s1[] = "small city" ;
        char s2[] ;
        int i=0 , count = 1 , n=5;
        while ( s1[i] = '\0' && count <= n)
        {
            s2[i] = s1[i] ;
            i++;
            count++;
        }
        puts (s2) ;
}
```

⑤ string compare

```
# include < stdio.h >
void main ()
{
        char s1[] = "small";
        char s2[] = "city";
        for(i=0 ; s1[i] == s2[i] == '\0' ; i++);
        if ( s1[i] > s2[i])
        {   printf (" s1 is less than s2 \n");
        }
```

```
    else if (s1[i] > s2[i])
    {
        printf (" s2 is less then s1 ");
    }
    else
    {
        printf (" s1 is equal to s2 \n");
    }
}
```

output

s2 is less then s1 .

Q31] Write a C program to
(i) comparing string
(ii) concatenate two strings [18CS32 Dec/Jan 19/20]

(i) comparing string

```
#include <stdio.h>
#include <string.h>
int main()
{
        char str1[20];
        char str2[20];
        int result;
        printf("Enter the first string:\n");
        scanf("%s", str1);
        printf("Enter the second string:\n");
        scanf("%s", str2);
        result = strcmp(str1, str2);
        if (result == 0)
        {
                printf("strings are same\n");
        }
        else
        {
                printf("strings are not same\n");
        }
        return 0;
}
```

output:

Enter the first string : GOOD
Enter the second string : NATURE
strings are not same.

Enter the first string : GOOD
Enter the second string : GOOD
string are same.

program to implement string concatinate

```
#include <stdio.h>
#include <string.h>
void main()
{
    char s1[30], s2[30];

    printf ("enter the string 1: \n");
    scanf ("%s", s1);
    printf ("enter the string 2: \n");
    scanf ("%s", s2);
    strcat (s1, s2);
    puts (s1);
}
```

output

```
Good
Enter the string 1: Good
Enter the string 2: Nature
Good Nature
```

32] Write a C program using pointer to

(1) concatenate of two strings

```
#include <stdio.h>
#define Max 100

int main()
{
        char str1[Max], str2[Max];
        char *s1 = str1;
        char *s2 = str2;
        printf("Enter 1st string : \n");
        gets(str1);
        printf("Enter 2nd string : \n");
        gets(str2);
        while(*(++s1));
        while(*(s1++) = *(s2++));
        printf("concatenated strings : %s", str1);
        return 0;
}
```

output

Enter 1st string : study
Enter 2nd string : hard
Concatenated string is : studyhard.

(11) reverse of a string

```
#include <stdio.h>

int string_length(char *);
void reverse(char *);

void main()
{
    char string[100];
    printf("enter a string\n");
    scanf(string);

    reverse(string);
    printf("reverse of entered string is %s", string);
}

void reverse(char *string)
{
    int length, c;
    char *begin, *end, temp;

    length = string_length(string);

    begin = string;
    end = string;

    for(c=0; c<(length-1); c++)
        for(c=0; c<length/2; c++)
        {
```

```
        temp = *end;
        *end = *beging;
        *beging = temp;
        beging++;
        end--;
      }
    }
    int string_length (char *pointer)
    {
        int c=0;
        while (*(pointer+c) != '\0')
            c++;
        return c;
    }
}
```

output

Enter a string : she is nice
Reversed of entered string is ecin si ehs

Enter a string : GMIT
Reversed of entered string is TIMG

33) Explain any methods of storing strings

Using character pointer we can store strings in two ways.

(1) Read only string in a shared segment. When a string value is directly assigned to a pointer, in most of compilers, it stored in data segment that is shared among function.

```
char * str = "GFG";
```

In the above line "GFG" is stored in shared read-only location, but pointer str is stored in read-write memory. So this kind of string should only be used when we don't want to modify string at later stage.

2) Dynamically allocated in heap segment.

Strings are stored like other dynamically allocated things is b.

```
char* str;
int size = 4;
str = (char *)malloc(sizeof(char)* size);
*(str+0) = 'G';
*(str+1) = 'F';
*(str+2) = 'G';
*(str+3) = '\0';
```

Example 1 (Try to modify strings)

Below program may crash because the line *(str+1) = 'n' tries to read only memory.

```
int main()
{
  char *str;
  str = "GFG";
  *(str+1) = 'n';
  getchar();
  return 0;
}
```

Below program works perfectly fine as str[] is stored in writable stack

```
int main()
{
    char str[] = "GFG";
    *(str+1) = 'n';
    getchar();
    return 0;
}
```

Below program also works perfectly fine as data at str is stored in heap.

```
int main()
{
    int size = 4;
    *(str+0) = 'G';
    *(str+1) = 'F';
    *(str+2) = 'G';
    *(str+3) = '\0'
                'n'
  } getchar();
```