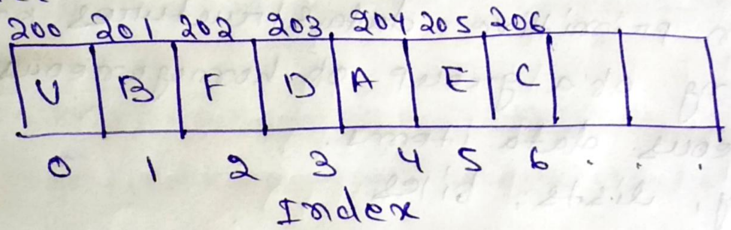
3rd semester CSE
Data structure

1(a) computer program, detailed plan or procedure for solving a problem with a computer; A program is a set of instructions that a computer follows in order to perform a particular task.

(b) An algorithm is an abbreviated statement in english (or any other language) that specifies the procedures that the program is to perform. It is a step by step method of solving a problem written in user's language.

(c) A data structure is a specialized format for organizing, processing, retrieving and storing data.
For example, we can store a list of items having the same data type using array data structure.



Array data structure.

(d) Data is defined as facts or figures, or information that is stored in or used by a computer. computer data may be processed by the computer's CPU and is stored in files and folders on the computer's hard disk.

2(a) Types of Data structure :-

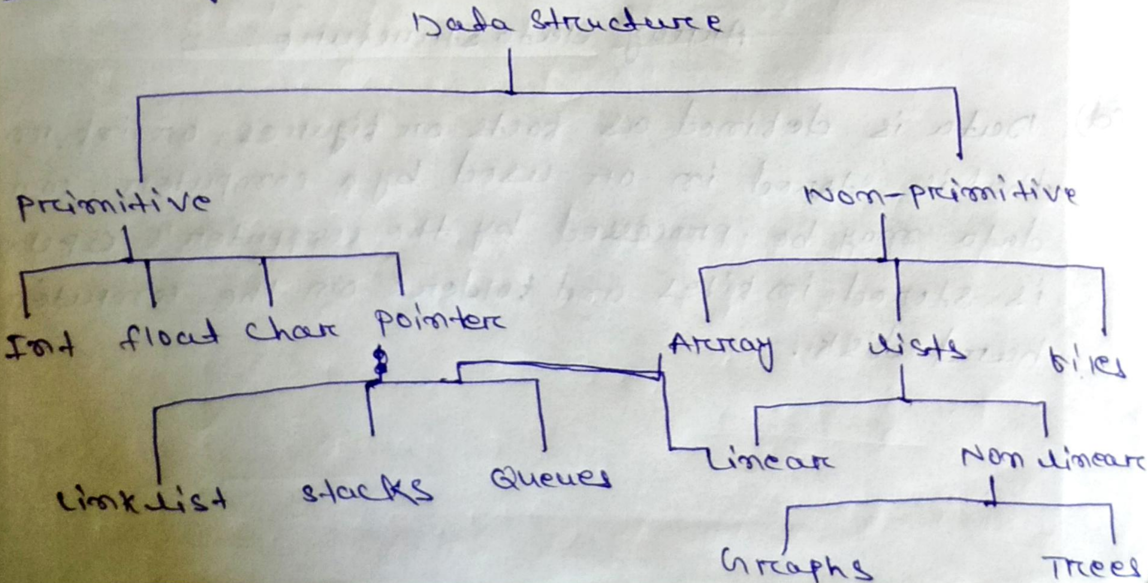
Data structure are normally divided into two broad categories
(1) primitive data structure
(2) Non primitive data structure

primitive data structure :

* These are basic structures and are directly operated upon by the machine instructions.
* These are having different representation in diff. computers.
Ex Integer takes 2 bytes in memory in a 16 bit compiler where as in 32-bit compiler it takes 32-bit.
→ Example of primitive data structures are Integer, float, Char, pointers.

Non-primitive data structure :

* These are derived from primitive datastructure
* The non primitive data structures emphasize on structuring of a group of homogeneous or heterogeneous data items.
Ex Array, lists, files



2(b) Space complexity:

The space complexity of an algorithm is the amount of memory it needs to run to completion.

Ex

```
1. Algorithm abc (a,b,c)
2. {
3. return a+b+b*c+(a+b-c)/(a+b)+4.0;
4. }
```

This problem consists of 3 variables a,b,c so fixed part is 3 and variable part is 0.

so space complexity is 3+0 = 3;

→ space complexity consists of two parts.

(1) Fixed part

(2) variable part

$\boxed{S(P) = C+SP}$ where S(P) = space complexity
C = Fixed part
SP = variable part

Time complexity:-

The time complexity of an algorithm is the amount of computer time it needs to run to completion.

The time T(P) taken by a program P is the sum of compile time and execution time.

Ex

Statements	Total steps
1. Algorithm sum (a,n)	0
2. {	0
3. s = 0	1
4. for i = 1 to n do	n+1
5. s = s + a[i]	n
6. return s	1
7. }	0
	2n+3

So time complexity is 2n+3

2(ɤ) one dimensional array is an array that has only one subscript specification that is needed to specify a particular element of an array.

Syntax

datatype arrayname [array-size]

Initialization of one dimensional array in c

An array can be initialized at either

(1) At compile time
(2) Dynamic Initialization

Compile time initialization

Syntax

datatype arrayname [array-size] = (list of elements of an array);

Example: int n[5] = {0, 1, 2, 3, 4}

Program

```
# include <stdio.h>
int main ()
{
int n[5] = {0, 1, 2, 3, 4};
printf ("%d", n[0]);
printf ("%d", n[1]);
printf ("%d", n[2]);
printf ("%d", n[3]);
printf ("%d", n[4]);
```

Output

```
0, 1, 2, 3, 4
```

Run time initialization:

```
# include <stdio.h>
int main ()
  {
   int a[5], i;
  for(i=0; i<5; i++);
   {
   Scanf ("%d", &a[i]);
 }
printf (" display");
 {
for (i=0; i<5; i++)
{
                              printf ("%d", a[i]);
                              }
                              }
                              }
```

Output

```
99
display 99
```

3(a) The two-dimensional array can be defined as an array of arrays. The 2D array is organized as matrices which can be presented as the collection of rows and columns. 2D arrays are created to implement a relational database lookalike data structure.

Syntax to declare the 2D array

datatype arrayname [rows] [columns];

ex int twodimen[4] [3];

Here 4 is the number of rows and 3 is the number of columns.

Two dimensional array examples in C

```
 # include <stdio.h>
  int main () {
 int i = 0; j=0;
 int arr [4][3] = {{1,2,3},{2,3,4},{3,4,5},{4,5,6}};
 traversing 2D array
  for (i=0; i<4; i++) {
  for(j=0; j<3; j++) {
 printf ("[%d][%d] = %d\n", i, j, arr[i][j]);
  } // end of j
  } // end of i
  return 0;
 }
```

Storing elements in a matrix

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

```
int arr[3][3], i, j;
for(i=0; i<3; i++)
 {
 for(j=0; j<3; j++)
 {
printf ("Enter a[%d][%d] ", i, j);
scanf ("%d", & arr[i][j]);
}
}
printf ("\n printing the element....\n");
for(i=0; i<3; i++)
 {
 printf ("\n");
 for(j=0; j<3; j++)
 {
printf ("%d\t", arr[i][j]);
}
}

}
```

O/P

```
Enter a[0][0] — 56
Enter a[0][1] —— 10
      a[0][2] — 30
      a[1][0] ——— 34
      a[1][1] ——— 21
      a[1][2] ———— 34

Enter a[2][0] : 45
      a[2][1] : 56
      a[2][2] : 78

Printing the elements . . .
```

$$\begin{bmatrix} 56 & 10 & 30 \\ 34 & 21 & 34 \\ 45 & 56 & 78 \end{bmatrix}$$

3(b)

Array	Stack
* A data structure consisting of a collection of elements each identified by the array index.	* An abstract data type that serves as a collection of elements with two principal operations: push and pop.
* contains elements of the same data type.	* contains elements of different data types.
* Basic operations include insert, delete, modify, traverse, sort and search and merge	* Basic operations are push, pop and peep
* Any element can be accessed using the array index.	* Only the topmost element can be read or removed at a time.
* It is used when we know all the data to be processed and require constant changes at any element.	* It is good to use when there are dynamic processes. It is useful when we do not know how much data would be required.