TITLE | PAGE NO. | TEACHER'S SIGN

Name ⇒ Shubham
Notes ⇒ DataStructure

Chapter-1 Data structure: An overview

Data :- Data is a collection of unorganized facts, concepts or instructions in a formalized manner suitable for interpretation, processing by the Computer system.

Information :- Information is defined as the processed data that will help to make the further decisions. Information will be generated after arranging the data into a suitable and arranging the data into a suitable and meaningful form.

E.g. A list of names and addresses
The contents of a letter

Data → Process → Information

Knowledge :- Knowledge is defined by the Oxford English Dictionary as (i) expertise, and skills acquired by a person through experience or education; the theoretical or practical understanding of a subject, (ii) what is known in a particular field or in total; facts and information or (iii) awareness or familarity gained by experience of a fact or situation.

Data Structure :- A data structure is a particular way of storing and organising data in a computer so that it can be used efficiently.

Features :-

- Data structure should be simple so that various operations can be performed on data effectively.

- Data structure represents the relationship of data in real world.

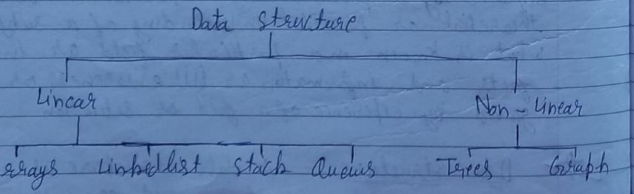
No. of Fields and Applications of Data structure:
- Graphics
- Compiler design
- Artificial Intelligence
- Simulation
- Numerical Analysis
- operating System
- Database Mangement system

Categories or classification of Data structures:

1) Linear = Sequence
   Non-Linear = Non-Sequence

2) Static = fix
   Dynamic = changeable

3) Homogenous = Same Type
   Heterogenous = Different Type



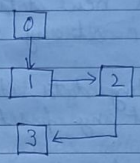
Linear Data structures :- A data structure is said to be linear if its elements are arranged in a sequence so that processing of elements is possible in a linear sequence.

Its types are:-

1) Arrays

a[5] = [0|1|2|3|4] Contigeous Memory Required

2) Linked list



Contigeous Memory Not Required

3) Stacks

LIFO ⇒ Last in, first out

4) Queues

[5|4|3|2|1| |] FIFO = First in, First out

Chapter - 1 Data Structure

Common operations on Data structures:

1) Traversing :-> Accessing each data element in the data structure exactly one, from first element to last element

2) Insertion :-> Addition of a new data item in the data structure at any position

3) Deletion :-> Removing existing data item in the data structure at any position.

4) Merging :-> Combining the elements of two similar structures into a single structure

5) Searching :- Finding the location of an element from a given set of elements

6) Sorting :- Arranging the elements in some specified order. The specified order may be ascending or descending in case of numeric values and alphabetically in case of characters

Chapter - 3 Arrays

Arrays:- Arrays are the simplest data structure. An array is a sequential list of elements which is fixed in number or we can say an array is a fixed-sized, homogenous and widely-used data structure.

One Dimensional Array:- If single subscript is required to reference an element of an array then it is known as one dimensional array. One dimensional array is a homogenous collection of finite data elements which are referenced by an index number one dimensional arrays are also known as linear array.

Declaring one dimensional array:-

data-type array-name[array size];

To find locations of array elements:-

LOC(A[J]) = base of A + W(J - lower bound)

E.g.1

Given: Base Address = 18400, J = 3, lower bound = 0

LOC A[3] = 18400 + 3(3 - 0)
= 18400 + 6
= 18406

E.g. 2

Given: Base Address = 2000, J = 5

LOC[5] = 2000 + 2(5-0)
= 2000 + 10
= 2010

E.g. 3

Given: Base Address = 3400, J = 2

LOC A[2] = 3400 + 2(2-0)
= 3400 + 4
= 3404

Inserting Element in one Dimensional Array.

Inserting element in one dimensional array is the operation of adding on element to existing set of elements. After insertion the size of one dimensional array is increased by a factor of one. Insertion in one dimensional array is possible only if the allocated memory space is large enough to accomodate the additional element. We have two cases:

- Inserting element in Unsorted array
- Inserting element in Sorted array.

1. Inserting element in Unsorted array:
   - Case 1: Inserting element at the end
   - Case 2: Inserting element in the beginning or in the middle

Algorithm: INSERT(A, N, J, ITEM)

// Let A is an array with N elements.
J is the location where ITEM has been inserted.
ITEM is the new value to be inserted.

Step 1: Set I = N (initialize counter)

Step 2: Repeat steps 3 and 4 while I >= J

Step 3: [Move the Ith element do rightwards]
Set A[I+1] = A[I]

Step 4: [Decrement the counter by 1]
Set I = I - 1
[End of STEP 2 loop]

Step 5: [Insert Element]
Set A[J] = ITEM

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

Step 7: Exit

Program

```
// Program to insert an element in Unsorted array:-
#include <stdio.h>
#include <conio.h>
int main() {
    int i, j, n, item;
    int a[20];
    printf("Enter the size of array");
    scanf("%d", &n);
    printf("\n Enter all the elements of array:-\n");
    for (i=1; i<=n; i++)
    {
        scanf("\n%d", &a[i]);
    }
```

```
printf("Enter the location of new element");
scanf("\n%d", &j);
printf("\n Enter the element to insert");
scanf("\n%d", &item);

    i = n;
    while (i >= j)
    {
      a[i+1] = a[i];
       i--;
    }
     a[j] = item;
     n = n+1;
     printf("\n After inserting new element, all elements
       are:-\n");
     for(i=1; i<=n; i++)
      {
        printf("\n%d", a[i]);
      }
       getch();
    }
```

2) Inserting element in Sorted Array

ALGORITHM: INSERT(A, N, ITEM)
// Given 'A' is an array whose elements are sorted in ascending order.
N is the number of elements in array 'A'
ITEM is the new element to be inserted in 'A'

Step 1 I = N

Step 2 Repeat step 3 and 4 while (ITEM < A[I]) and I >= 1

Step 3 A[I+1] = A[I]

Step 4 I = I - 1
[End of while loop]

Step 5 A[I+1] = ITEM

Step 6 N = N+1

Step 7 Exit

Deleting Element from One Dimensional Array
ALGORITHM: DELETE(A, N, J)
// Given 'A' is an array
N is the number of elements in array A
J is the position from where ITEM to be deleted

Step 1 [Initialize Counter]
Set I = J

Step 2 Repeat while I <= N

Step 3 [Move the elements to left upwards]
A[I] = A[I+1]

Step 4 [Increase the Counter]
Set I = I+1
[End of Step 2 loop]

Step 5 [Reset N]
Set N = N-1

Step 6 Exit

Program

```
// Program to insert and delete multiple elements in an array:-
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
Void is();
Void iu();
Void del();
int main(){
    print:
        system("cls");
        char ch;
        for(int i=0; i<60; i++)
            printf("--");
        printf("\n\t\t\t\t\t\t\t\t T.E.S.T.\n");
        for(int i=0; i<60; i++)
            printf("--");
        printf("\n Enter\n i to insert in sorted array
            \nu to insert in unsorted array
            \nd to delete\n");
        scanf("\n%c", &ch);
        switch(ch)
        {
            case 'i': is();
                      break;
            case 'u': iu();
                      break;
            case 'd': del();
                      break;
            default:
                printf("Choose correct option for operation")
```

Multidimensional Array → Two Dimensional Array

- Row Major Representation
  LOC[R[I][K]] = Base address of R + w[N(I-1)+(K-1)]

- Column Major Representation
  LOC[c[I][K]] = Base address of A + w[M(K-1)+(I-1)]

ALGORITHM: TRAVERSE (A[R][I])

// Here A is 2-Dimensional Array

Step 1 Repeat for I = 1 to R
Step 2 Repeat for J = 1 to C
Step 3 Apply PROCESS to A[I][J]
[End of step 2 loop]
[End of Step 1 loop]

Step 4 Exit

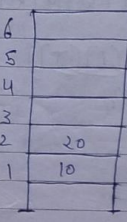
Addition of two Matrices

ALGORITHM(A[M][N], B[X][Y])
// Here A is a 2D Array with M rows and N Columns
// Here B is a 2D Array with X rows and Y Columns

Step 1 If (M ≠ X) and (N ≠ Y) then
Print Addition is not possible and EXIT
Step 2 Repeat for I = 1 to M
Step 3 Repeat for J = 1 to N
Step 4 C[I][J] = A[I][J] + B[I][J]
[End of Step 3 loop]
[End of step 2 loop]
Step 5 EXIT

Chapter 5 Stacks

Stack :- " A stack can be defined as a structure in which elements are added and removed from only one end or we can say it is a "last in first out" (LIFO) structure.



MAX = 6
PUSH = Insert
Pop = Delete
PEEK = To view top most element

PUSH

ALGORITHM : PUSH (STACK, TOP, MAX, ITEM)
// STACK is an array with MAX location. TOP is the pointer variable which shows the topmost element. ITEM is a variable which contains the element to be inserted

Step 1 [check for stack overflow]
if [TOP = = MAX]
Print stack is full and exit
[End of if structure]

Step 2 TOP = TOP + 1
[Increment TOP by 1]

Step 3 STACK [TOP] = ITEM

Step 4 EXIT

POP

ALGORITHM: POP (STACK, TOP, MAX, ITEM)

Step 1 [check for stack underflow]
if [TOP = = 0]
Print stack is empty and EXIT
[End of if structure]

Step 2 ITEM = STACK [TOP]

Step 3 TOP = TOP - 1
[Decrement TOP by 1]

Step 4 Exit

PEEK

ALGORITHM: PEEK(STACK, TOP)
// let TOP contains the location of topmost element of the stack

Step 1 [check for Underflow of stacks]
If (TOP == 0) then
Print Underflow and return [stack is already empty]
Exit
[End of If structure]

Step 2 STACK[TOP] ← ITEM
Step 3 Return (ITEM)
Step 4 Exit

Applications of stacks

- Evaluation of Arithmetic expressions
- Recursion
- Reversing a string
- Converting one number system to another
- Delimiter matching
- Quick sort for sorting data
- Adding very large numbers
- Backtracing

ALGORITHM: POSTFIX(P)

// Here P is a Postfix expression
Right Parenthesis ')'

Step 1 Add ")" to the end of expression P.
Step 2 Scan expression P from left to right until ")" is encountered
Step 3 If operand is encountered, PUSH it into stack
Step 4 If operator (↑) is encountered.
Step 5 Pop top two elements Such that B is the topmost element and A is second topmost element.
Step 6 Evaluate A ↑ B
Step 7 PUSH the result back to stack
[End of If step 4]
[End of step 2 loop]
Step 8 Pop the topmost element from stack and set to any Variable
Step 9 Exit

- Converting Infix Expressions into Postfix Expression

ALGORITHM: CONVERT(A, P)

Here A is an arithmetic expression written using infix notation. P is the equivalent postfix expression

Step 1 Push left "(" onto stack and
Step 2 Add ")" to the end of expression A
Step 3 Scan expression A from left to right and repeat step 4 for each element of A until the stack is empty
Step 4 Do following for the scanned element:
I If scanned element is an operand then add it to P
II If scanned element is a left "(" then push it onto stack,
III If scanned element is an operator then do following
(a) Repeat while topmost element of the stack is an operator with same precedence as scanned operator or higher precedence then scanned operator:
(i) Pop topmost element from the stack and add to P [End of step III(a) loop]
(b) Add newly scanned operator to stack
IV If scanned element is a right ")" then do the following:
(a) Repeat while left "(" appears on the top of stack
(i) Pop top element from stack and add to P
[End of step IV (a) loop]
(ii) Remove the left "("

[End of step 3 loop]

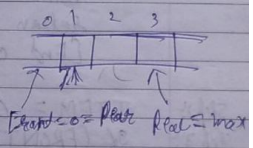
Step 5 Exit

QUEUES

ALGORITHM 1: INSERT_QUEUE (QUEUE, FRONT, REAR, ITEM, MAX)

// let QUEUE is one dimensional array representing queue containing MAX number of elements.
FRONT and REAR are the pointer variables that points to front and rear of queue respectively
ITEM is the new element to be inserted.
Max is the size of array

Step 1: [check for queue overflow]
If (REAR = MAX) Then
Print QUEUE is full
and Return
[End of If structure]



Step 2: If (FRONT = 0 and REAR = 0) then
a) Set FRONT = 1
b) Set REAR = 1
Else

Step 3: [Increment REAR]
REAR = REAR + 1
[End of If structure]

Step 4: [Insert new element]
QUEUE [REAR] = ITEM

Step 5: Exit

ALGORITHM: DELETE_QUEUE(QUEUE, FRONT, REAR, MAX)

// let QUEUE is a one dimensional array representing queue containing MAX number of elements.
FRONT and REAR are the pointer variable that points the front and rear of queue respectively
MAX is the size of an array.

Step 1:- [check for queue Underflow]
if (REAR = 0) then
Print "Underflow"
return
[End of if structure]

Step 2:- ITEM = QUEUE[FRONT]

Step 3:- if (FRONT = REAR)
[QUEUE Contains single element]
FRONT = 0
REAR = 0
[queue becomes empty]
Else if (FRONT < REAR) then
FRONT = FRONT + 1
[Increament FRONT by 1]
[End of If structure]

Step 4) Print the ITEM

Step 5) Exit

ALGORITHM :- INSERT_CQ(QUEUE, FRONT, REAR ITEM, N)

// let QUEUE is a circular queue with N locations
FRONT and REAR are the pointer variable that store the address of first and rear elements of QUEUE
ITEM is the value to be inserted

Step 1:- [check for overflow]
if (FRONT = 1 and REAR = N) or (FRONT = REAR + 1) then
Print "overflow"
return
[End of if structure]

Step 2 if (REAR = 0) then
Set FRONT = 1 and
REAR = 1
Else if (REAR = N) then
Set REAR = 1
Else
Set REAR = REAR + 1
[End of if structure]

Step 3) [Insert New Element]
Set QUEUE [REAR] = ITEM

Step 4) Exit

ALGORITHM : DELETE_CQ (QUEUE, FRONT, REAR, N)

// let QUEUE is an circular queue with N location. FRONT and REAR are the pointer variable that stores the adress of first and rear elements of QUEUE

Step 1> [check for Underflow]
If (REAR=0) then
print "Underflow"
return
[End of If structure]

Step 2> Set ITEM = QUEUE[FRONT]

Step 3> If (FRONT = REAR) then
Set FRONT = 0 and
REAR = 0
Else If (FRONT=N)
Set FRONT=1
Else
Set FRONT = FRONT+1
[End of If structure]

Step 4> Exit

TREE

1> ALGORITHM : PREORDER (ROOT)
// let ROOT is a pointer which Contains the address of root node of the binary tree.

Step 1 If (ROOT == NULL) then
Return
[End of If structure]

Step 2 PTR = ROOT (Root, L, R)

Step 3 Print PTR -> Info

Step 4 PREORDER (PTR -> LCHILD)

Step 5 PREORDER (PTR -> RCHILD)

Step 6 Exit

2> ALGORITHM : INORDER (ROOT)
// let Root is a pointer which Contains the address of the root node of the binary tree.

Step 1 If (ROOT == NULL) then
Return
[End of If structure] [left, Root, Right]

Step 2 PTR = ROOT

Step 3 INORDER (PTR -> LCHILD)
Step 4 Print PTR -> INFO
Step 5 INORDER (PTR -> RCHILD)
Step 6 Exit.

3) ALGORITHM : POSTORDER (ROOT)

// Let ROOT is a pointer which contains the address of the root node of the binary tree

Step 1: if (ROOT == NULL) then
Return
[End of if structure]

Step 2: PTR = ROOT (Left, Right, Root)

Step 3: POSTORDER (PTR -> LCHILD)

Step 4: POSTORDER (PTR -> RCHILD)

Step 5: Process PTR -> INFO

Step 6: Exit

4) ALGORITHM: BST_INS (ROOT, ITEM)

// Let ROOT is a pointer variable which contains the address of the root node and ITEM is the value, which we want to insert into binary search tree.

Step 1: if (ROOT == NULL) then
ROOT = New node
ROOT -> LCHILD = NULL
ROOT -> INFO = ITEM
ROOT -> RCHILD = NULL
Else

Step 2: Set PTR = ROOT

Step 3: if (ITEM < PTR -> INFO) then
BST_INS (PTR -> LCHILD, ITEM)
Else

Step 4: BST_INS (PTR -> RCHILD, ITEM)
[End of if structure]
[End of Step 1 structure]

Step 5: Exit

LINKED LIST

1) ALGORITHM: TRAVERSES(START)
// Let START is a pointer variable which contains address of the first node.

Step 1: Set PTR = START
Step 2: Repeat while PTR ≠ NULL
Step 3: Apply process to PTR → INFO
Step 4: PTR = PTR → LINK
[End of step 2 loop]
Step 5: Exit

2) ALGORITHM: INSERT_FIRST (START, ITEM)
// Let START is a pointer variable which holds the address of the first node and ITEM is the value to be inserted
Step 1: if (START = = NULL) then
Set START = new node.
START → INFO = ITEM
START → LINK = NULL
Step 2: Else
Step 3: Set PTR = START
Step 4: START = New node [create new node]
Step 5: START → INFO = ITEM
Step 6: START → LINK = PTR
[End of If structure]
Step 7: Exit.

3) ALGORITHM: INSERT_LAST (START, ITEM)
// let START is a pointer variable which holds the address of the first node and ITEM is the value to be inserted.

Step 1: if (START = NULL) then
Set START = New node
START → INFO = ITEM
START → LINK = NULL
Step 2: Else
Step 3: set PTR = START
Step 4: Repeat while (PTR → LINK ≠ NULL)
PTR = PTR → LINK
[End of while loop]
Step 5: PTR → LINK = new node
Step 6: PTR = PTR → LINK
Step 7: PTR → INFO = ITEM
Step 8: PTR → LINK = NULL
[End of If structure]
Step 9: Exit

4) ALGORITHM : INSERT_INTO_SORTED(START, ITEM)
// let START is a pointer variable which contains the address of first node and item is the value to be inserted.

Step 1: if (START == NULL) then
a) set START = New node
b) START -> INFO = ITEM
c) START -> LINK = NULL

Step 2: Else

Step 3: if (ITEM < START -> INFO) then
a) PTR = START
b) START = New node
c) START -> INFO = ITEM
d) START -> LINK = PTR

Step 4: Else
Set FLAG = 1, TEMP = START PTR = START -> LINK

Step 5. Repeat step 6 to 8 while (PTR ≠ NULL and FLAG ≠ 1)

Step 6: if (ITEM < PTR -> INFO) then
a) TEMP -> LINK = New Node
TEMP = TEMP -> LINK
TEMP -> INFO = ITEM
TEMP -> LINK = PTR
FLAG = 0

Step 7: Else if (PTR -> LINK == NULL) then
PTR -> LINK = new node
PTR = PTR -> LINK
PTR -> INFO = ITEM
PTR -> LINK = NULL
FLAG = 0

Step 8: Else
a) TEMP = PTR
b) PTR = PTR -> LINK
[End of if structure]
[End of while loop]
[End of step 1 if structure]

Step 9 Exit

5) ALGORITHM : DELETE_FIRST(START)
// let START is a pointer variable which contains the address of first node

Step 1: if (START == NULL) then
Print linked list does not exist

Step 2: Else
a) Set PTR = START
b) START = START -> LINK
c) Delete PTR [End of if]

Step 3: Exit

ALGORITHM : DELETE_LAST (START)

// let START is a pointer variable which contain the address of first node

Step 1: If (START == NULL) then
Print linked list doesn't exist and return

Step 2: Set TEMP = START

Step 3: If (TEMP -> LINK == NULL) then
Set START = NULL

Step 4: Else

Step 5: Repeat while (TEMP -> LINK ≠ NULL)

Step 6: PREV = TEMP
TEMP = TEMP -> LINK
[End of If structure]

Step 7: PREV -> LINK = NULL
[End of If structure]

Step 8: Delete TEMP

Step 9: Print ITEM is deleted

Step 10: Exit

ALGORITHM : D_INSERT_FIRST (START, LAST, ITEM)

// let START is a pointer variable which contain the address of the first node

Step 1: If (START == NULL) then

Step 2: Set START = new node
LAST = START

Step 3: START -> INFO = ITEM

Step 4: START -> BACK = NULL

Step 5: START -> FORW = NULL

Step 6: Else

Step 7: Set PTR = START

Step 8: START = new node

Step 9: START -> BACK = NULL

Step 10: START -> INFO = ITEM

Step 11: START -> FORW = PTR

Step 12: PTR -> BACK = START

Step 13: Exit.

a) ALGORITHM : D_INSERT_LAST(ITEM, START)

// let START is a pointer variable which contains the address of the first node and ITEM is the value to be inserted.

Step1: if (START == NULL) then

Step2: START = new node
LAST = START

Step3: START -> INFO = ITEM

Step4: START -> BACK = NULL

Step5: START -> FORW = NULL

Step6: Else

Step7: set PTR = LAST
PTR -> FORW = new node
LAST = PTR -> FORW
~~LAST -> FORW = NULL~~
LAST -> INFO = ITEM
LAST -> FORW = NULL
[End of if structure]

Step8: Exit

a) ALGORITHM : D_DELETE_LAST (START, LAST)

// let START is a pointer variable which contains the address of the first node and LAST is the pointer which contain the address of last node

Step1: if (START == NULL) then
Print node doesn't exist

Step2: Else if (START == LAST) then
TEMP = START
START = LAST = NULL

Step3: Else
TEMP = LAST
LAST = LAST -> BACK
LAST -> FORW = NULL
[End of if]

Step4: Delete TEMP

Step5: Exit

(b) ALGORITHM: D_DELETE_FIRST(START,)

// let START is a pointer variable which contains the address of first node and LAST is a pointer which contains the address of last node.

Step 1: if (START == NULL) then
Print underflow and return

Step 2: Else if (START == LAST) then
TEMP = START
START = LAST = NULL

Step 3: Else
TEMP = START
START = START → LINK
START → BACK = NULL
[End of if]

Step 4: Delete TEMP

Step 5: Exit

Linear Search in array

ALGORITHM :- LSERCH (A, N, item)

Step 1 LOC = -1

Step 2 i = 1

Step 3 Repeat while $i \leq N$ and $A[i] \neq item$
$i = i+1$

Step 4:- if $A[i] = item$ then
loc = i

Step 5:- Return loc

Algorithm for Selection Sort

Step 1 For i = 1 to n-1

Step 2: Set min = arr[i]

Step 3: set position = i

Step 4: for j = i+1 to n-1 repeat:
if (min > arr[j])
set min = arr[j]
Set position = j
[end of if]
[end of loop]

Step 5: Swap a[i] with arr[position]
[End of loop]

Step 6: END

MCQ's
{ Arrays }

1) The Largest element of an array index is called its
a) upper bound b) lower bound
c) range d) All of these

2) The smallest array element of an array index is called its
a) Lower bound b) upper bound
c) range d) extraction

3) ~~The smallest element of an~~

3) Two dimensional array are also called
a) table arrays b) matrix arrays
c) both d) none of above

4) which operation can be performed on an array
a) Traversing b) Merging
c) Sorting d) All

5) Merging of two arrays means arranging an array in some logical order.
a) true b) False

6) Bubble sort method in a array is used for
a) Merging two arrays b) Sorting arrays
c) Inserting element in array d) none of above

7) which is not a searching method in array
a) Linear Search b) Average Search
c) Binary Search d) none of above

8) Another name of bubble sort algorith is
a) Quick sort b) Exchange sort
c) Selection sort d) Insertion sort

9) Memory used by array cannot be increased or decreased at runtime or ~~f~~ Compile time.
a) Yes b) No

10) The element of an array need not to be in sorted order in _____ search.
a) linear b) binary
c) Both c) Tree

11) which sort is applicable to both sorted and unsorted data
a) linear search b) Binary search
c) Both a and b d) none

12) _____ is very slow at the begining of array.
a) Deletion b) Insertion
c) Both a and b d) none

13) _____ is an example of linear data structure.
a) linked list b) Array
c) Both a and b d) Tree

14) which is the correct way of declaring array
a) int arr[10]; b) int arr;
c) arr{20}; d) arr int[10];

15) How can you initialize array in c.
a) int arr[2] = {10,20}; b) int arr(2) = {10,20};
c) int arr[2] = {10,20}; d) int arr(2) = (10,20);

16) which one of the following is the size of int arr[9] assuming that int is of 4 bytes
a) 9 b) 36
c) 35 d) none

17) which of the following highly use the concept of an arrays.
a) Binary Search tree b) Caching
c) space locality d) scheduling of process

18) which of the following is the process of inserting an element in the stack.
a) Insert b) Add
c) Push d) none

(a) Array consists the elements of
(a) Same datatype b) different data type
c) Both d) None

20) when does the array Index out of Bounds Exception occurs.
a) Compile time b) Runtime
c) not an error d) non an exception

STACK

1) The postfix form of the expression (A+B) * (C+D - E) * F/G is
a) AB+CD*E-FG/** b) AB+CD*E-F**G/
c) AB+CD*E-*F*G/ d) AB+CDE*-*F*G/

2) which of the following is linear data structure
a) STACK b) Tree
c) Both a, b d) none

3) which of the following D.S. serve principle of last on Eigst out.
a) STACK b) Queue
c) None d) Both

4) which one of the following is the process of inserting an element in the stack?
a) Insert b) Add
c) Push d) none

5) when the user tries to delete the element from the empty stack. then the Condition is said to be
a) Underflow b) overflow
c) Garbage Collection d) None

6) when the user tries to insert the element from the full stack then the Condition is said to be.
a) Underflow b) overflow
c) Garbage d) None

7) which of the following is not the application of the stack data structure
a) String reversal b) Backtracing
c) Recursion d) Asynchronous data transfer.

8) which data structure is mainly used for implementing the recursive algorithm
a) Queue b) stack
c) Binary tree d) linked list

9) which data structure is required to convert the infix to postfix.
a) Stack b) linked list
c) Binary tree d) Queue

10) which of the following is infix expression
a) A+B*C b) AB+*
c) +A*BC d) None

11) which of the following is the postfix form of A+B*C ?
a) A+(BC*) b) AB+*

(c) +AB*C (d) +A*BC

12) If the element '1', '2', '3', are added in a stack, so what would be the order of removal.
(a) 1234 b) 2154
(c) 4321 d) none

13) What is the outcome of prefix expression + - * 32 / 941?
(a) 12 b) 11
(c) 5 d) 4

14) The minimum number of stack required to implement a stack is
(a) 1 b) 3
(c) 2 d) 5

15) Which one of the following is linear data structure.
(a) Tree (b) STACK
(c) both a and b d) none

16) On which principle, stack based upon
(a) LIFO b) Linear
(c) FIFO d) Tree

17) Process of inserting an element in stack is called
a) Pop (b) Push
c) Peek d) None

18) Process of deleting an element in stack is called
(a) Pop b) Push
c) Peek d) None

19) Process of viewing element in stack is called
(a) Pop b) PUSH
(c) PEEK d) None

20) What is the value of the postfix expression 632*4+-*?
(a) 1 b) 40
(c) 74 (d) -18

6, 3, 2, 4
6, 6
6, 2
-12
4-8
2-6 = -3

QUEUE

1) Which of the following data structure works on the principle of "First In First out"?
a) Linked List b) stack
c) Heap d) Queue

2) What is another name for the circular queue among the following options:
a) Square buffer b) Rectangular buffer
c) Ring buffer d) None

3) If the element '1', '2', '3', '4' is added in queue, what would be order of removal
a) 1 2 3 4 b) 4 3 2 1
c) 3 2 4 1 d) None

4) A list of elements in which enqueue operation takes place from one end, and dequeue takes from one end is.
a) Binary tree b) Queue
c) stack d) Linked list

5) Which of the following principle does Queue use
a) LIFO b) Linear tree
c) FIFO d) ordered array

6) which are the type of Queue:
a) Linear Queue b) Circular Queue
c) Both a, b d) None

7) Which one of the following is the overflow condition if linear queue is implements using an array with a size MAX_SIZE
a) rear = front - 1 b) rear = front + 1
c) rear = MAX_SIZE - 1 d) rear = MAX_SIZE

8) which one of following that determines that need of circular queue:
a) wastage of memory b) Access the Queue using pointer
c) follow FIFO d) None

9) which one of the following is the correct way to increment the rear and end in a circular queue:
a) rear = rear + 1 b) (rear + 1) % max
c) (rear x max) + 1 d) None

10) How many queue are required to implement a stack
a) 3 b) 2
c) 1 d) 4

11) Which one of the following is not the application of Queue Data structure.
a) Load balancing b) Balancing of Symbols
c) Data is transferred asynchronously d) Resources shared b/w various systems

12) The necessary condition to be checked before deletion from the queue is
a) Overflow b) Underflow
c) Rear Value d) Front Value

13) The necessary condition to be checked before insertion in queue is.
a) overflow b) underflow
c) Rear Value d) Front Value

14) which statement is not true regarding the priority queue.
a) Process with diff. priority b) Easy to implement
c) Deletion is easier d) None

15) which is the linear data structure
a) Queue b) Tree
c) Both a and b d) None

16) Insertion of element in linear queue is possible with which end.
a) Front b) Rear
c) Both d) None

17) Deletion of element in linear queue is possible with which end.
a) Front b) Rear
c) Both d) None

18) Process of inserting an element at the end of queue is known as.
a) Dequeue b) Enqueue
c) Push d) Pop

19) which function is used to return the value of ~~get~~ element without dequeuing is
a) Peek b) Ispfed of
c) Id d) none of above

20) ~~Frequeue~~ Enqueue, Dequeue function are found in
a) STACK b) Queue
c) Tree d) Array

TREE

(1) What is the maximum number of children that a node can have in a binary tree
a) 3 b) 1
c) 4 d) 2

2) which one of the following techniques is not used in binary tree
a) Randomized traversal b) Inorder Traversal
c) Preorder traversal d) Postorder Traversal

3) which of the following is a Non-Linear DS
a) Tree b) Queue
c) STACK d) Array

4) The no. of edges from the root to the node is called ___ of the tree
a) Height b) Depth
c) length d) width

5) The number of edges from the node to the defest leaf is called ___ of the tree
a) Height b) Depth
c) length d) width

6) what is full binary tree?
a) Each node has exactly zero or two children
b) Each node has exactly two children.
c) All the leaves are at the same level
d) Each node has exactly one or two children.

7) which of the following is not an advantage of tree
a) Hierachical str. b) Faster Search
c) Router algorithms d) Undo/Redu

8) which type of traversal in Binary Search tree output the value in sorted order.
a) Pre-order b) Post order
c) In-order d) None

9) of a node having two children is to be deleted from binary tree, it is replaced by its
a) In-order predecessor b) In-order Successor
c) Pre-order predecessor d) None.

10) which of the follow statement is true
a) Every tree is a bipartite graph
b) A tree contains a cycle
c) A tree with n nodes contain
d) A tree is connected graph

11) A Complete binary tree with the property that the value at each node is at least as large as the values at its children is called.
a) binary Search tree b) Binary tree
c) Completely balanced tree d) Heap

12) A full binary tree with n nodes contains
a) n nodes b) log n
c) 2n-1 nodes d) 2n+1 nodes

13) A full binary tree with n leaves contains
a) n nodes b) log n
c) 2n-1 nodes d) 2n+1 nodes

14) A full binary tree with n non-leaf nodes contains.
a) log n nodes
b) n+1 nodes
c) 2n-1 nodes
d) 2n+1 node

15) A Complete binary of level 5 has how many nodes.
a) 15
b) 25
c) 63
d) 30

16) Traversing a binary tree first root and then left and right subtree called ___ traversal
a) Postorder
b) Preorder
c) Inorder
d) None

17) The no. of nodes in a Complete binary tree of level is
a) 15
b) 20
c) 63
d) -79

18) Traversing a binary tree first left then root and right is called
a) Preorder
b) Postorder
c) Inorder
d) None

19) Traversing a binary tree first left then right then node is called
a) Preorder
b) Postorder
c) Inorder
d) None

20) Number of possible binary tree with 3 nodes is
a) 12
b) 9
c) 14
d) 5

LINKED LIST

1) A data structure in which linear sequence is maintained by pointers is known as
a) Array
b) stack
c) linked list
d) Pointer based.

2) which is an example of linear data structure
a) Array
b) Linked list
c) Graph
d) both a and b

3) A linear Collection of data elements where the linear node is given by means of pointer is Called
a) linked list
b) Node list
c) Primitive list
d) None

4) A variant of the linked list in which one of the node Contains NULL pointer is:
a) Singly linked list
b) Doubly linked list
c) Circular linked list
d) None

5) In Circular linked list, insertion of Node requires modification of:
a) one pointer
b) Two pointer
c) 3 pointer
d) None

6) linked list are not suitable for the implementation of
a) Insertion Sort
b) Radix Sort
c) Polynomial manipulation
d) Binary Search

7) A linear collection of data elements where the linear node is given by means of pointer is
a) linked list b) node list
c) Primitive list d) None

8) Linked list are not suitable for the operation
a) Deletion b) Search
c) Insertion d) All

9) Which of these are the application of linked list?
a) To implement file system
b) for separate chaining in hash tables
c) To implement none-binary trees
d) All

10) Which of the following sorting algorithm can be used to sort a random L.L with minimum complexity
a) Insertion sort b) Quick sort
c) Heap sort d) Merge sort

11) What type of L.L is best to answer question like "what is the item at position n
a) Singly L.L b) Doubly L.L
c) Circular L.L d) Array implementation of L.L

12) Which type of link list stores the address of the header node in the next field of the last node?
a) Single L.L b) Circular L.L
c) Double L.L d) Hashed list

13) In a double linked list the number of pointer affected for an insertion operation ~~were to be~~ will be
a) 4 b) 0
c) 1 d) Depends

14) Which D.S. in polynomial addition is implemented using _____ D.S
a) Queue b) linked list
c) Trees d) Stack

15) Which of the following operation is perform more efficiently by double L.L. than by single L.L.
a) Deleting a node whose location is given

16) In linked list each node contain minimum of two field is data field to store the data rebed field is
a) Pointer to character b) Pointer to integer
c) Pointer to node d) None

17) A variant of L.L. in which last node of the list pointes to the first node the list is:
a) Singly L.L. b) Doubly L.L.
c) Circular L.L. d) multiple L.L.

18) In doubley L.L. traversal can be performed
a) forword direction b) reverse direction
c) Both direction d) None

(19) In L.L. implementation of L.L. dnow with both pointers.
Sol

(20) which of the following is not non-linear D.S.
(a) Tree
c) Both
b) Graph
d) Linked list

Data Structures.

1) which of the following is not a Data structure
a) tree
c) pointer
b) Graph
d) Array

2) which of the following is not a linear data structure.
a) Tree
c) Array
b) STACK
d) QUEUE

3) A data structure in which linear sequence is maintained by pointer is known as
a) Linked list
c) Tree
b) Array
d) Graph

4) on what principle on a STACK depends
a) LIFO
c) FIFO
b) LILO
d) None

5) Queue follows.
a) LIFO
c) FIFO
b) FILO
d) None

6) which of the following is not a linear data structure
a) Linked list
c) tree
b) Array
b) Stack

7) which operation we can apply if to add an element in queue
a) Enqueue
c) Dequeue
b) Add
d) Insert

9) which operation we can apply to delete an element in queue
a) Enqueue b) Add
c) Dequeue d) Insert

a) _____ is very slow at the bigging of stack
a) Insertion b) Deletion
c) Both d) none

b) which of the following is not a linear data structure.
a) Array b) Graph
c) Tree d) none

11) Array contists the elements of
a) Same datatype b) Diff datatype
c) Both d) none

12) Process of viewing element in STACK is called
a) Pop b) PUSH
c) PEEK d) NONE

13) Process of insertion on element in STACK is called
a) Pop b) PUSH
c) PEEK d) None

14) Process of deleting an element from STACK is called
a) Pop b) PUSH
c) PEEK d) None

15) which of the following is a non-linear data structure
a) STACK b) Graph
c) Tree d) Both b and c

16) the necessary condition to be checked before the deletion in queue.
a) overflow b) Underflow
c) Both d) none

17) The necessary condition to be checked befor the insertion in queue
a) overflow b) Underflow
c) Both d) none

18) which of the following is not the operation of STACK
a) PREORDER b) POSTORDER
c) INORDER d) PEEK

19) which of the following is an infix expression
a) A+B*C b) AB+*
c) +AB*C d) none

20) which of the following is an prefix expression
a) A+B*C b) AB+*
c) +AB*C d) +A*BC