UNIT-1-2020- INTRODUCTION

Explain Data, Information, Data Types
Define data structure & Explain different Operations.
Explain Abstract Data Types.
Discuss Algorithm & its Complexity.
Explain Time, Space Trade-off.

INTRODUCTION TO DATA STRUCTURE

Data is the basic entity or fact that is used in calculation or manipulation process. There are two types of data such as numerical and alphanumerical data. Integer and floating-point numbers are of numerical data type and strings are of alphanumeric data type. Data may be single or a set of values and it is to be organized in a particular fashion. This organization or structuring of data will have profound impact on the efficiency of the program.

DATA

Data is value or set of values which does not give any meaning. It is generally a raw fact.

For example:

1. 34
2. 13/05/2008
3. Chintan
4. 12,34,43,21

ENTITY

An entity is a thing or object in the real world that is distinguishable from all other objects.

- The entity has a set of properties or attributes and the values of some sets of these attributes may uniquely identify an entity.
- An entity set is a collection of entities.

Example:

Entity	Student			
Attributes	Roll No.	Name	DOB	% of marks scored
Values	123	Ram	01/01/1980	79%

All students of a particular class constitute an entity set.

INFORMATION

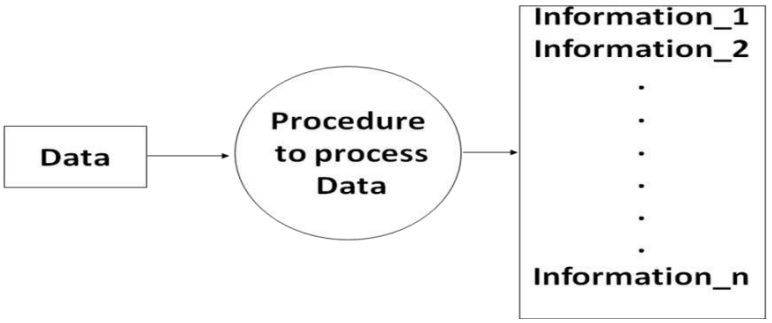
It can be defined as meaningful data or processed data. When the raw facts are processed, we get a related piece of information as its output/result.

Example:

Data (01/01/1980) becomes information if entity Ram is related to Date of birth attribute (01/01/1980) as follows:

DOB of student Ram is 01/01/1980.

Difference between Data & Information



DATA TYPE

A data type is a term which refers to the kind of data that may appear in computation. Every programming language has its own set of built-in data types.

Example:

Data	**Data type**
• 34	Numeric
• Chintech	String
• 21,43,56	Array of integers
• 12/05/2008	Date

In C, the following are the basic data types are: int, long, char, float, double, void etc.

Data structure is the structural representation of logical relationships between elements of data. In other words a data structure is a way of organizing data items by considering its relationship to each other.

Definition:

Data Structure is a specialized format for storing data so that the data's can be organized in an efficient way.

Or

"The way information is organized in the memory of a computer is called a data structure".

Or

A data structure is a mathematical or logical way of organizing data in the memory that consider not only the items stored but also the relationship to each other and also it is characterized by accessing functions.

It is clear from the above discussion that the data structure and the operations on organized data items can integrally solve the problem using a computer.

Data Structure = Organized data + Allowed Operations.

MAIN OBJECTIVES OF DATA STRUCTURE:

- To identify and create useful mathematical entities and operations to determine what classes of problems can be solved by using these entities and operations.
- To determine the representation of these abstract entities and to implement the abstract operations on these concrete representation.

NEED OF A DATA STRUCTURE:

- To understand the relationship of one data elements with the other and organize it within the memory.
- A data structures helps to analyze the data, store it and organize it in a logical or mathematical manner.
- Data structures allow us to achieve an important goal: component reuse.

- Once each data structure has been implemented once, it can be used over and over again in various applications.

APPLICATIONS OF DS:

A few applications of data structures are listed below:

- Compiler design
- Operating System
- Database Management system
- Network analysis

HOW TO CHOOSE THE SUITABLE DATA STRUCTURE:

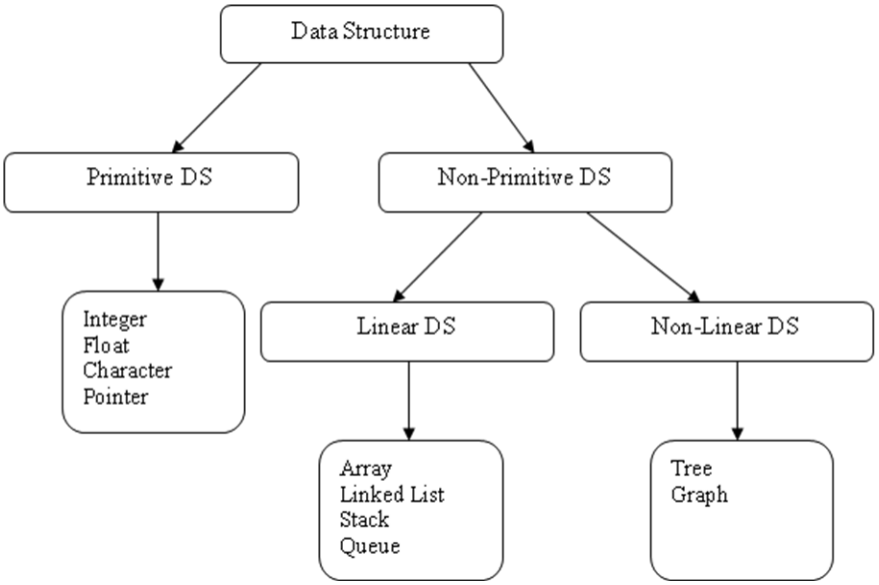
For each set of data there are different methods to organize these data in a particular data structure. To choose the suitable data structure, we must use the following criteria.

- Data size and the required memory.
- The dynamic nature of the data.
- The required time to obtain any data element from the data structure.
- The programming approach and the algorithm that will be used to manipulate these.

CLASSIFICATION OF DATA STRUCTURE

Based on how the data items are operated, it will classify into two broad categories.

- Primitive Data Structure
- Non-Primitive Data Structure



- **Primitive Data Structure:** These are basic DS and the data items are operated closest to the machine level instruction.
  Example: integer, characters, strings, pointers and double.

- **Non-Primitive Data Structure:** These are more sophisticated DS and are derived from primitive DS. Here data items are not operated closest to machine level instruction. It emphasize on structuring of a group of homogeneous (same type) or heterogeneous (different type) data items.

    - **Linear Data Structure:** In which the data items are stored in sequence order.
      Example: Arrays, Linked Lists, Stacks and Queues.

- **Non Linear Data Structure:** In Non-Linear data structures, the data items are not in sequence.
  Example: Trees, Graphs.

DATA STRUCTURE OPERATIONS:

The various operations that can be performed on different data structures are described below:

1. **Creating** – A data structure created from data.
2. **Traversal** – Processing each element in the list
3. **Search** – Finding the location of given element.
4. **Insertion** – Adding a new element to the list.
5. **Deletion** – Removing an element from the list.
6. **Sorting** – Arranging the records either in ascending or descending order.
7. **Merging** – Combining two lists into a single list.
8. **Modifying** – the values of DS can be modified by replacing old values with new ones.
9. **Copying** – records of one file can be copied to another file.
10. **Concatenating** – Records of a file are appended at the end of another file.
11. **Splitting** – Records of big file can be splitting into smaller files.

ABSTRACT DATA TYPES:

Abstract data type (**ADT**) is a specification of a set of data and the set of operations that can be performed on the data.

- It is a logical description of how we view the data and the operations that are allowed without knowing how they will be implemented.
- This means that we are concerned only with what the data is representing and not with how it will eventually be constructed.

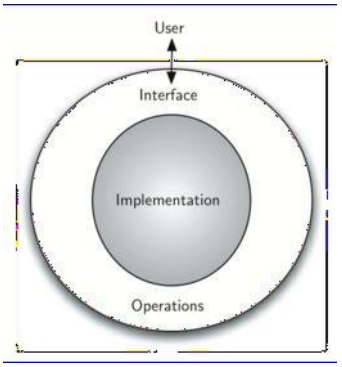


Fig shows a picture of what an abstract data type is and how it operates.

- The user interacts with the interface by using the operations.
- The user is not concerned with the details of the implementation.
- The implementation of an abstract data type, often referred to as a data structure
- An ADT is a data declaration packaged together with the operations that are meaningful on the data type.
  1. Declaration of Data
  2. Declaration of Operations

Examples: Objects such as lists, sets and graphs with their operations can be called as ADT. For the SET ADT, we might have operations such as *union, intersection, and complement* etc.

Uses of ADT:

1. It helps to efficiently develop well designed program
2. Facilitates the decomposition of the complex task of developing a software system into a number of simpler subtasks
3. Helps to reduce the number of things the programmers has to keep in mind at any time.
4. Breaking down a complex task into a number of earlier subtasks also simplifies testing and debugging.

ALGORITHM:

Definition: An *algorithm* is a finite set of instructions which, if followed, accomplish a particular task.

(OR)

An *algorithm* is a method of representing the step by step logical procedure for solving a problem. It is a tool for finding the logic of a problem.

In addition every algorithm must satisfy the following criteria:

- **Input:** There are zero or more quantities which are externally supplied. i.e. each algorithm must take zero, one or more quantities as input data.
- **Output:** At least one quantity is produced. i.e. produce one or more output values.
- **Definiteness:** Each instruction or step of an algorithm must be clear and unambiguous.
- **Finiteness:** An algorithm must terminate in a finite number of steps.
- **Effectiveness:** Each step must be effective, in the sense that it should be primitive (easily convertible to program).

COMPUTATIONAL COMPLEXITY OR ALGORITHM COMPLEXITY:

- After designing an algorithm, we have to be checking its correctness. This is done by analyzing the algorithm. The algorithm can be analyzed by tracing all step-by-step instructions, reading the algorithm for logical correctness, and testing it on some data using mathematical techniques to prove it correct.
- There may be more than one algorithm to solve a problem. The choice of a particular algorithm depends on its complexity.
    - Complexity refers to the rate at which the required storage or consumed time grows as a function of the problem size.
    - It is the performance evaluation or analysis / measurement of an algorithm is obtained by totaling the number of occurrences of each operation when running the algorithm.
    - Two important ways to characterize the effectiveness of an algorithm are its space complexity and time complexity.
        - Space Complexity
        - Time Complexity

SPACE COMPLEXITY:

- Analysis of space complexity of an algorithm or program is the amount of memory it needs to run to completion.
- We can measure the space by finding out that how much memory will be consumed by the instructions and by the variables used.

Example:

Suppose we want to add two integer numbers. To solve this problem we have following two algorithms:

Algorithm 1:
Step 1- Input A.
Step 2- Input B.
Step 3- Set C: = A+ B.
Step 4- Write: ‘Sum is ‘, C.
Step 5- Exit.

Algorithm 2:
Step 1- Input A.
Step 2- Input B.
Step 3- Write: ‘Sum is ‘, A+B.
Step 4- Exit.

TIME COMPLEXITY:

- It is the amount of time which is needed by the algorithm (program) to run to completion. We can measure the time by finding out the compilation time and run time.
- The compilation time is the time which is taken by the compiler to compile the program. It depends on the compiler and differs from compiler to compiler. One compiler can take less time than other compiler to compile the same program.
- This time is not under the control of programmer.
- So we ignore the compilation time and only consider the run time. The run time is the time which is taken to execute the program. We can measure the run time on the basis of number of instructions in the algorithm.

Example:
Suppose we want to add two integer numbers. To solve this problem we have following two algorithms:

Algorithm 1:
Step 1- Input A.
Step 2- Input B.
Step 3- Set C: = A+ B.
Step 4- Write: 'Sum is ', C.
Step 5- Exit.

Algorithm 2:
Step 1- Input A.
Step 2- Input B.
Step 3- Write: 'Sum is ', A+B.
Step 4- Exit.

- Both algorithms will produce the same result. But first takes 6 bytes and second takes 4 bytes (2 bytes for each integer). And first has more instructions than the second one. So we will choose the second one as it takes less space than the first one. (Space Complexity).
- Suppose 1 second is required to execute one instruction. So the first algorithm will take 4 seconds and the second algorithm will take 3 seconds for execution. So we will choose the second one as it will take less time. (Time Complexity).

TIME-SPACE TRADE-OFF:

- There may be more than one approach (or algorithm) to solve a problem.
- The best algorithm (or program) to solve a given problem is one that requires less space in memory and takes less time to complete its execution.
- But in practice, it is not always possible to achieve both of these objectives.
- One algorithm may require more space but less time to complete its execution while the other algorithm requires less time space but takes more time to complete its execution.
- Thus, we may have to sacrifice one at the cost of the other.
- If the space is our constraint, then we have to choose a program that requires less space at the cost of more execution time.
- On the other hand, if time is our constraint such as in real time system, we have to choose a program that takes less time to complete its execution at the cost of more space.

Time Complexity

➢ The time complexity of a program / algorithm is the amount of computer time that is need to run to completion.
➢ But the calculation of exact amount of time required by the algorithm is very difficult. So we can estimate the time and to estimate the time we use some asymptotic notation, which are described below. Let the no. of steps of an algorithm is measured by the function f(n).

1. Big 'O' notation

The given function f(n) can be expressed by big 'O' notation as follows. f(n) = O (g(n)) if and only if there exist the +ve const. 'C' and no such that $f(n) \leq C * g(n)$ for all $n \geq$ no.

2. Omega (Ω) notation

The function f(n) = Ω(g(n)) if and only if there exist the +ve const C and no such that $f(n) \geq C * g(n)$ for all $n \geq$ no

3. Theta (θ) notation:

The function f(n) = θ(g(n)) if and only if there exist 3 +ve const 'C1' , 'C2' and no such that $C1 * g(n) \leq f(n) \leq C2 * g(n)$

Big 'O' is the upper bound Ω is the lower bound θ is the avg bound which can be estimated in time complexity.

Space Complexity

The space complexity is the program that the amount of memory that is needed to run to completion. The space complexity need by a program have two different Components.

1. Fixed space requirement :- The components refers to space requirement that do not depend upon the number and sine of the programs input and output.
2. Variable space requirement :- This component consist of the space needed by structure variable whose size depends on the particular instruction I , of the program begin solved.

Time Complexity of Different types of Sorting and Searching

Bubble Sort

$T(n) = (n-1) + (n-2) \cdots\cdots + 2 + 1$

$= \frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2} = O(n^2)$

Insertion Sort

Worst Case $\frac{n(n-1)}{2} = O(n^2)$

Avg Case $\frac{n(n-1)}{4} = O(n^2)$

Selection Sort $\frac{n(n-1)}{2} = O(n^2)$

Merge Sort $n \log n = O(n \log n)$

Radix Sort $O(n^2)$

Linear Searching $O(n)$

Binary Searching $O(\log n)$

Quick Sort

Worst Case = $\theta(n^2)$

Best Case = $\theta(n \log n)$

UNIT-2

STRING

- A string in C is a array of character.
- It is a one dimensional array type of char.
- Every string is terminated by null character('\0') .
- The predefined functions gets() and puts() in C language to read and display string respectively.

Declaration of strings:

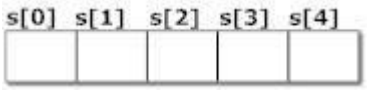
Strings are declared in C in similar manner as arrays. Only difference is that, strings are of char type.

Syntax:

char str_name[str_size];

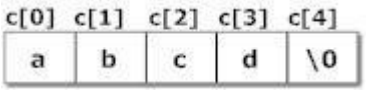
Example:

char s[5];



Initialization of strings

char c[]="abcd";



String handling functions

- Many library function are defined under header file <string.h> to perform different tasks.
- Diffrent user defined functios are:
  - **Strlen()**
  - **Strcpy()**
  - **Strcmp()**
  - **Strcat()**

Strlen():

- The strlen( ) function is used to calculate the length of the string.
- It means that it counts the total number of characters present in the string which includes alphabets, numbers, and all special characters including blank spaces.

Example:

```
char str[] = "Learn C Online";
int strLength;
strLength = strlen(str);    //strLength contains the length of the string i.e. 14
```

Strcpy()

- strcpy function copies a string from a source location to a destination location and provides a null character to terminate the string.

Syntax:

strcpy(Destination_String,Source_String);

Example:

```
char *Destination_String;
char *Source_String = "Learn C Online";
```

```
strcpy(Destination_String,Source_String);
printf("%s", Destination_String);
```

Output:

Learn C Online

Strcmp()

- Strcmp() in C programming language is used to compare two strings.
- If both the strings are equal then it gives the result as zero but if not then it gives the numeric difference between the first non matching characters in the strings.

Syntax:

```
int strcmp(string1, string2);
```

Example:

```
char *string1 = "Learn C Online";
char *string2 = "Learn C Online";
int ret;
ret=strcmp(string1, string2);
printf("%d",ret);
```

Output:

0

Strcat()

- The strcat() function is used for string concatenation in C programming language. It means it joins the two strings together

Syntax:

```
strncat(Destination_String, Source_String,no_of_characters);
```

Example:

```
char *Destination_String="Visit ";
char *Source_String = "Learn C Online is a great site";
strncat(Destination_String, Source_String,14);
puts( Destination_String);
```

Output:

Visit Learn C Online

Program to check wheaher a word is polyndrome or not

```
#include<stdio.h>
#include<string.h>
void main()
{
        char s1[10], s2[10];
        int x;
        gets(s1);
        strcpy(s2,s1);
        strrev(s1);
        x=strcmp(s1,s2);
        if(x==0)
            printf("pallindrome");
        else
        printf("not pallindrome");
        getch();}
```

UNIT-3
ARRAY

An array is a finite, ordered and collection of homogeneous data elements.

finite – It contain limited no. of element.

ordered – All the elements are stored one by one in a contiguous memory location.

homogeneous -All the elements of an array of same type.

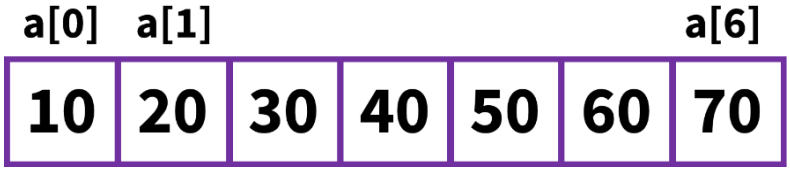
The elements of an array are accessed by means of an index or subscript. That's why array is called subscripted variable.

LINEAR ARRAY

If one subscript is required to refer all the elements in an array then this is called Linear array or one-dimensional array.

Representation of Linear Array in memory

Let a is the name of an integer array.
It contains a sequence of memory location.



Let b = address of the 1st element in the array i.e. base address
If w = word size
then address of any element in array can be obtained as
address of
(a[i]) = b + i x w i = index no.
address of
a[3] = b + i x w
= b + 3 x 2 = b + 6

Operation on Array

1. **TRAVERSAL**

   This operation is used to visit all the elements of the array.

```
Void traverse (int a[], int n)
{
   int i;
for(i=0; i<n; i++)
Printf("%d",a[i]);
}
```

2. **INSERTION**

   Inserting the array at the end position can be done easily, but to insert at the middle of the array we have to move the element to create a vacant position to insert the new element.

```
int insert (int a[], int n, int pos, int ele)
{
  int i;
for(i=n-1; i>=pos-1; i--)
  {
    a[i+1] = a[i];
  }
    a[pos-1] = ele;
 n++;
 return(n);
}
```

3. **DELETION**

```
int delete (int a[], int n, int pos)
{
for (i = pos-1; i<n-1; i++)
    {
       a[i] = a[i+1];
    }
n- - ;
    return(0);
}
```

- Deleting an element at the end of the array can be done easily by only decreasing the array size by 1.
- But deleting an element at the middle of the array require that each subsequent element from the position where to be deleted should be moved to fill up the array.

4. SEARCHING

Finding the location of a given item in a particular array is called as searching.

These are 2 types Searching

1. Linear Search
2. Binary Search

Linear Search

- Suppose 'a' is an array with n element.
- To find an item we have to search the array 'a' from the first element sequentially.
- This sequential search is known as Linear Search.

Algorithm:

```
void linearsearch (int a[], int n, int no)
{
int loc = -1, i ;
for (i=0; i<n; i++)
{
   if (a[i] == no)
   loc = i ;
}
  if (loc == -1)
  Printf ("Data not found");
  else
  Printf("Data found at %d position/location",loc);
}
```

Where a[] = name of the array

n = no. of element present in array 'a'

no = number to be searched

loc = location to be searched

Binary Search

- Sequential Search is a simple method for searching an element in array.
- But this is not efficient for large list because we will have to make n comparison to search for the last element in the list.
- The binary search technique is very much efficient and applied only in sorted array.
  Ex: Searching a name in telephone directory or searching a word in dictionary.
- In this method we make a comparison between the key and the middle element of the array.
- Since the array is sorted this comparison results either in a match between key and the middle element of the array or identifying the left half or result half of the array.
- When the current element is not equal to the middle element of the array, the procedure is repeated on the half in which the desired element is likely to be present.
- Proceeding in this way, either the element is detected or final division leads to a half consisting of no element.

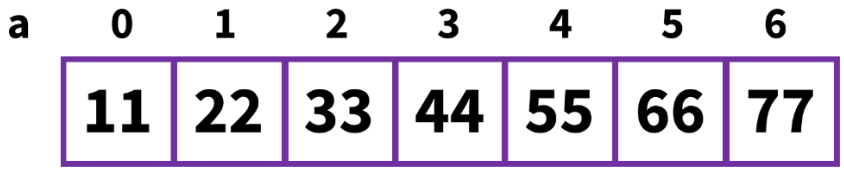
Algorithm:

```
int BinarySearch (int key, int a[], int n)
   {
    int low, hi, mid;
        low = 0 ;
        hi = n-1;
while  (low<=hi)
  {
     mid = (low+hi)/2;
     if (key = = a[mid])
        return(mid);
     if (key<a[mid])
        hi = mid-1;
     else
         low = mid+1;
}
   return(-1);
}
```

Ex:

Suppose the array elements are

11, 22, 33, 44, 55, 66, 77



We wish to search for 33

So key = 33

low = 0

hi = 6

n = 7

Is low <= hi Yes

then mid = $\frac{low+hi}{2} = \frac{0+6}{2} = 3$

Is 33 == a[3] No

Is 33 < a[3] Yes

the steps will be repeated for lower half

hi = mid-1

= 3-1 = 2

low = 0

Is low <= hi

mid = $\frac{0+2}{2} = 1$

If key (33) == a[1] No

If 33 < a[1] No

low = 1+1 =2

hi = 2

mid = $\frac{2+2}{2} = \frac{4}{2} = 2$

Is 33 == a[2]

Yes the search is successful at index 2.

Memory Representation of 2D Array

- The array having two subscript is called as 2D array.
- In 2D array the elements are stored in contiguous memory location as in single dimensional array.
- There are 2 ways of storing any matrix in memory.
  1. Row-major order
  2. Column-major order

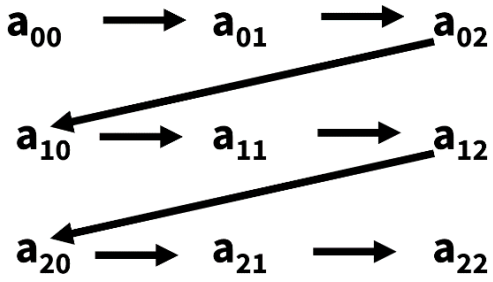
Row-major Order

In row-major order the row elements are focused first that means elements of matrix are stored on a row-by-row basis.

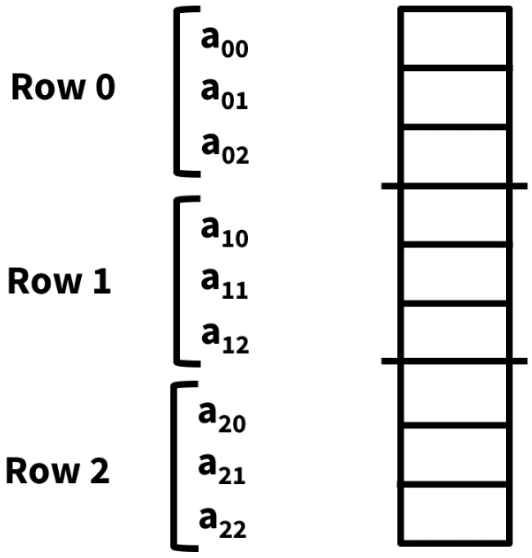
Logical Representation of array a[3][3]

a[0][0]	a[0][1]	a[0][2]
a[1][0]	a[1][1]	a[1][2]
a[2][0]	a[2][1]	a[2][2]

Row major order representation of a[3][3]



Storage Representation in Row-major order

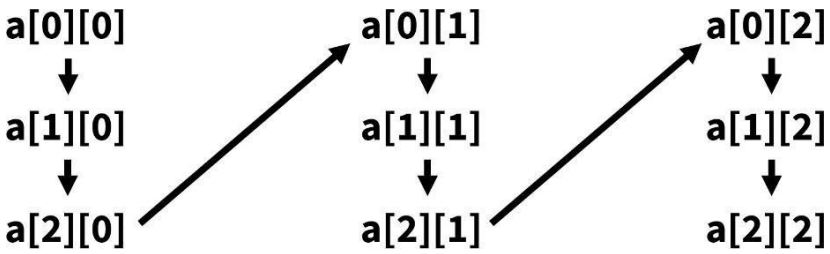


(Row-by-Row)

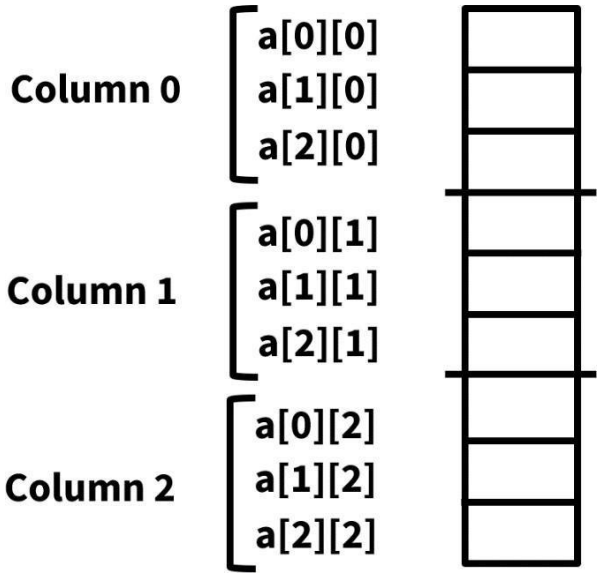
Column-major Order

In column major order the column elements are focused first that means elements of the matrix are stored in column-by-column basis.

Ex: Column major order representation of a[3][3]



Storage Representation of matrix in Column-major Order



Address Calculation of Matrix in Memory

Row-major order

Suppose a $[u_1]$ $[u_2]$ is a 2D array
u_1 = no. of row
u_2 = no. of column
Address of a[i][j] = $b+(i \times u_2 + j) \times w$

Column-major order

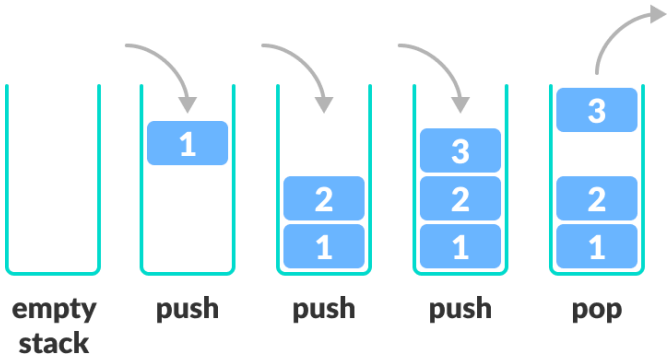
Address of a[i][j] = $b + (j \times u_1 + i) \times w$

UNIT-4

STACKS & QUEUES

STACK

- Stack is a linear data structure in which an element may be inserted or deleted at one end called TOP of the stack.
- That means the elements are removed from a stack in the reverse order of that in which they were inserted into the stack.



Stack Push and Pop Operations

- Stack is called LIFO (Last-in-first-Out) Str. i.e. the item just inserted is deleted first.
- There are 2 base operations associated with stack
    1. Push :- This operation is used to insert an element into stack.
    2. Pop :- This operation is used to delete an element from stack.

Condition also arise :

1. Overflow :- When a stack is full and we are attempting a push operation , overflow condition arises.
2. Underflow :- When a stack is empty , and we are attempting a pop operation then underflow condition arises.

Representation of Stack in Memory :

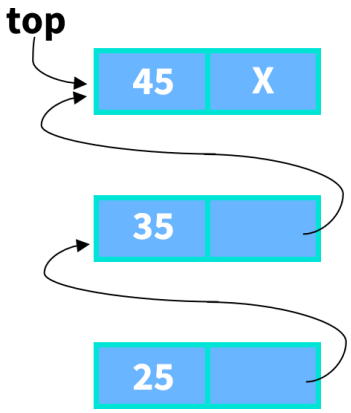
A stack may be represented in the memory in two ways:

1. Using one dimensional array i.e. Array representation of Stack.
2. Using single linked list i.e. Linked list representation of stack.

Array Representation of Stack :
To implement a stack in memory, we need a pointer variable called TOP that hold the index of the top element of the stack, a linear array to hold the elements of the stack and a variable MAXSTK which contain the size of the stack.

Linked List Representation of Stack :

- Array representation of Stack is very easy and convenient but it allows only to represent fixed sized stack.



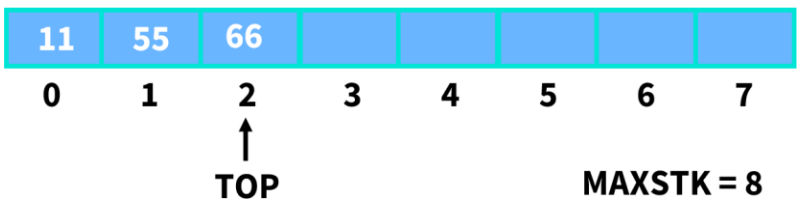
- But in several application size of the stack may very during program execution, at that cases we represent a stack using linked list.
- Single linked list is sufficient to represent any Stack.
- Here the 'info' field for the item and 'next' field is used to print the next item.

INSERTION IN STACK (PUSH OPERATION)

This operation is used to insert an element in stack at the TOP of the stack.

Algorithm :-

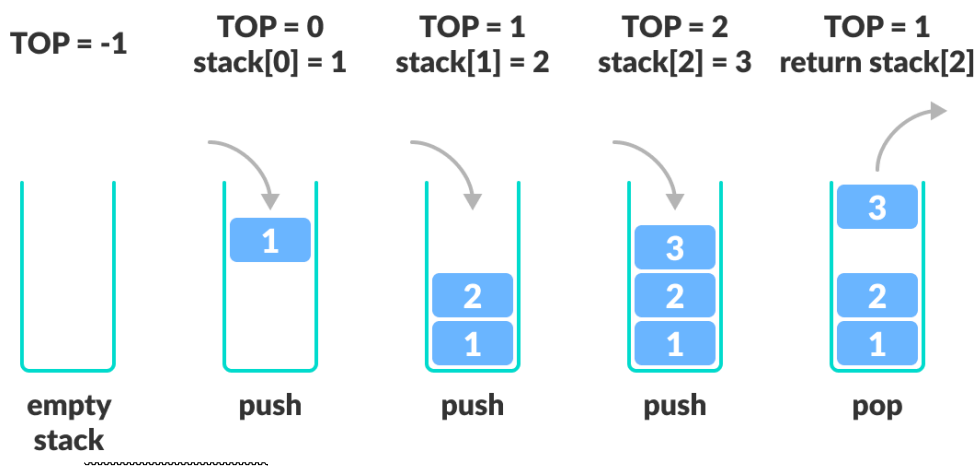
PUSH (STACK, TOP, MAXSTK, ITEM)
This procedure pushes an item into stack.
1. If TOP = MAXSTK then print Overflow and Return.
2. Set TOP = TOP+1 (Increase TOP by 1)



3. Set STACK[TOP] = ITEM (Inserts ITEM in new TOP position)
4. Return
(Where Stack = Stack is a list of linear structure.
TOP = printer variable which contain the location of TOP element of the stack.
MAXSTK = A variable which contain size of stack
ITEM = Item to be inserted)

DELETION IN STACK (POP OPERATION)

This operation is used to delete an element from stack.



Algorithm :-

POP(STACK, TOP, ITEM)
This procedure deletes the top element of STACK and assigns it to the variable ITEM.

1. If TOP = 0 the print Underflow and Return.
2. Set ITEM = STACK[TOP] (Assigns TOP element to ITEM)
3. Set TOP = TOP-1 (Decreases TOP by 1)

Working of Stack Data Structure

4. Return

ARITHMETIC EXPRESSION

There are 3 notation to represent an arithmetic expression.

1. Infix notation
2. Prefix notation
3. Postfix notation

INFIX NOTATION

The conventional way of writing an expression is called as infix.
Ex: A+B, C+D, E*F, G/M etc.

Here the notation is
<Operand><Operator><Operand>

This is called infix because the operators come in between the operands.

PREFIX NOTATION

This notation is also known as "POLISH NOTATION"

Here the notation is
<Operator><Operand><Operand>

Ex: +AB, -CD, *EF, /GH

POSTFIX NOTATION

This notation is called as postfix or suffix notation where operator is suffixed by operand.

Here the notation is
<Operand ><Operand><Operator>

Ex: AB+, CD-, EF*, GH/

This notation is also known as " REVERSE POLISH NOTATION."

CONVERSION FROM INFIX TO POSTFIX EXPRESSION

Algorithm:

POLISH(Q,P)

Q is an arithmetic expression written in infix notation. This algorithm finds the equivalent postfix expression P.

1. Push " (" onto stack and add ") " to the end of Q.
2. Scan Q from left to right and repeat steps 3 to 6 for each element of Q until the STACK is empty.
3. If an operand is encountered, add it top.
4. If a left parenthesis is encountered, push it onto stack.
5. If an operator X is encountered, then:
    a) Repeatedly POP from STACK and add to P each operator (on the top of the STACK) which has the same precedence as or higher precedence than the operator X .
    b) Add the operator X to STACK.
6. If a right parenthesis is encountered then:
    a) Repeatedly POP from STACK and add to P each operator (on the top of the STACK) until a left parenthesis is encountered.
    b) Remove the left parenthesis.
    [End of if str.]

    [End of Step-2 Loop]

7. Exit

Ex: A+(B+C – (D/E+F)*G)*H)

Symbol Scanned	STACK	EXPRESSION (POSTFIX)
	(	
A	(	A
+	(+	A
(	(+(	A
B	(+(	AB
*	(+(*	AB
C	(+(*	ABC
-	(+(-	ABC*
C	(+(-(	ABC*
D	(+(-(	ABC*D
/	(+(-(/	ABC*D
E	(+(-(/	ABC*DE
^	(+(-(/^	ABC*DE
F	(+(-(/^	ABC*DEF
)	(+(-	ABC*DEF^/
*	(+(-*	ABC*DEF^/
G	(+(-*	ABC*DEF^/G
)	(+	ABC*DEF^/G*-
*	(+*	ABC*DEF^/G*-
H	(+*	ABC*DEF^/G*-H
)		ABC*DEF^/G*-H*+

Equivalent Postfix Expression ->ABC*DEF^/G*-H*+

EVALUATION OF POSTFIX EXPRESSION

Algorithm:

This algorithm finds the value of an arithmetic expression P written in Postfix notation.

1. Add a right parenthesis ") " at the end of P.
2. Scan P from left to right and repeat steps 3 and 4 for each element of P until the ") " is encountered.
3. If an operand is encountered, put it on STACK.

4. If an operator X is encountered then:
    a) Remove the two top element of STACK, where A is top element and B is the next-to-top element.
    b) Evaluate B X A.
    c) Place the result of (b) on STACK.
    [End of if str.]
5. Set value equal to the top element on STACK.
6. Exit

Ex: 5, 6, 2, +, *, 12, 4, /, -)

Symbol Scanned	STACK
5	5
6	5,6
2	5,6,2
+	5,8
*****	40
12	40,12
4	40,12,4
/	40,3
-	37 (Result)
)	

APPLICATIONS OF STACK:

- Recursion process uses stack for its implementation.
- Quick sort uses stack for sorting the elements.
- Evaluation of antithetic expression can be sone by using STACK.
    - Conversion from infix to postfix expression
    - Evaluation of postfix expression
    - Conversion from infix to prefix expression
    - Evaluation of prefix expression.
- Backtracking
- Keeptrack of post-visited (history of a web- browsing)

IMPLEMENTATION OF RECURSION

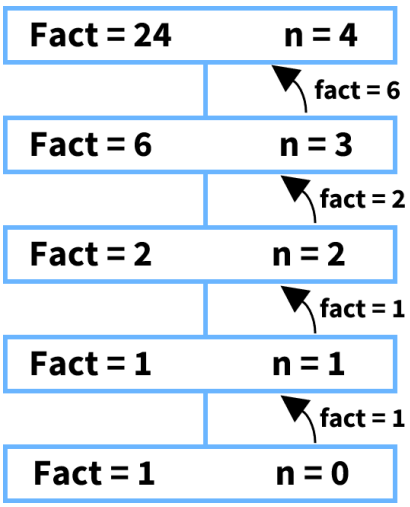
A function is said to be Recursive function if it call itself or it call a function such that the function call back the original function.

This concept is called as Recursion.

The Recursive function has two properties.

I. The function should have a base condition.
II. The function should come closer towards the base condition.

The following is one of the example of recursive function which is described below.



Factorial of a no. using Recursion

The factorial of a no. 'n' is the product of positive integers from 1 to n.

n ! = 1 X 2 X 3 X . . . . . .X (n-) X n
Physically proved n ! = n X (n-1) !

The factorial function can be defined as follows.

I. If n=0 then n ! = 1
II. If n>0 then n ! = n.(n-1) !

The factorial algorithm is given below factorial (fact , n)

This procedure calculates n! and returns the value in the variable fact.

1. If n=0 then fact=1 and Return.
2. Call factorial (fact, n-1)
3. Set fact = fact*n
4. Return

Ex: Calculate the factorial of 4.

4 ! = 4 x 3 !
3 ! = 3 x 2 !

2 ! = 2 x 1 !

1 ! = 1 x 0 !

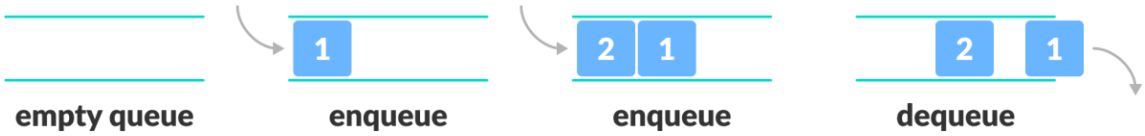
0 ! = 1

1 ! = 1 x 1 = 1

2 ! = 2 x 1 = 2

3 ! = 3 x 2 = 6

4 ! = 4 x 6 = 24



QUEUE

- Queue is a linear data structure or sequential data structure where insertion take place at one end and deletion take place at the other end.
- The insertion end of Queue is called rear end and deletion end is called front end.
- Queue is based on (FIFO) First in First Out Concept that means the node i.e. added first is processed first.
- HereEnqueue is Insert Operation.

FIFO Representation of Queue

Dequeue is nothing but Delete Operation.

Types of Queue:

1. General Queue

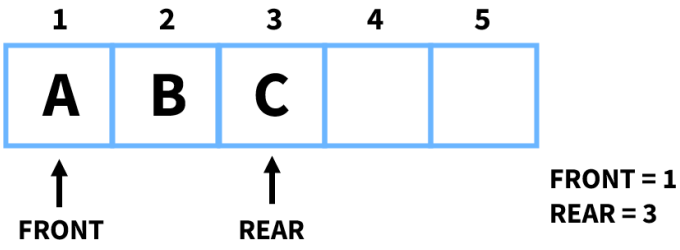
2. Circular Queue
3. Deque
4. Priority Queue

REPRESENTATION OF QUEUE IN MEMORY

- The Queue is represented by a Linear array "Q" and two pointer variable FRONT and REAR.
- FRONT gives the location of element to be deleted and REAR gives the location after which the element will be inserted.
- The deletion will be done by setting
  Front = Front + 1
- The insertion will be done by setting
  Rear = Rear + 1

INSERTION IN QUEUE(Enqueue)

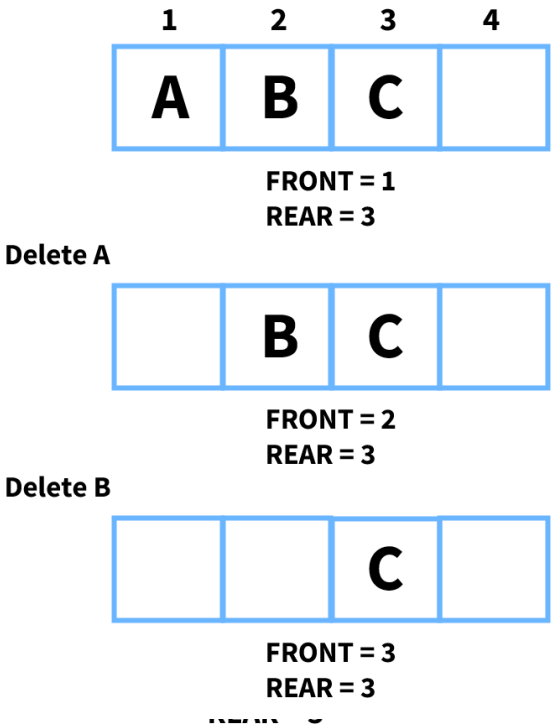
Ex:



Algorithm:

Insert (Q, ITEM, Front, Rear)
This procedure insert ITEM in queue Q.

1. If Front = 1 and Rear = Max



   Print 'Overflow' and Exit.
2. If front = NULL than
      Front = 1
      Rear = 1
   else
      Rear = Rear +1
3. Q [Rear] = ITEM
4. Exit

DELETION IN QUEUE(Dequeue)

Ex:

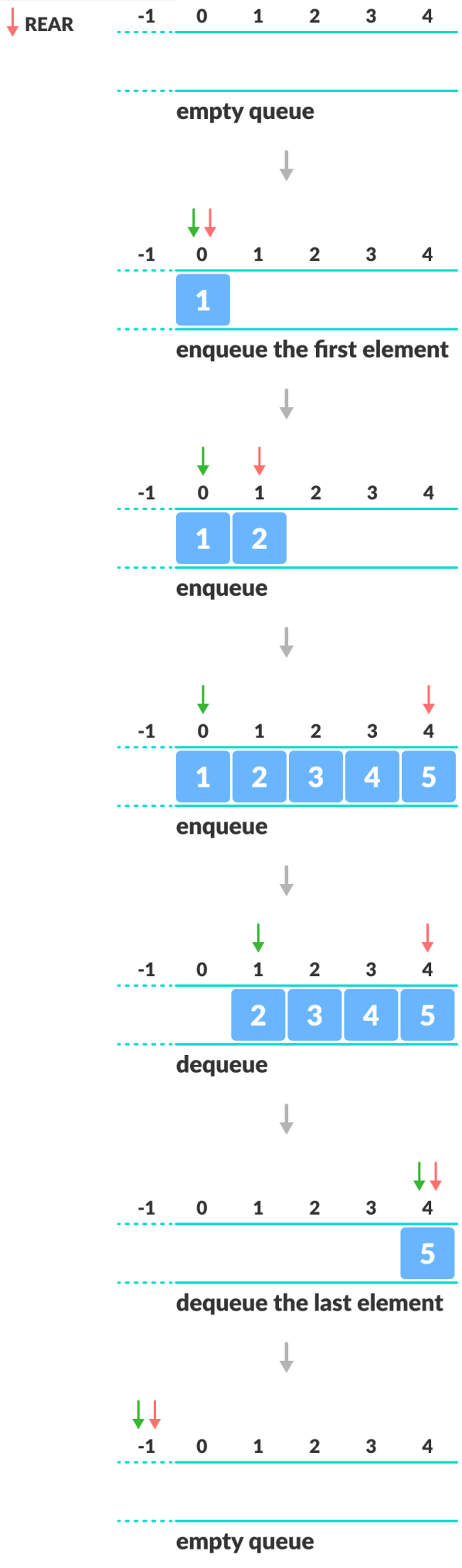
Algorithm:

Delete (Q, ITEM, FRONT, REAR)

This procedure remove element from queue Q.

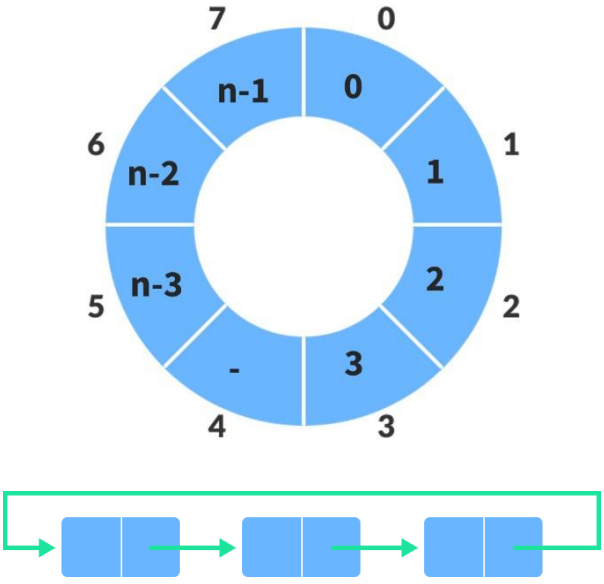
1. If Front = Rear = NULL
   Print 'Underflow' and Exit.
2. ITEM = Q (FRONT)
3. If Front = Rear
   Front = NULL
   Rear = NULL
   else
   Front = Front + 1
4. Exit

Enqueue and Dequeue Operations



CIRCULAR QUEUE

- Let we have a queue that contain 'n' elements in which Q[1] comes after Q[n].
- When this technic is used to construct a queue then the queue is called circular queue.
- In Circular queue when REAR is 'n' and any element is inserted then REAR will set as 1.
- Similarly when the front is n and any element is deleted then front will be 1.



Circular Queue representation

INSERTION ALGORITHM OF CIRCULAR QUEUE

Insert (Q, N, FRONT, REAR, ITEM)
This procedure inserts an element ITEM into the circular queue Q.

1. If Front = 1 and Rear = N
   then print 'Overflow' and Exit.
2. If front = NULL
   then Front = 1 and Rear = 1
   else if Rear = N then
   set Rear = 1
   else
   set Rear = Rear+1
   (End of if Str.)

3. Set Q [Rear] = ITEM (Insert a new element)
4. Exit

DELETION ALGORITHM OF CIRCULAR QUEUE

Delete (Q, N, ITEM, FRONT, REAR)
This procedure delete an element from a circular queue.

1. If Front = NULL then write Underflow and Return.
2. Set ITEM = Q [FRONT]
3. If Front = Rear
   Then Front = NULL and Rear = NULL
   Else if Front = N then
   Set Front = 1
   else
   Front = Front+1
   [End of if str.]
4. Return

PRIORITY QUEUE

- If is a type of queue in which each element has been assigned a priority such that the order in which the elements are processed according to the elements are processed according to the following rule
  I. This element of high priority is processed first.
  II. The element having same priority are processed according to the order in which they are inserted.
- The lower ranked no. enjoy high priority.

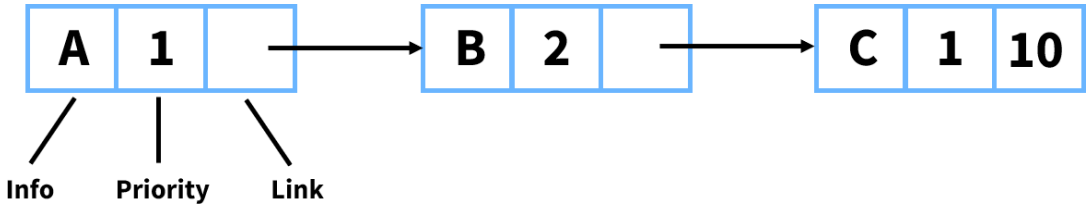
DELETION ALGORITHM OF CIRCULAR QUEUE

Priority Queue can be represented in memory in 2 ways.

I. One way list representation
II. Array representation

One way list representation

Each node in the list contain 3 fields.
i.e. INFO, PRIORITY, LINK



Array Representation

- Here the priority queue is represented by means of 2D array.
- Where row value represent the priority number and each row maintained as a circular queue.
- Each row has its FRONT and REAR value to represent the starting and ending of row.

	1	2	3	4	5
1		X	Y	Z	
2	A	D			
3	A		Q	R	S

FRONT	REAR
2	4
1	2
3	1

TYPES OF PRIORITY QUEUE

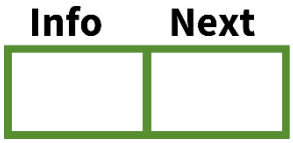
I. Ascending Priority Queue :-
It is the collection of items into which items can be inserted arbitrarily and from which only the smallest item can be removed.

II. Descending Priority Queue :-
This is similar to ascending priority Queue but it allows deletion of only the largest item.

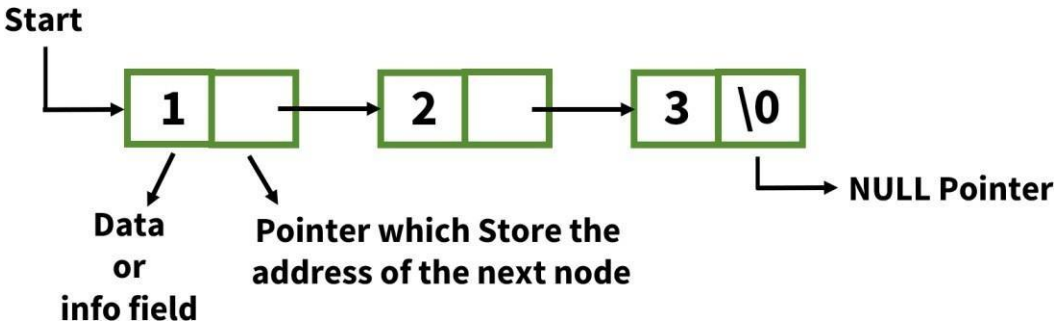
UNIT-5

LINKED LIST

- Linked List is a collection of data elements called as nodes.
- The node has 2 parts
  1. Info is the data part
  2. Next i.e. the address part that means it points to the next node.



- If linked list adjacency between the elements are maintained by means of links or pointers.
- A link or pointer actually the address of the next node.



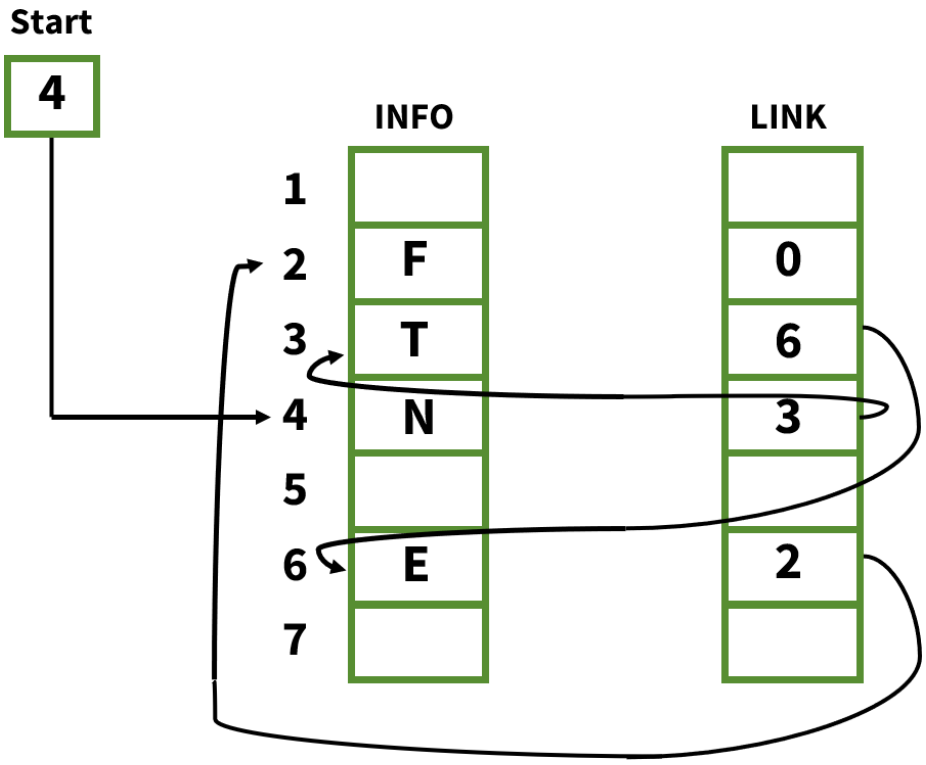
- The last node of the list contain '\0' NULL which shows the end of the list.
- The linked list contain another pointer variable 'Start' which contain the address the first node.
- Linked List is of 4 types
  1. Single Linked List
  2. Double Linked List
  3. Circular Linked List
  4. Header Linked List

Representation of Linked List in Memory

- Linked List can be represented in memory by means 2 linear arrays i.e. Data or info and Link or address.

- They are designed such that info[K] contains the K^{th} element and Link[K] contains the next pointer field i.e. the address of the K^{th} element in the list.

- The end of the List is indicated by NULL pointer i.e. the pointer field of the last element will be NULL or Zero.



Start = 4

info [4] = N	Link [4] = 3
info [3] = T	Link [3] = 6
info [6] = E	Link [6] = 2
info [2] = F	Link [2] = 0 i.e. the null value
	So the list has ended

Representation of a node in a Linked List

```
Struct Link
 {
 int data ;
 Struct Link * add ;
 };
```

SINGLE LINKED LIST

A Single Linked List is also called as one-way list. It is a linear collection of data elements called nodes, where the linear order is given by means of pointers. Each node is divided into 2 parts.

I. Information
II. Pointer to next node.

OPERATION ON SINGLE LINKED LIST

1. TRAVERSAL

Algorithm:
Display (Start) This algorithm traverse the list starting from the 1st node to the end of the list.

Step-1 : "Start" holds the address of the first node.
Step-2 : Set Ptr = Start [Initializes pointer Ptr]
Step-3 : Repeat Step 4 to 5, While Ptr ≠ NULL
Step-4 : Process info[Ptr] [apply Process to info(Ptr)]
Step-5 : Set Ptr = next[Ptr] [move Print to next node]
[End of loop]
Step-6 : Exit

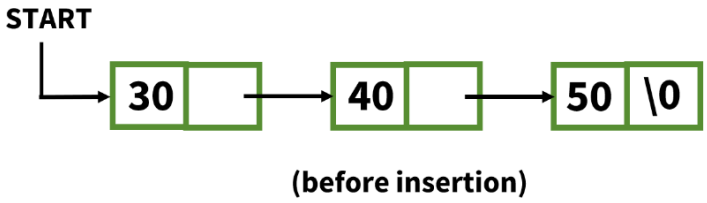
2. INSERTION

The insertion operation is used to add an element in an existing Linked List. There are various position where node can be inserted.

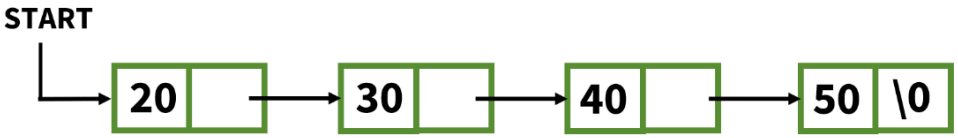
a) Insert at the beginning
b) Insert at end
c) Insert at specific location.

INSERT AT THE BEGINNING

Suppose the new mode whose information field contain 20 is inserted as the first node.



(before insertion)



(After insertion)

Algorithm:

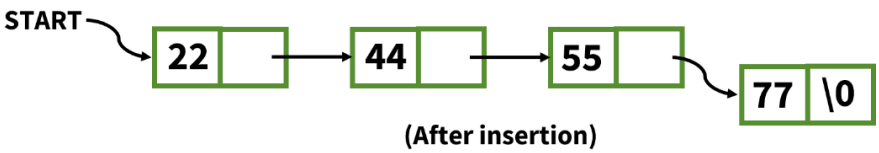
This algorithm is used to insert a node at the beginning of the Linked List. Start holds the address of the first node.

Step-1 : Create a new node named as P.
Step-2 : If P == NULL then print "Out of memory space" and exit.
Step-3 : Set info [P] = x (copies a new data into a new node)
Step-4 : Set next [P] = Start (new node now points to original first node)
Step-5 : Set Start = P
Step-6 : Exit

INSERT AT THE END

- To insert a node at the end of the list, we need to traverse the List and advance the pointer until the last node is reached.
- Suppose the new node whose information field contain 77 is inserted at the last node.

(before insertion)



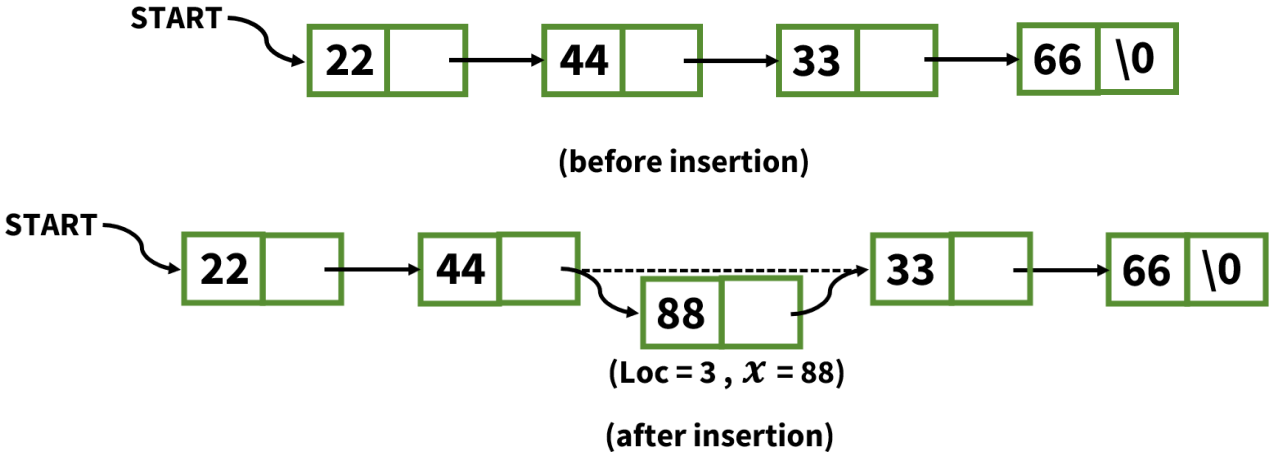
(After insertion)

Algorithm:
This algorithm is used to insert a node at the end of the linked list.
'Start' holds the address of the first node.

Step-1 : Create a new node named as P.
Step-2 : If P = NULL then print "Out of memory space" and Exit.
Step-3 : Set info [P] = x (copies a new data into a new node)
Step-4 : Set next [P] = NULL
Step-5 : Set Ptr = Start
Step-6 : Repeat Step-7 while Ptr ≠ NULL
Step-7 : Set temp = Ptr
Ptr = next [Ptr]
(End of step-6 loop)
Step-8 : Set next [temp] = P
Step-9 : Exit

INSERT AT ANY SPECIFIC LOCATION

To insert a new node at any specific location we scan the List from the beginning and move up to the desired node where we want to insert a new node.
In the below fig. Whose information field contain 88 is inserted at 3rd location.



Algorithm:
'Start' holds the address of the 1st node.

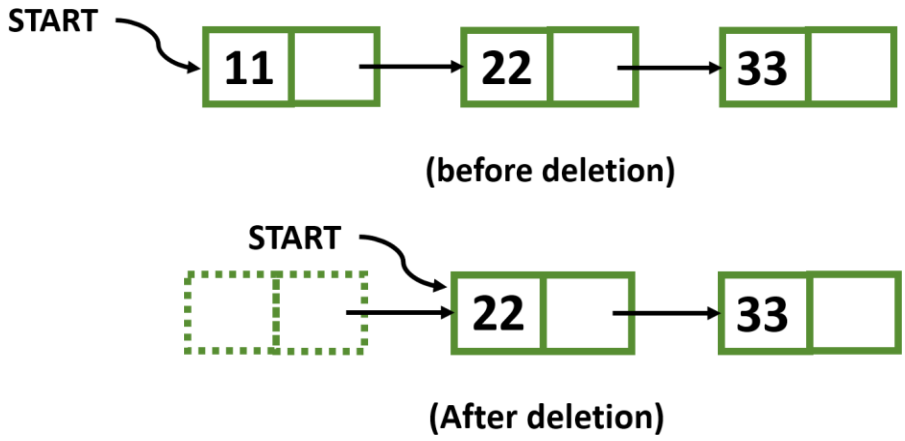
Step-1 : Set Ptr = Start
Step-2 : Create a new node named as P.
Step-3 : If P = NULL then write 'Out of memory' and Exit.
Step-4 : Set info [P] = x (copies a new data into a new node)
Step-5 : Set next [P] = NULL

Step-6 : Read Loc
Step-7 : Set i = 1
Step-8 : Repeat steps 9 to 11 while Ptr ≠ NULL and i < Loc
Step-9 : Set temp = Ptr.
Step-10 : Set Ptr = next [Ptr]
Step-11 : Set i = i + 1
[End of step-7 loop]
Step-12 : Set next [temp] = P
Step-13 : Set next [P] = Ptr
Step-14 : Exit

3. DELETION

The deletion operation s used to delete an element from a single linked list. There are various position where node can be deleted.

a) Delete the 1st Node



Algorithm:
Start holds the address of the 1st node.
Step-1 : Set temp = Start
Step-2 : If Start = NULL then write 'UNDERFLOW' & Exit.
Step-3 : Set Start = next[Start]
Step-4 : Free the space associated with temp.
Step-5 : Exit

b) Delete the last node

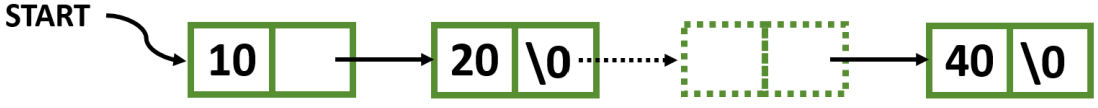
(before deletion)

(After deletion)

Algorithm:
Start holds the address of the 1st node.
Step-1 : Set Ptr = Start
Step-2 : Set temp = Start
Step-3 : If Ptr = NULL then write 'UNDERFLOW' & Exit.
Step-4 : Repeat Step-5 and 6 While next[Ptr] ≠ NULL
Step-5 : Set temp = Ptr
Step-6 : Set Ptr = next[Ptr]
(End of step 4 loop)
Step-7 : Set next[temp] = NULL
Step-8 : Free the space associated with Ptr.
Step-9 : Exit

c) Delete the node at any specific location

(before deletion)



(After deletion (Loc=3))

Algorithm:
Start holds the address of the 1st node.

Step-1 : Set Ptr = Start
Step-2 : Set temp = Start
Step-3 : If Ptr = NULL then write 'UNDERFLOW' and Exit.
Step-4 : Set i = 1
Step-5 : Read Loc
Step-6 : Repeat Step-7 to 9 while Ptr ≠ NULL and i < Loc
Step-7 : Set temp = Ptr
Step-8 : Set Ptr = next[Ptr]
Step-9 : Set i = i+1
(End of Step 6 loop)
Step-10 : Set next[temp] = next[Ptr]
Step-11 : Free the space associated with Ptr.
Step-12 : Exit

4. SEARCHING

Searching means finding an element from a given list.

Algorithm:
Start holds the address of the 1st node.

Step-1 : Set Ptr = Start
Step-2 : Set Loc = 1
Step-3 : Read element
Step-4 : Repeat Step-5 and 7 While Ptr ≠ NULL
Step-5 : If element = info[Ptr] then Write 'Element found at position', Loc and Exit.
Step-6 : Set Loc = Loc+1
Step-7 : Set Ptr = next[Ptr]
(End of step 4 loop)
Step-8 : Write 'Element not found'
Step-9 : Exit

UNIT-9

FILE MANAGEMENT AND HASHING

Concept of File:

- A file is an external collection of related data treated as a unit.
- Files are stored in auxiliary/secondary storage devices.
  - Disk
  - Tapes
- A file is a collection of data records with each record consisting of one or more fields.
- A file is a collection of data records grouped together for purpose of access control and modification.
- It is the primary resource in which we can store information and can retrieve the information when it is required.
- The absolute file name consists of:
  - drive name
  - directory name(s)
  - file name
  - Extension

  For example: d:/network/LAN.doc

Terms Used with Files

- **Field:** This is the basic element of data which contains a single value and characterized by its length and data type.
- **Record:** This is the collection of related fields that can be treated as a unit by some application program.

Extension	Type of Document	Application
.doc	Word-processing document	Microsoft Word 2003
.docx	Word-processing document	Microsoft Word 2007
.wks	Word-processing document	Microsoft Works word processing
.wpd	Word-processing document	Corel WordPerfect
.xls	Spreadsheet	Microsoft Excel
.slr	Spreadsheet	Microsoft Works spreadsheet
.mdb	Database	Microsoft Access
.ppt	PowerPoint Presentation	Microsoft PowerPoint
.pdf	Portable Document Format	Adobe Acrobat or Adobe Reader

Types:

Operations:

- Different types of operations can be performed on the file.
  - Create

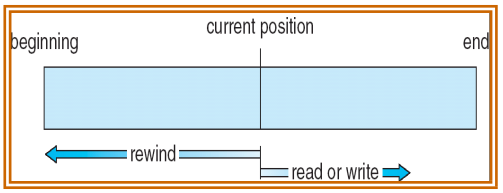
- Delete
- Open
- Close
- Read
- Write

FILE ACCESSING METHODS

Information is stored in files and files reside on secondary storage. When this information is to be used, it has to be accessed and brought into primary main memory. Information in files could be accessed in many ways. There are 3 different file access methods.

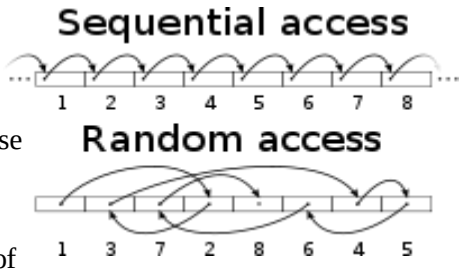
- **Sequential access method**
- **Direct access method**
- **Indexed sequential access method**

Sequential Access Method:



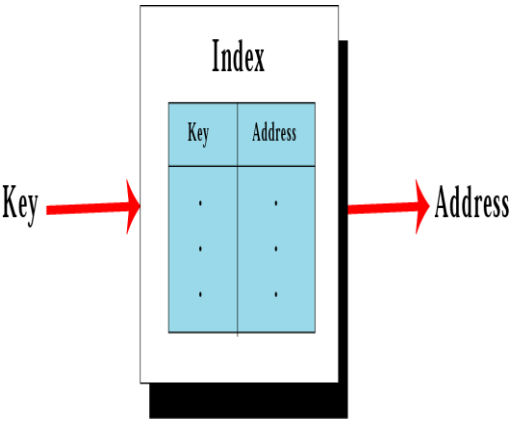
- This method is simplest among all methods. Information in the file is processed in order, one record after the other.
- Magnetic tapes are supporting this type of file accessing.
- Ex: A file consisting of 100 records, the current position of read/write head is 45th record, suppose we want to read the 75th record, then it access sequentially from 45,46,. 74,75.
- So, the read/write head traverse all the records between 45 to 75.
- Sequential files are typically used in batch application and payroll application.

Direct access/ Random access method:



- Direct access is also called as relative access.
- In this method records can read/write randomly without any order.
- The direct access method is based on disk model of a file, because disks allow random access to any file block.
- A direct access file allows arbitrary blocks to be read or written.
- For ex. A disk consisting of 256 blocks, the current position of read/write head is at 95th block. The block to be read or write is 250th block. Then we can access the 250th block directly without any direction.
- Best example for direct access is a CD consisting of 10 songs, at present we are listening the song no.3, suppose we want listening the song no.9, then we can shift from song no.3 to 9 without any restriction.

Indexed sequential access method:

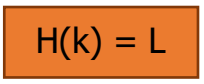


- The main disadvantage in the sequential file is, it takes more time to access a record, to overcome this problem, we are using this method.
- In this method, the records are stored sequentially for efficient processing. But they can be accessed directly using index or key field. That's why this method is said to be the indexed sequential file.
- Keys are pointer which contains address of various blocks.
- Records are organized in sequence based on a key field.
- Generally indexed files are used in airline reservation system and payroll packages.

Address Calculation or Hashing

Hashing is the function to generate a physical address from the record key value.

So hashing is a process to generate the physical address from key value in direct access file.



Where H -> function
K -> key
L -> address

Characteristics of hashing function

- It should be very easy and quick to compute.
- The hash function should be as far as possible uniformly distribute the hash address through out the set L, so that there will be minimum no. of collision.

Types of hash function

1) Division Remainder method
2) Mid square method
3) Folding method

Division- Remainder Method

choose a number m larger than the number n of keys in K. (m is usually either a prime number or a number without small divisor) the hash function H is defined by

H(k) = k (mod m) or **H(k) = k (mod m) + 1**.

here k (mod m) denotes the reminder when k is divided by m. the second formula is used when we want a hash address to range from 1 to m rather than 0 to m-1.

The remainder is calculated physical address for that record.
Suppose key value of the record is 2345.
i.e. k =2345 and m = 97
then H(k) = 2345 % 97 = 17
So that particular record will be sorted in 17 location of memory.
Generally m should be a prime number.

Mid Square method

The key value is squared. Then the mid digits are the address for that key.
Suppose key value = 2345
k^2 = 5499025
If the memory is of two bit then mid will be 99.

Folding Method

In folding method the key value is folded into no. of parts, where each part contains same no. of digits except possibly the last.
$H(k) = k_1 + k_2 + k_3 \ldots . k_n$
Suppose key value = 2356

k = 2356 k_1 = 23, k_2 = 56
23+56 = 79
So the address for that record is 79.

Collision

Suppose we have calculated an address L for a given key value k. But if that location L is previously occupied by some other record then that situation is known as collision.
So if the hash function will generate same address for two different key values. Then it is said to be collision.
Consider the division remainder method.
Let k_1 = 150 , k_2 = 184 , m =17
H (k_1) = 150 mod 17 = 14
H (k_2) = 184 mod 17 = 14
So for key values 150 and 184 the calculated address is 14. This is collision.

There are various collision resolution technique.

- Linear probing (also called open addressing or array method)
- Quadratic Probing
- Double hashing
- Separate Chaining (also called closed hashing or linked list method)

Advantages

- Accessing a record is faster.
- No need to arrange the record on a sorted order.
- If required, records can be processed sequentially.
- Insertion, deletion is easier

Disadvantage

- Expensive I/O device as compared to sequential file structure.
- Address generation overhead due to hashing function.
- Updation of all record is inefficient as compared to sequential file.

Application

Interactive online application such as airline and railway reservation system.