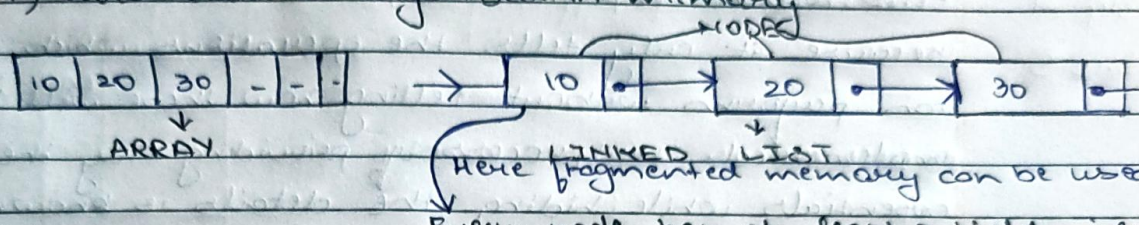
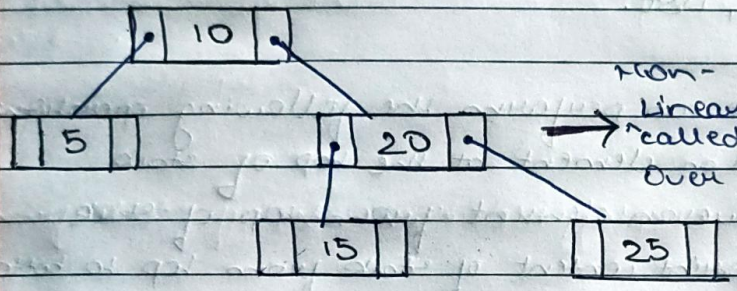
Data Structure

Data structure is a way of organising data in memory so that it can be used efficiently. It consists of two parts

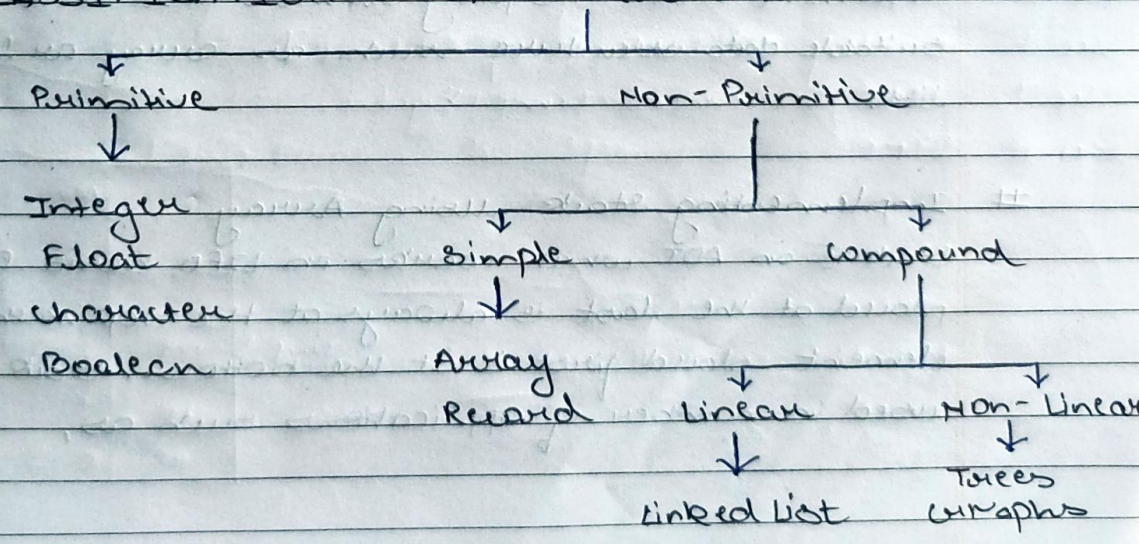
i) How data is organised in memory





ii) What all operations can be performed on that data structure

→ CLASSIFICATION OF DATA STRUCTURE



ADT {Abstract Data Type}

ADT is a conceptual or mathematical model which specifies different type of operations on that data type. However, it does not specify how these operations will be implemented, it is called abstract because it only tells the operations which ADT can perform without telling how these will be implemented. This process of providing only the essentials while hiding the details is known as abstraction. Examples of Abstract Data Type are Stack, Queue, List.

In Stack we can perform the following operations,

i) Push: To add an element at the top of stack
ii) Pop: To remove an element from top of stack
iii) Display: To print content of stack from top to bottom
iv) PeeR: To print topmost element.

These operations are specified in stack ADT which works on LIFO{Last In First Out} basis. To implement stack the programmer will be using a suitable data structure such as array or linked list.

Implementing Stack Using Array

Stack is an ADT which works on LIFO basis. Element placed at the last is always at the top as the element placed first is at the bottom. Stack is used in number of applications such as,

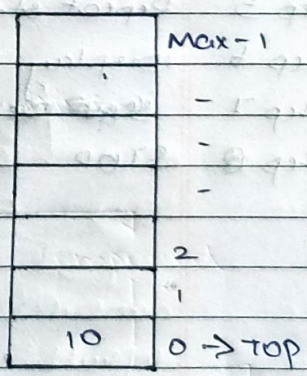
→ Converting infix expression to postfix
→ Evaluating postfix expression
→ Reverse a string
→ Checking the parenthesis
→ Back Tracking
→ Storing the return address in function call
→ Memory Allocation
→ Compiler Design

We use static memory allocation for allocating the array memory i.e. memory will be allocated during compilation time.

☆ Algorithm to push an element

Step 1 Start
Step 2 If Top = Max-1, then display "Stack is full", goto Step 6
Step 3 Input the element to be pushed
Step 4 Increment top by 1
Step 5 Assign element to A[Top]
Step 6 STOP

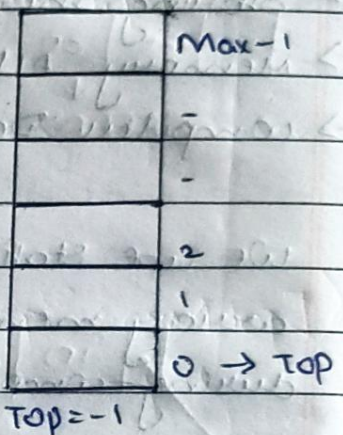
If stack is empty then Top = -1



✱ Algorithm to pop an element

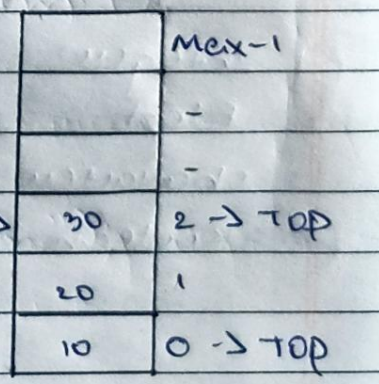
Step 1 START
Step 2 If Top = -1 then display "Stack is empty", goto step 5
Step 3 Display the contents of A[Top]
Step 4 Decrement Top by 1
Step 5 STOP

Over here as stack is empty, Top = -1



✱ Algorithm to display the elements of stack from top to bottom

Step 1 START
Step 2 If Top = -1 then display "stack is empty", goto step 8
Step 3 Take an integer variable temp
Step 4 Assign top to temp
Step 5 Repeat step 6 & 7 until temp = -1
Step 6 Display the contents of A[temp]
Step 7 Decrement temp by 1
Step 8 STOP



* Algorithm to peek

Step 1 START

Step 2 If Top = -1 display "Stack is empty", goto Step 4

Step 3 Display A[Top]

Step 4 STOP

	Max-1
	-
	-
30	2 -> Top
20	1
10	0 -> Top

Code for STACK OPERATIONS

```
void main()
{
    int arr[10], max=10, top=-1, c;
    printf("STACK OPERATIONS \n");
    printf(" 1. PUSH \n
             2. POP \n
             3. PEEK \n
             4. DISPLAY \n"
             5. Exit \n");
    printf("Enter your choice: ");
    scanf("%d", &c);

    switch(c)
    {
        case 1: printf("Enter a number: ");
                scanf("%d", &arr[top]);
                break;
if(top == max-1)
    printf("Stack is full");
else
```

```
case 2: if (top == -1)
            printf("Stack is empty");
        else
        {
            printf("%d is getting deleted",
                    arr[top]);
            top = top - 1;
        }
        break;
case 3: printf("%d is at the top", arr[top]);
        break;

case 3: if (top == -1)
            printf("Stack is empty");
        else
            printf("%d is at the top", arr[top]);
        break;

case 4: if (top == -1)
            printf("Stack is empty");
        else
    for (int temp = top; temp != -1; temp--)
            printf("%d", arr[temp]);
        break;

case 0: break;
default: printf("Invalid choice.\n");
    }
}
```

Queue

Queue ADT can be implemented using an array or linked list data structure. Here we are implementing queue ADT using an array

Applications of Queue

-> Processor scheduling
-> Device scheduling
-> In implementing BFS (Breadth First Search) for graph traversal algorithm
-> In service provider systems where there are multiple inquiries to less no. of handlers.
-> In flight landing & take off systems, as more than one aeroplane compete for single runway at a given time for landing and take off.

Queue Operations

-> Enqueue: To add an element at the rear end of the queue
-> Dequeue: To remove an element from front of the queue
-> Display: To print all elements in queue from front to rear
-> Peek: To print front element of the queue

Implementing Queue using an array

Assumptions: We have an integer array with a size max, where max is a constant. Take two integer variables front & rear & initialize them to -1 as initially queue is empty.

☆ Algorithm to Implement Queue

Step 1 START

Step 2 If ((rear+1) % max) = front then display a message queue is full, goto step 7

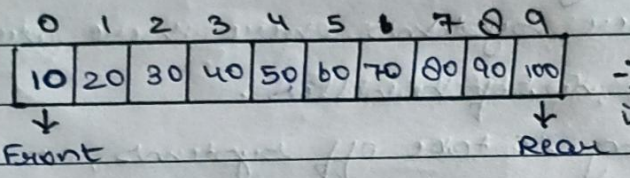
Step 3 Input the element to be inserted

Step 4 rear = (rear+1) % max

Step 5 Assign the element to a[rear]

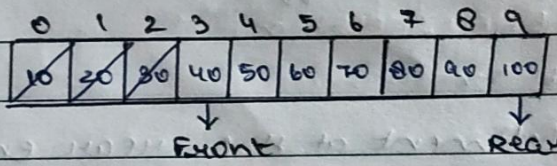
Step 6 If front = -1 it means element inserted is first element then increment front to 0.

Step 7 STOP



Max=10

-> Queue is full
i.e. (rear+1) % max = front
(9+1) % 10 = 0
(10 % 10) = 0
0 = 0



-> Queue is empty
i.e. (rear+1) % max ≠ front
(0+1) % 10 ≠ 3
(1 % 10) ≠ 3
1 ≠ 3

If (rear+1) % max = front → Queue is full

☆ Algorithm to implement Dequeue

Step 1 START

Step 2 If front = -1 then display a message queue is empty.

Step 3 goto step 6.

Step 4 If front = rear then display the value of a[front] & reset front & rear to -1, goto step 6

Step 5 Display a[front]
front = (front+1) % max

Step 6 STOP

* Algorithm to implement display

Step 1 START

Step 2 If front = -1 then display a message that queue is empty, goto step 8

Step 3 Take an integer temp to assign value of front to it

Step 4 while (temp != rear) repeat step 5 & 6

Step 5 Display a[temp]

Step 6 temp = (temp + 1) % max

Step 7 Display a[temp]

Step 8 STOP

* Algorithm to implement peek

Step 1 START

Step 2 If front = -1 then display a message that queue is empty, goto step 4

Step 3 Display the value of a[front]

Step 4 STOP