Data Structures

Assignment - 1

19101-CM-202
R. Jyothiprakash

1. Define and classify the data structures?

A Definition:- A data structure is collection of data elements organized in a specific manner, characterized by its own specific methods of accessing data elements.

Data structures are classified into 2 types.

They are :- ① primitive type
② Non-primitive type.

Primitive type:-

These are basic structures and directly operated upon the machine instructions. In this primitive type are classified in to 4 types. They are

1. Integer
2. float
3. character
4. pointer.

Integer:-

An integer is a whole number that can be positive, negative or zero. Integers are also used to determine an item's location within an array.

Ex:- 10, 0, -25, 5, 14, 8 etc....

Float:-

Float is a shortened term for floating point. It is a data type in many languages. A number which have a decimal point in number is called float.

Ex:- 10.5, 2.2, 5.33, 5148.2 etc.;

Character:-

It stores strings of letters, numbers and symbols. character data can be stored fixed length or variable - length strings.

Ex:- Array, Bitcoin.

pointer:-

A pointer is a variable whose value is the address of another variable.

Ex:- int * ptr;

int a = 10;

ptr a = * ptr;

Non- primitive Data structures:-

Non- primitive data structures are more complicated data structures and are derived from primitive data structures. They emphasize on grouping same or different data items with relationship between each data item. Arrays, lists and files are come under this category.

Array :-

An array is a collection of data items, all of the same type, accessed by using a common name is called Array.

Ex :- int a[5] = {1, 2, 3, 4, 5};

Files :-

A file represents a sequence of bytes on the disk where a group of related data is stored. File is created for permanent storage of data.

Lists :-

A linked list is a linear collection of data elements whose order is not given by their physical placement in memory. In this lists are two types. They are :- 1. Linear
2. Non-linear

Linear lists :-

A linked list is a linear data structure where each element is a separate object. Each element of a list is comprising of two items. The data and a reference of the next node. The last node has a reference to null. In this linear lists are divided in to two types. They are 1. Stacks
2. Queues.

Stacks :-

A stack is a basic data structure that can be logically thought of as a linear structure represented by a real physical stack or pile. A structure where insertion and deletion of items place at one end called top of the stack.

Queues :-

Queues is an abstract data structures, Some what similar to stacks, Unlike stacks, a Queue is open at both its ends. One end is always used to inserted data and the other is used to remove data. Queue follows First-In-First-Out methodology.

Non-linear lists :-

Non-linear data Structures is a data Structure in which data items are not stored linearly in the memory allocation of the data. This feature is included because it uses the memory optimally. In this Non-linear lists are divided in to 2 types. They are :- ① Graphs

② Trees

Graphs:- Graphs are used to represent non-hierarchical relationship among data elements.

Trees:- Trees are used to represent hierarchical relationship among data elements.

Data Structures

19101-CM-202

Assignment-2

R. Jyothiprakash.

1. Define stack?

Stack is a collection of similar data items in which both insertion and deletion operations are performed based on LIFO [Last - In - First - Out] principle.

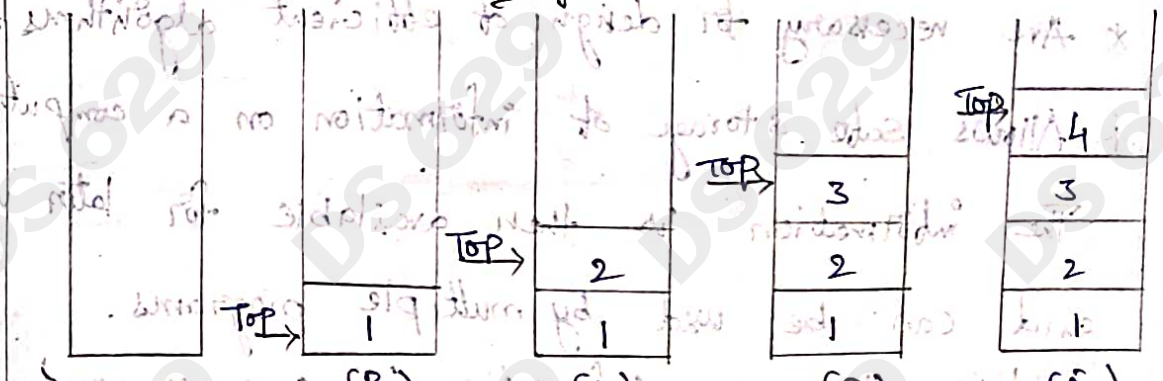
2. Explain the operations of stack?

A. There are two operations we perform in Stack. They are 1. Push operation
2. Pop operation.

1. Push operation:-

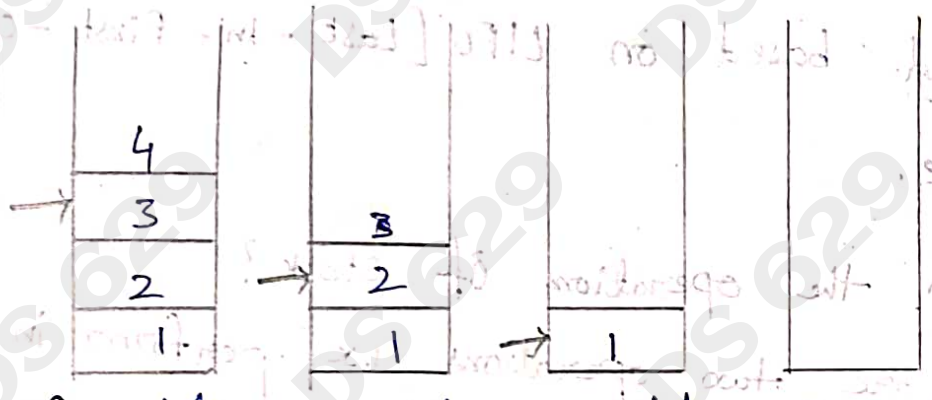
In case of stack insertion of any item in stack is called push. In stack any item is inserted from the top of the stack, when you insert any item in stack top will be increased by 1.

Diagram



Pop Operation:-

In case of stack deletion of any item from stack is called pop. In any item is delete from top of the stack, when you delete any item from stack top will be decreased by 1.



(f) popdata popped in = 4 (g) popdata popped in = 30 (h) pop data popped in = 20 (i) pop data popped in = 10.

Pop operation Diagram.

3. What are the advantages of Stacks?

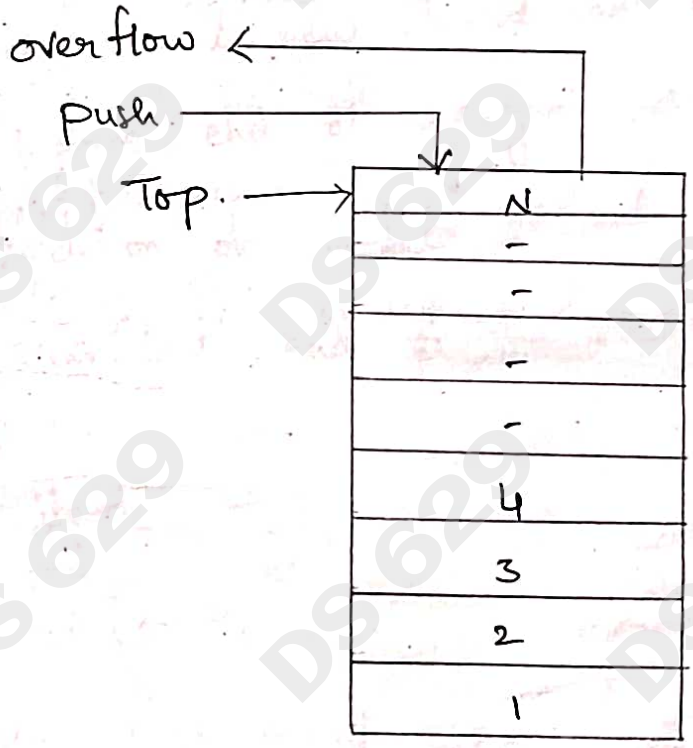
A. * Data structure allows information storage on hard disks.

* Provides means for management of large data set such as databases or internet indexing services.
* Are necessary for design of efficient algorithms.
* Allows safe storage of information on a computer. The information is then available for later use and can be used by multiple programs. Additionally, the information is secures and can not be lost.

* Allows the data use and processing on a software system.
* Allows easier processing of data.
* Using internet, we can access the data anytime from any connected machine (computer, laptop, tablet, phone etc.).

4. Explain Overflow and underflow of stacks?

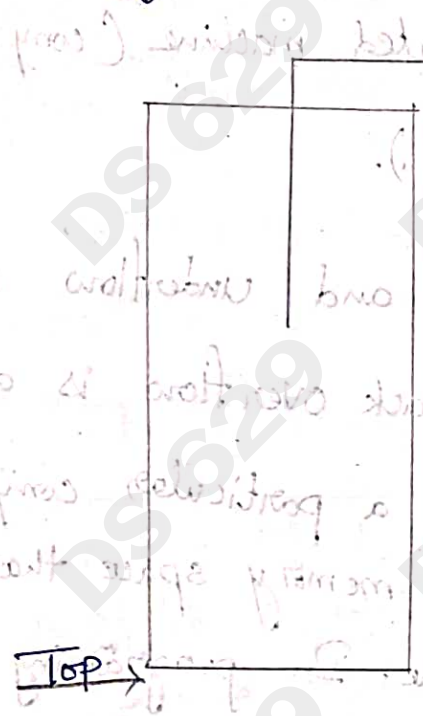
A. Overflow:- A stack overflow is an undesirable condition in which a particular computer program tries to use more memory space than the call stack has available. In programing, the call stack is a buffer that stores requests that need to be handled.



Push causes over flow when stack is full.

Underflow:-

An error condition that occurs when an item is called for from the stack, but the stack is empty is called stack underflow.



pop causes underflow when stack is empty.

Data Structures

19101-CM-202

R. Jyothiprakash.

Assignment - 3

1. Define Queue?

A. A Queue is a first-In-First-Out (FIFO) data structure, in which additions are made at one end and deletions are made at the other end.

2. Explain the operations of Queue?

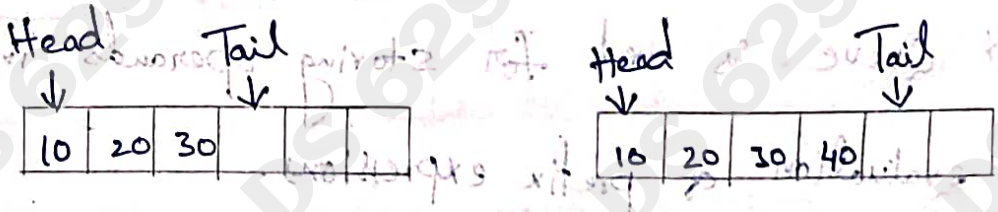
A. In Queue performs two types of operations.

They are:- 1. Add

2. Remove.

Add operation:-

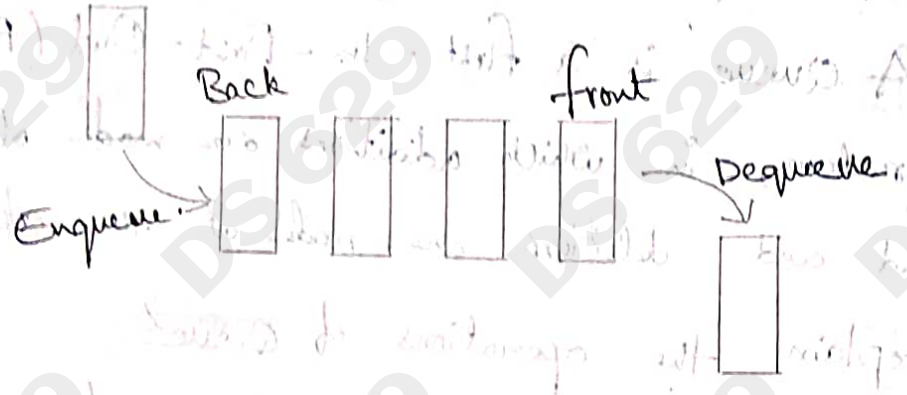
Add operation is to add one or more data elements in to an array. Based on the requirement, new element can be added at the begining, end or any given index val of array. Addition of a data element is allowed only at rear (tail) end of the Queue.

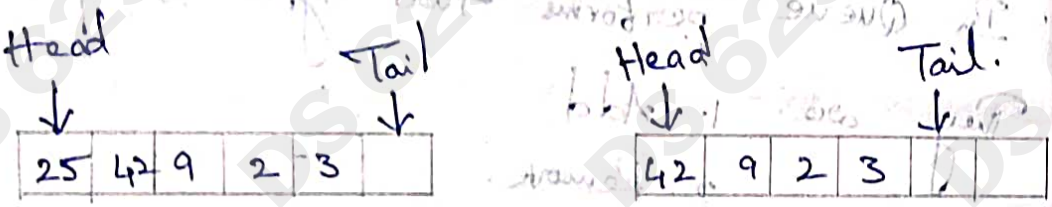


Before add operation After add operation.

Remove operation :-

Remove of a data element is allowed only from the front (head) end of the queue.





Before Remove operation

After Remove operation.

3. What are the advantages of Queues?

A * When a resource is shared among multiple consumers. Example include cpu scheduling, Disk Scheduling.

* When data is transferred asynchronously between two processes. Examples include IO buffer, Pipes, file IO, etc.

* Queue is used for storing operands in evaluation of prefix expressions.

* Queues are also useful in batch-processing of transactions for updating stock/cash balance.

Data Structures

19101-CM-202

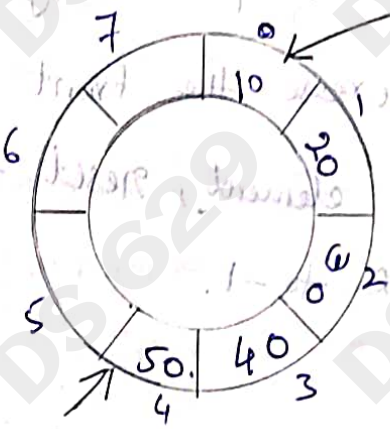
Assignment-4

R. Jyothiprakash

1. What is circular Queue? Explain briefly?

A. Circular Queue is a linear data structure in which the operations are performed based on FIFO (First-In-First-Out) principle and the last position is connected back to the first position to make a circle. It is also called "Ring Buffer".

Ex:-



The circular queue work as follows:-

* two pointers front and Rear
* Front track the first element of the Queue.
* Rear track the last element of the Queue.
* Initially, set value of front and Rear to -1.

In this circular Queue performs 2 operations.

They are * EnQueue
* DeQueue.

Enqueue operations :-

* Check if the Queue is full.
* for the first element, set value of front to 0.
* circularly increase the Rear index by 1.
* Add the new element in the position pointed to by REAR.

Dequeue operation :-

* Check if the Queue is empty.
* Return the value pointed by front.
* Circularly increase the Front index by 1
* For the last element, reset the values of Front and REAR to -1.

Program :-

```
#include <stdio.h>
#define SIZE 5
int items [SIZE];
int front = -1, rear = -1;
int isFULL()
{
  if ((front == rear+1) || (front==0 && rear == SIZE-1))
                                    return 1;
  return 0;
}
```

```
int isEmpty()
{
 if (front == -1) return 1;
 return 0;
}
void enQueue (int element)
{
 if (isFULL())
   printf ("\n Queue is full !! \n");
 else
 {
 if (front == -1) front = 0;
 rear = (rear + 1) % SIZE;
 items [rear] = element;
 printf ("\n inserted -> %d", element);
 }
}
int deQueue()
{
 int element;
 if (is empty()):
 {
   printf ("\n queue is empty !! \n");
   return (-1);
 }
 else
```

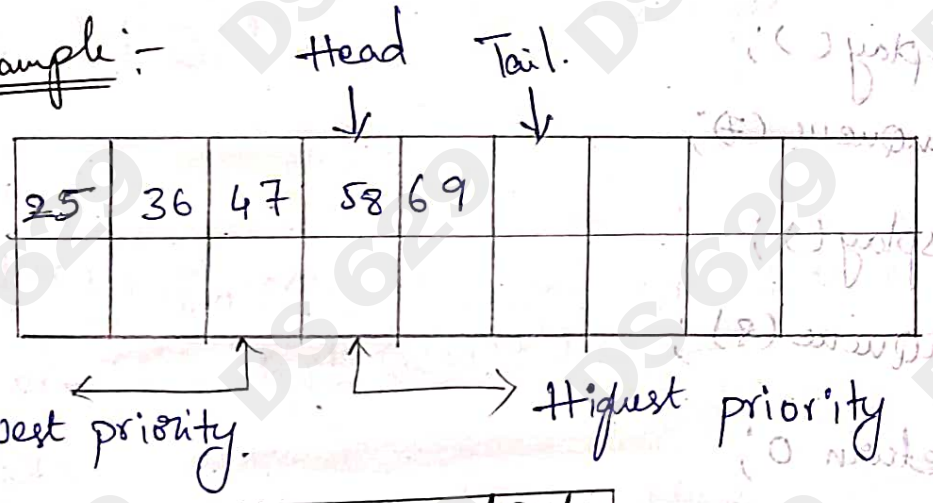
```
{
element = items [front];
if (front == rear)
{
  front = -1;
  rear = -1;
}
else
{
 front = (front + 1) % SIZE;
}
printf("\n Deleted element -> %d \n", element);
return (element);
}
}
void display()
{
int i;
if (isEmpty())
 printf("\n Empty Queue \n");
else
{
printf("\n front -> %d", front);
printf ("\n items ->");
for (i= front; i != rear; i=(i+1) % SIZE)
{ printf ("%d", items [i]);
}
```

```
        printf("%d", items[i]);
    printf("\n rear -> %d \n", rear);
    }
}
int main()
{
    deQueue();
    enQueue(1);
    enQueue(2);
    enQueue(3);
    enQueue(4);
    enQueue(5);
    enQueue(6);
    display();
    deQueue();
    display();
    deQueue(7);
    display();
    enQueue(8);
    return 0;
}
```

2. What is priority Queue? Explain briefly?

A: The persons in the Queues are attended on a First - cum - first served basis. In some situations, though there is a Queue or waiting list, a particular person who comes late not put at the end of the Queue, but gets attention immediately, as a special case. The emergency quota in railway reservation, the special darshan system in Tirupathi Balaji temple, the caste wise lists of candidates for engineering / medical college admission and for recruitment/ promotion in Government departments are all examples of such "Priority Queues".

Example:-



Order of output	
Data	Priority
58	1
25	2
36	2
69	3
47	3

Program :-

```
#include <stdio.h>
int size=0;
void swap(int *a, int *b)
{
  int temp = *b;
  *b = *a;
  *a = temp;
}
void heapify (int array[], int size, int i)
{
 if (size == 1)
 {
  printf(" Single element in the heap");
 }
 else
 {
  int largest = i;
  int l = 2 * i + 1;
  int r = 2 * i + 2;
  if (l < size && array[l] > array[largest])
    largest = l;
  if (r < size && array[r] > array[largest])
    largest = r;
  if (largest != i)
  {
   swap(&array[i], &array[largest]);
   heapify (array, size, largest);
  }
```

```
    }
  }
}
Void insert (int array[], int newNum)
{
  if (size ==0)
  {
   array [0] = newNum;
   size +=1;
  }
  else
  {
  array [size] = newNum;
   size +=1;
  for (int i = size /2 -1; i>=0; i--)
  {
   heapify (array , size, i);
  }
}
}
void deleteRoot (int array[], int num)
{
  int i;
  for (i=0 ; i< size; i++)
  {
   if (num = = array [i])
   break;
  }
```

```
    swap(&array[i], &array[size-1]);
    size -= 1;
    for(int i = size/2-1; i>=0; i--)
    {
      heapify(array, size, i);
    }
  }
void printArray(int array[], int size)
  {
   for(int i=0; i< size; ++i)
    printf("%d", array[i]);
    printf("\n");
  }
int main()
{
  int array[10];
   insert(array, 3);
   insert(array, 4);
   insert(array, 9);
   insert(array, 5);
   insert(array, 2);
   printf("Max-Heap array:");
   printArray(array, size);
   deleteRoot(array, 4);
```

```
Printf("After deleting an element:");
printArray(array, size);
}.
```

Data structures

Assignment -5

19101-CM-202
R. Jyothiprakash.

Implementation of stack :-

Algorithm for stack using Array :-

Creation of stack & Initialisation

1. Declare an integer array of N elements, name 'stak'. (N is the max no. of items in stack).
2. Declare 3 integer variable : sp (stack pointer), empty & full.
3. Initialise : empty = -1;
   full = N-1;
   sp = empty.

Push

1. If sp = full;
   display message 'stack is full';
   go to step 4.
   [else : go to step 2]
2. Increment SP.
3. Store data at Stak[SP]
4. End.

POP

1. If sp = empty :
   display message 'stack is empty';

goto step 4.
[else : goto step 2]

2. Take data from [SP] and display it.
3. Decrement SP.
4. End.

Program :-

```
/* Stack using Array */
#include <stdio.h>
# include <conio.h>
int Stak [10];
int sp = -1;
void push (void);
void pop (void);
void main(void)
{
  int n;
  char opt = '1';
  while (opt == '1' || opt == '2')
  {
    clrscr();
    printf("STACK OPERATION \n\n");
    printf("\n 1 push");
    printf("\n 2 pop");
    printf("\n 3 Quit");
    printf("\n\n Choice :");
```

```
    opt = getche();
    switch (opt)
    {
     case '1' : push();
     break;
      case '2' : pop();
      break;
    }
  }return;
 }
void push (void)
{
 int n;
 if (sp ==9)
 {
  printf ("\n\n cannot push:");
  printf (" \n stack is full.");
 }
else
 {
  printf("\n\n data to push:");
  scanf ("%d", &n);
  sp++;
  stack[sp] = n;
  printf (" \n Done.");
 }
 getch ();
 return;
}
```

```
Void pop (void)
{
  if (sp == -1)
  {
    printf ("\n\n cannot pop.");
    printf ("\n\n Stack is empty.");
  }
  else
  {
    printf("\n\n Data popped = %d.", stak[sp]);
    sp--;
  }
  getch();
  return;
}
```

Creation & Initialisation of Queue :-

1. Declare an array of N integers, named que (N is the max no. of items in Queue).
2. Declare 3 integers variables : head, tail, count.
3. Initialise : head = 0, tail = 0, count = 0;

Add :-

1. If count = N
   display message 'Queue is full';
   goto step 5.
2. store data at que[tail].
3. Increment tail.

4. Increment count.
5. End (return).

Remove:-

1. If count = 0
   display message 'Queue is empty';
   goto step 6.
2. Read data from que[head]
3. Increment head
4. Decrement count.
5. If count = 0;
   Set head = 0;
   Set tail = 0.
66. End [return].

Program:-

```
/* Array implementation of Queue */.
#include <stdio.h>
#include <conio.h>
int que[10];
int head=-1, tail=0;
void add (void);
void remove (void);
void main(void)
{
   char opt = '1';
   while (opt == '1' || opt == '2')
```

```
{
  clrscr();
  printf(" QUEUE (non-moving) \n\n");
  printf(" \n 1 Add ");
  printf (" \n 2. Remove ");
  printf (" \n 3 Quit ");
  printf (" \n\n Choice :");
  opt = getche();
  switch (opt)
  {
     case '1': add();
               break;
     case '2': remove();
               break;
  }
}
return;
}
void add (void)
{
  int n;
  if (tail == 10)
  {
     printf (" \n\n cannot add :");
     printf (" \n Queue is full.");
  }
```

```
else
{
    printf ("\n\n data to add :");
    scanf ("%d", &n);
    que [tail++] = n;
    if (head == -1)
    head++;
    printf ("\n Added");
}
getch()
return();
}
void remov (void)
{
    int n;
    if (head == -1)
    {
    printf ("\n\n\ncannot remove.");
    printf ("\n Queue is empty.");
}
else
{
    n = Que [head++]
    if (head == tail)
    {
        head = -1;
        tail = 0;
    }
    printf ("\n\n Data removed = %d", n);
}
getch();
return();
}
```

Data Structures

Assignment

1. Explain how to perform insertion/ deletion operation on a single linked list?

A. **Definition** :- A linked list it an ordered collection of elements called nodes each of which has two parts.

1. A data part that stores an element of the list.
2. A next part that stores a link or pointer that indicates the location of the node containing the next list element. If there is no next element, then a special null value is used.

Inserting node at start in the single linked list:-

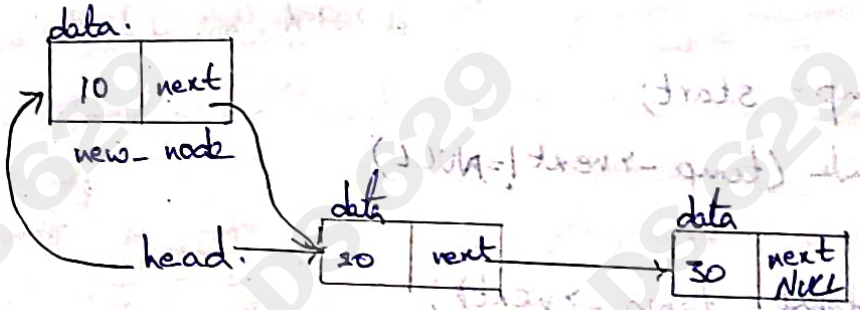
1. Create New node.
2. Fill data in to "Data field".
3. Make its "pointer" (or) "Next field" as Null.
4. Attach this newly created node to start.
5. Make newnode as starting node.

Ex:-

```
Void insert_at_beg()
{
Struct node *new_node, *current;
```

```
new node = (struct node *) malloc (size of (struct node));
if (new node == Null)
    printf(" failed to Allocate memory");
printf(" Enter the data:");
scanf (" %d", &new_node -> data);
new node -> next = NULL;
if (start == NULL)
{
start = New node;
Current = New-node;
}
else
{
new_node -> next = start;
start = new_node;
}
}
```

Diagram:-



Inserting node at last/End position in single linked list :- 1. Create a new node.

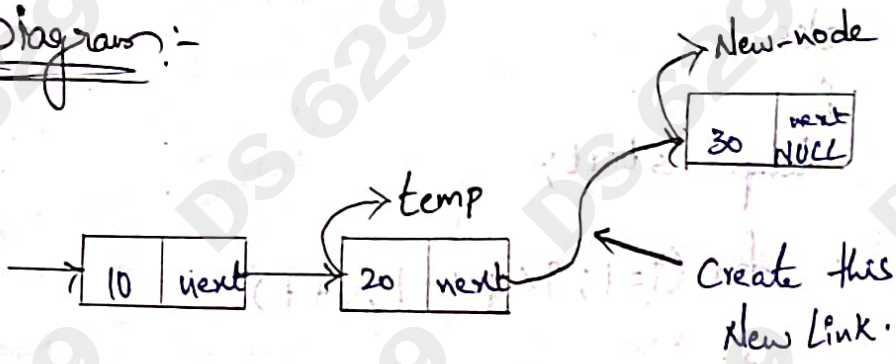
2. Fill data in to "Data Field".
3. Make its "pointer" (or) Next field as NULL
4. Node is to be inserted at last position so we need to traverse single linked list upto last node.

5. Make link between lastnode and newnode.

Example :-

```
Void insert_at_end()
{
 struct node * new node , * current;
 new node = (struct node * ) malloc (size of (struct node));
 if (new_node == NULL)
    Printf (" Failed to Allocate Memory");
 printf (" Enter the data ");
 scanf (" %d", &new_node -> data);
 new_node -> next = NULL;
 if (start == NULL )
 {
   start = New node;
   current = New node;
 }
 else
 {
   temp = start;
   While (temp -> next != NULL)
   {
     temp = temp -> next;
   }
 temp -> next = new node;
 }
}
```

Diagram:-



Insert node at middle position in single linked list:-

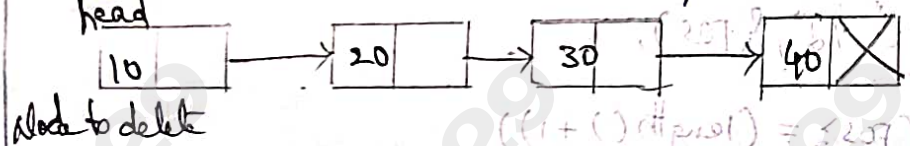
```
Void insert mid()
{
  int pos,i;
  struct node *newnode, *current, *temp, *temp1;
  newnode=(struct node*) malloc (sizeof(struct node));
  printf(" Enter the data :");
  scanf("%d",&newnode->data);
  newnode->next=NULL;
  st:
  printf(" Enter the position: ");
  scanf(" %d", &pos);
  if (pos>=(length()+1))
  {
    printf(" Error : pos > length ");
    goto st;
  }
  if (start ==NULL)
  {
    start = new node;
    current = new node;
  }
```

```
else
{
 temp = start;
   for (i=1; i<pos-1; i++)
   {
     temp = temp -> next;
   }
 temp1 = temp -> next;
 temp -> next = newnode;
 newnode -> next = temp1;
}
}
```

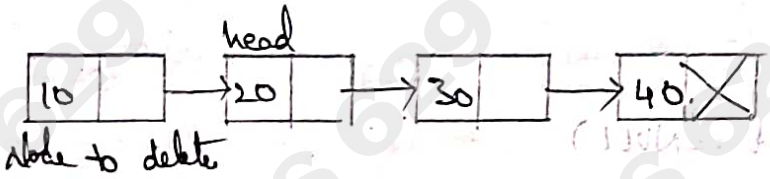
Delete first node from single linked list :-

* Copy the address of first node.

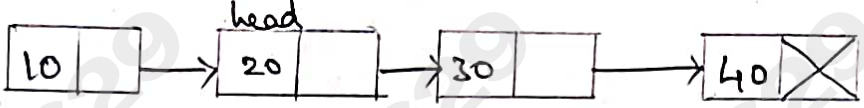
  Ex :- "head" node to some temp variable say "todelete"



* Move the "head" to the second node of the linked list. `head = head -> next`

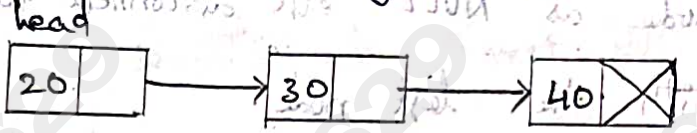


* Disconnect the connection of first node to second node.



Node to delete

* free the memory occupied by the first node.

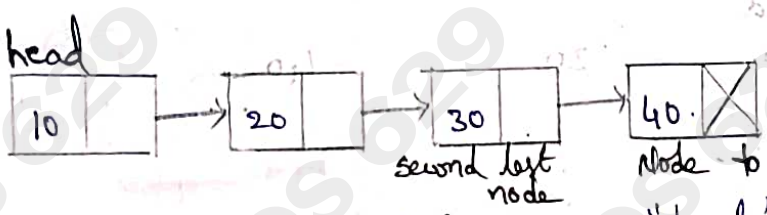


Example :-

```
Void deletefirstnode()
{
  struct node * todelete
   if (head == NULL)
   {
   printf ("List is already empty.");
   }
   else
   {
     todelete = head;
     head = head->next;
     printf("\n data deleted = %d\n", todelete->data);
     free (todelete);
     printf("sucessfully deleted first node from list\n");
   }
}
```

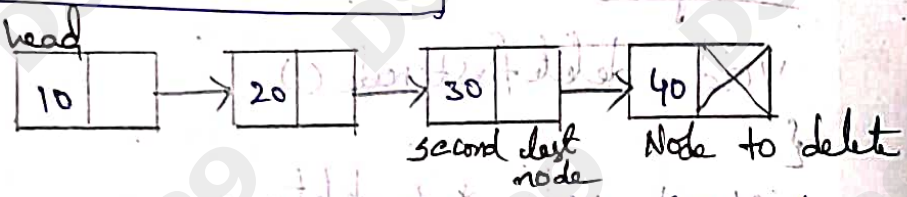
Delete last node of a single linked list :-

* Traverse to the last node of the linked list keeping track of the second last node in some temp variable say "secondlastnode".

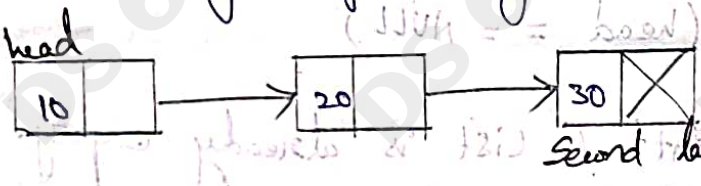


* If the last node is the "head" node then make the head node as "NULL" else disconnect the second last node with the last node.

`Second last node -> next = NULL`



* Free the memory occupied by the last node.



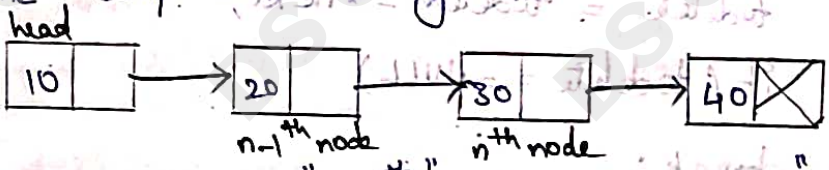
Example :-

```
Void deletelastNode( )
{
struct node * todelete , * secondlastnode;
if (head == = NULL)
{
printf(" List is already empty. ");
}
else
{
todelete = head;
secondlastnode = head;
while (todelete -> next != NULL)
{
secondlastnode = todelete;
todelete = todelete -> next;
}
```

```
if (todelet == need)
{
  head = NULL;
}
else
{
  secondlastnode -> next = NULL;
}
free (todelete);
printf ("successfully deleted last node of list \n");
}
}
```

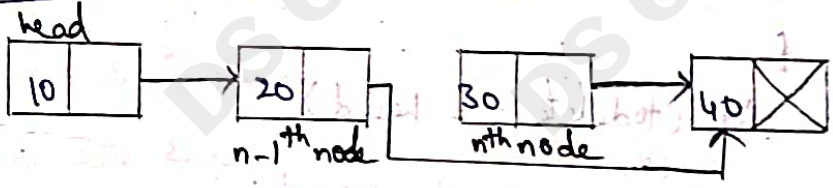
Delete middle node of single linked list :-

* Traverse to the "n^th" node of the single linked list and also keep reference of "n-1^th" node in some temp variable say "prevnode".

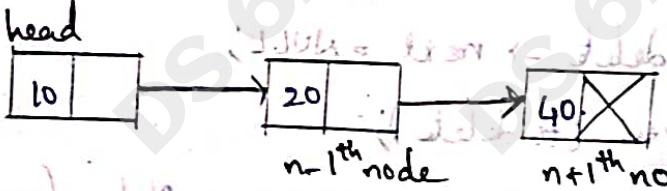


* Reconnect the "n-1^th" node with the "n+1^th" node

```
Prevnode -> next = todelete -> next
```



* Free the memory occupied by the "n^th" node.



Example :-

```
Void delete middlenode (int position)
{
    int i;
    struct node * todelete , * prevnode;
    if (head == NULL)
    {
      printf(" List is already empty.");
    }
    else
    {
      todelete = head;
      prevnode = head;
      for (i=2; i<= position ; i++)
      {
        prevnode = todelete;
        todelete = todelete -> next;
        if (todelete == NULL)
        break;
      }
      if (todelete != NULL)
      {
        if (todelete == head)
          head = head -> next;
        prev node -> next = todelete -> next;
        todelete -> next = NULL;
        free (todelete);
        printf ("sucessfully deleted Node from
                middle of list \n");
    }
```

```
        else
        {
          printf("Invalid position unable to delete.");
        }
      }
    }
}.
```

Assignment-6

Data structures using C

Single linked list program:—

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<ctype.h>
 struct node
 {
   int data;
   struct node *next;
 };
 int count=0;
   struct node *first = NULL;
   struct node * Last = NULL;
   struct node * temp = NULL;
   struct node * prev = NULL;
   struct node * newnode = NULL;
   void append (int);
   void showlist(void);
   main()
   {
     int x;
    char ch;
```

```
ch = 'y';
clrscr();
printf("creating a linked list\n");
while (toupper(ch) == 'Y')
{
  printf("\n input the node");
  scanf("%d", &x);
  append(x);
  printf("You want to create node [Y/n]");
  ch = getch();
}
show list();
}
void append(int x)
{
new node = (struct node *) malloc (sizeof(struct node));
new node -> data = x;
newnode -> next = NULL;
if (first == NULL)
{
  first = newnode;
  printf("List was empty, creating list with one node\n");
}
else
{
  last -> next = newnode;
  count++;
  printf("Appending a newnode to the end of the list \n");
}
```

```
int i;
if (first == NULL)
    printf("\n list is empty\n");
else
{
printf("\n node address data ");
printf ("\n ---------- ---- ----- ");
temp = first;
for (i=0; i<= count; i++)
{
 printf("\n %3d %p %5d", i, temp, temp -> data);
 if (i != count)
 temp = temp -> next;
}
}
getch ( );
return;
}
```

Circular Queue program :-

```
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
int que[10];
int count = 0, head = 0, tail = 0;
void add (void )
void remov (void);
void main (void)
{
```

```
    char opt='1';
    while (opt=='1' || opt=='2')
    {
     clrscr();
     printf("CIRCULAR QUEUE\n\n\n");
     printf("\n 1 Add");
     printf("\n 2 Remove");
     printf("\n 3 Quit");
     printf("\n\n choice:");
     opt = getch();
     if (opt=='1')
        add();
     else if (opt=='2')
        remove();
     }
    return;
  }
void add (void)
{
 int n;
 if (count==10)
 {
  printf("\n\n cannot add.");
  printf("\n Queue is full.");
 }
 else
 {
  printf("\n\n data to add:");
  scanf("%d",&n);
  Que[tail++]=n;
  if (tail>9)
```

```
        tail=0;
        count++;
        printf("\n\n added.");
    }
    getch();
    return(0);
}
void remov(void)
{
  int n;
  if(count==0)
  {
  printf("\n\n cannot remove.");
  printf("\n Queue is empty.");
  }
  else
  {
    n=Que[head++]
    if(head>9)
        head=0;
        count--;
        printf("\n\n Data removed = %d", n);
    }
  getch();
  return;
  }
}
```

Stack program

```
#include <stdio.h>
#include <conio.h>
int stack[10];
int sp=-1;
void push(void);
void pop(void);
void main(void)
{
 char opt='1';
 while(opt=='1'||opt=='2')
{
 clrscr();
 printf("stack operations\n\n");
 printf("\n  1 push");
 printf("\n  2 pop");
 printf("\n  3 quit");
 printf("\n\n choice:");
 opt=getche();
 switch(opt)
 {
  case'1': push();
           break;
  case'2': pop();
           break;
 }
 return(0);
}
void push(void)
{
 int n;
 if(sp==9)
 {
  printf("\n\n cannot push.");
  printf("\n stack is full.");
 }
 else
 {
  printf("\n\n data to push:");
  scanf("%d",&n);
  sp++;
  stack[sp]=n;
  Printf("\n Done");
 getch();
 return(0);
}
void pop(void)
{
 if(sp==-1)
 {
  printf("\n\n cannot pop.");
  printf("\n stack is empty.");
 }
 else
 {
  printf("\n\n data popped=%d.",stack[sp]);
  sp--;
 }
 getch();
}
```

Queue Program

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
int que[10];
int head=-1,tail=0;
void remove(void);
void add (void);
void main(void)
{
  char opt = '1';
  while (opt=='1'||opt=='2')
  {
   clrscr();
   printf("Queue operations : \n\n");
   printf ("\n 1  add");
   printf("\n 2 remove");
   printf ("\n 3 Quit");
   printf ("\n\n choice:");
   opt = getch();
   switch (opt)
   {
    case '1': add()
              break;
    case '2': remov();
              break;
   }
  }
  return;
}
void add (void)
{
 int n;
 if (tail == 10)
 {
  printf ("\n\n cannot add.");
  printf ("\n queue is full.");
 }
 else
 {
  printf ("\n\n Data to add:");
  scanf ("%d", &n);
  que[tail++] = n;
  if (head == -1)
  head++;
  printf("\n Added.");
 }
 getch();
 return;
}
void remov (void)
{
 int n;
 if (head == -1)
 {
  printf ("\n\n cannot remov.");
  printf ("\n Queue is empty.");
 }
 else
 {
  n = que[head++]
  if(head == tail)
  {
   head = -1;
   tail=0;
  }
  printf ("\n\n Data removed = %d",n);
 }
 getch();
 return(0);
}
```

Double Linked List program:-

```
#include <stdio.h>
#include <stdlib.h>
struct Node;
typedef struct Node * PtrToNode;
typedef PtrToNode List;
typedef PtrToNode Position;

struct Node
{
  int e;
  position previous;
  position next;
};
void Insert (int x, List l, Position p)
{
  position Tmpcell;
  Tmpcell = (structNode *)
   malloc(size of (structNode));
   if (Tmpcell == NULL)
     printf ("Memory out of space\n");
   else
    {
     Tmp cell -> e = x;
     Tmp cell -> previous = P;
     Tmpcell -> next = P -> next;
     P -> next = Tmpcell;
    }
}
```

```
int islast (position p)
{
  return(p->next == NULL);
}
position Find(int x List l)
{
  position p = l->next;
  while (p != NULL && p->e != x)
  p = p->next;
  return p;
}
void delete (int x, List l)
{
  position p, P1, P2;
  p = find (x,l);
  if (p != NULL)
  {
    P1 = p->previous;
    P2 = p->next;
    P1 -> next = p->next;
    if (P2 != NULL)
    P2 -> previous = P -> previous;
  }
  else
    printf ("Element does not exist !!! \n");
}
void Display(List l)
{
  printf(" The list element are :: ");
  position p = l->next;
```

```
while (p != NULL)
{
    printf ("%d ->", p->e);
    p = p->next;
}
}
void main()
{
  int x, pos, ch, i;
  List l, l1;
  l = (struct node *) malloc (sizeof (struct Node));
  l -> previous = NULL;
  List p = l;
  printf ("DOUBLY LINKED LIST IMPLEMENTATION
          OF LIST ADT \n\n");
  {
    printf ("\n\n 1. INSERT 2. DELETE 3. FIND
          4. PRINT 5. QUIT \n\n Enter the choice ::");
    scanf ("%d", &ch);
    switch (ch)
    {
      case 1 : p = l;
               printf ("Enter the element to be
                       inserted ::");
               scanf ("%d", &x);
               printf ("Enter the position of the
                       element ::");
               scanf ("%d", &pos);
```

```
for (i=1; i<pos; i++)
{
  p= p->next;
}
insert (x, 1, p);
break;
case 2:
p=1;
printf("Enter the element to be deleted ::");
scanf("%d",&x);
Delete (x,p);
break;
case 3:
p=1;
printf("Enter the element to be searched ::");
scanf("%d",&x);
p= Find (x,p);
if(p==NULL)
printf("Element does not exist !!! \n");
else
printf("Element exist !!! \n");
break;
case 4:
display (1);
break;
}
}while (ch<5);
}
```

Assignment - 7

Single linked List program

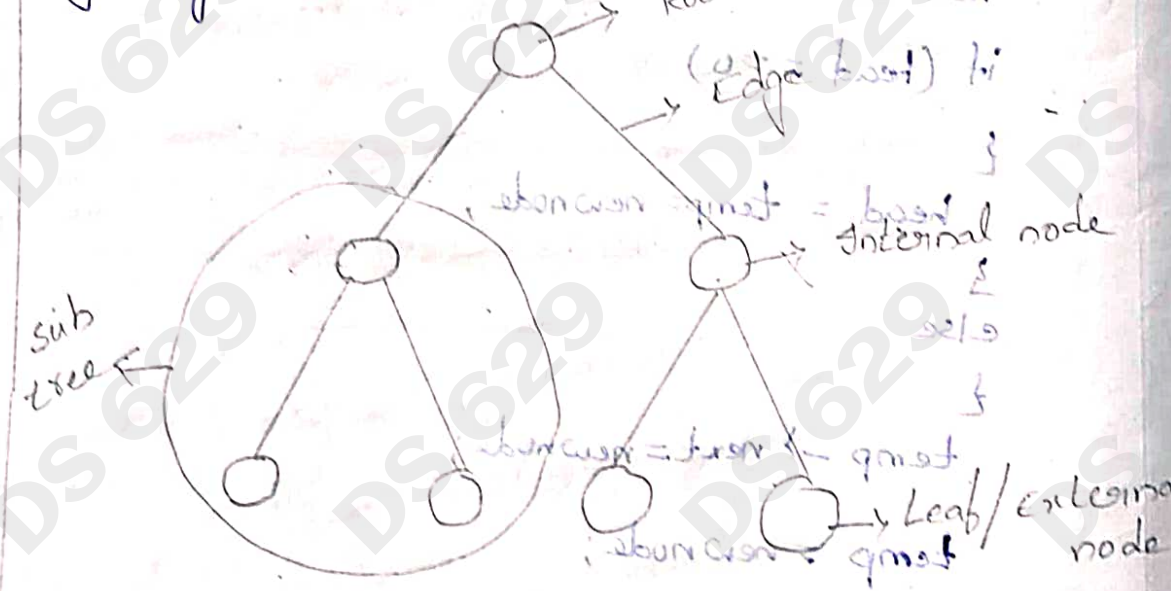
```
#include<stdio.h>
#include<conio.h>
void main()
{
  struct node
  {
    int data;
    struct node * next;
  }
  struct node * head, * newnode, * temp;
  head=0;
  int choice;
  {
  newnode=(struct node *)malloc(sizeof(struct node));
  printf("Enter data");
  scanf("%d", &newnode->data);
  newnode->next=0;
  if (head==0)
  {
    head = temp= newnode;
  }
  else
  {
    temp->next=newnode;
    temp = newnode;
  }
```

```
printf ("Do you want to continue (0,1) ? ");
scanf ("%d", & choice);
}
temp = head;
printf(" Enter data:");
scanf ("%d", & newnode -> data);
newnode -> next = temp -> next;
temp -> next = newnode;
}
getch();
}
```

NON LINEAR DATA STRUCTURES

Definition:-

A tree is a non-linear hierarchical data structures that consists of nodes connected by edges.



Terminology related to tree :-

Root :- It is the topmost node of a tree.

Node :- A node is an entity that contains a key or value and pointers to its child.

Internal Node :- The node having at least a childnode is called an internal node.

External Node :- The last nodes of each path are called leaf / External nodes. [that do not contain a link / pointer to child nodes.

Order of the tree :- The maximum number of branches (links) allowed in any node of a tree, is called the order of the tree.

Degree :- Maximum no. of ↑ childerens that is possible for a node is known as degree of a node.

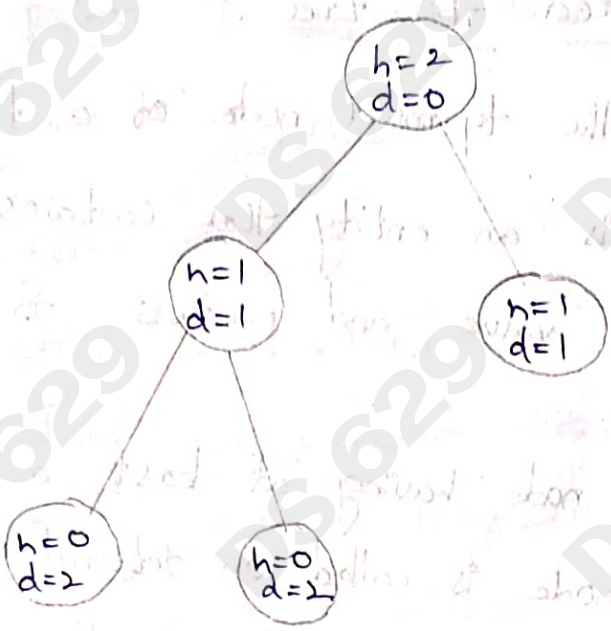
Parent & Children :-

Parent :- Any node expect the root node has one edge upward to a node called parent.

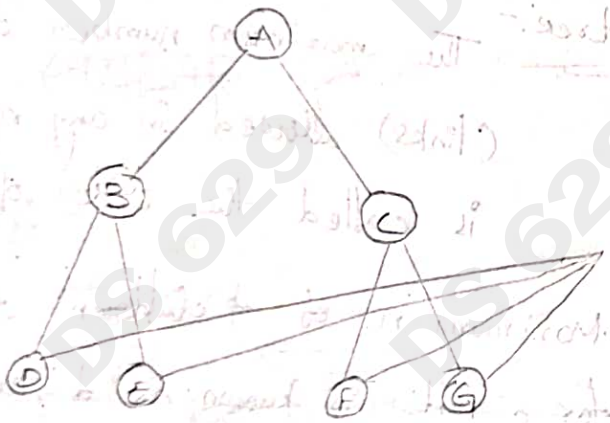
Children :- The node below a given node connected by its edge downward is called its children node.

Height / Depth of a tree :-

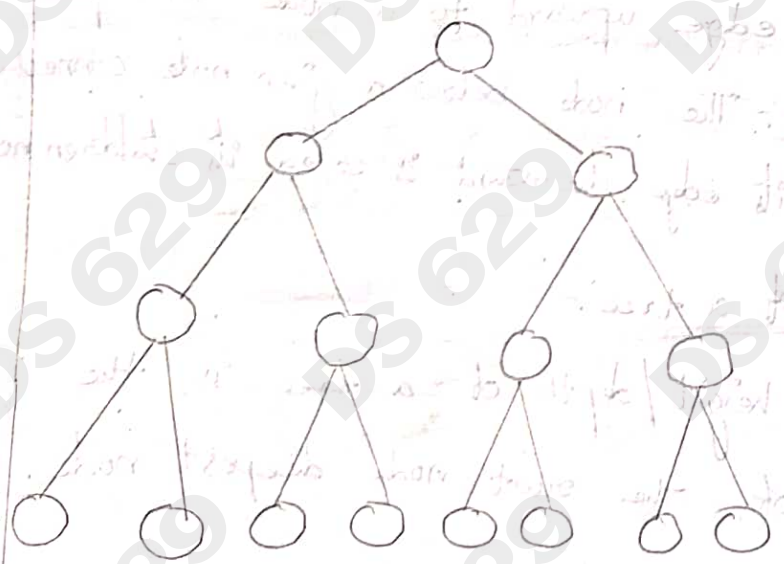
The height / depth of a tree is the height / depth of the root node deepest node.



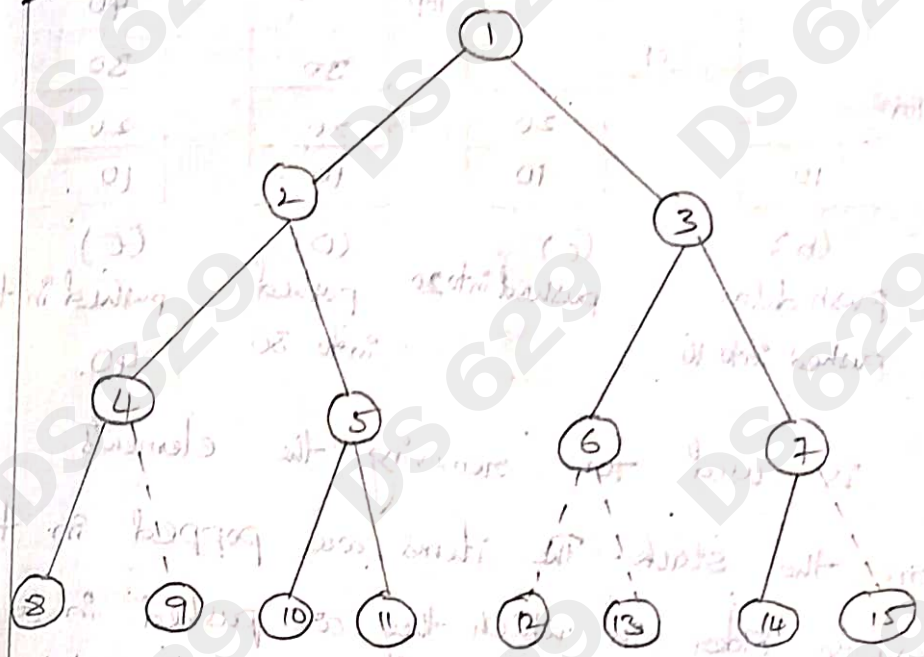
Siblings:- The children nodes of a given parent node are called siblings. They are also called brothers.



Binary tree:- A binary tree is a tree data structure in which each part node can have at most two children.



Linear representation of a binary tree:-

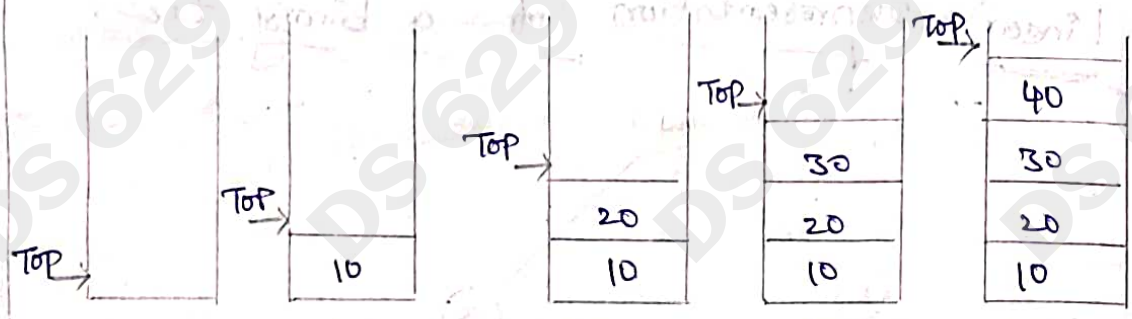


1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	1	1	1	1	1	0	1	1	0	0	1	0

Stack:- Stack is a linear data structure which allows a particular order in which the operations are performed. The order may be LIFO [Last-In-First-Order] manner.

We can perform two operations on stack. One is for adding elements in to ~~stand~~ stack and another one is for deleting the elements from the stack.

Push:- Adds an item in the stacks. ~~If is~~ st If the stack is full, then it is said to be an overflow stack.



(a) empty stack

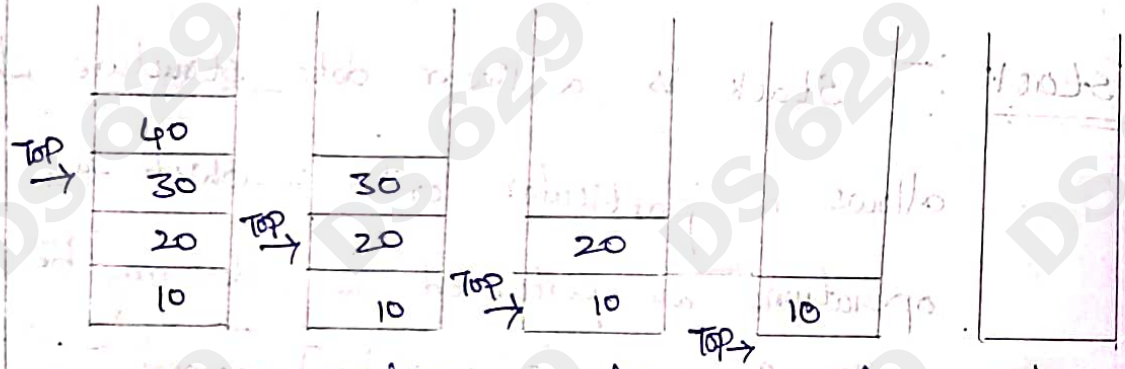
(b) Push data... pushed into 10

(c) pushed into 20

(d) pushed in to 30

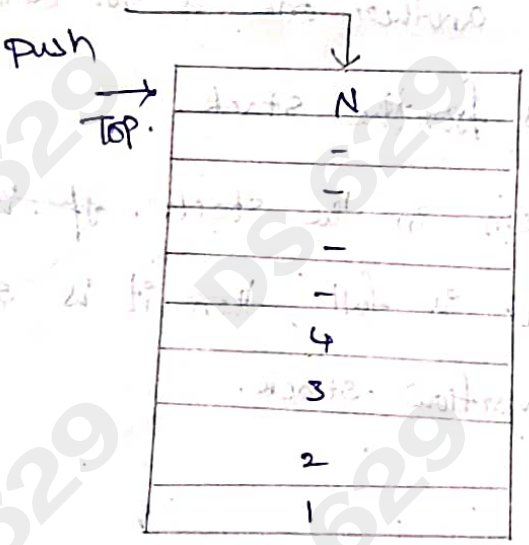
(e) pushed in to 40.

Pop:- It is used for removing the elements from the stack. The items are popped in the reverse order in which they are pushed in to the stack. If the stack is empty in the pop operation then it is said to be an under flow condition:

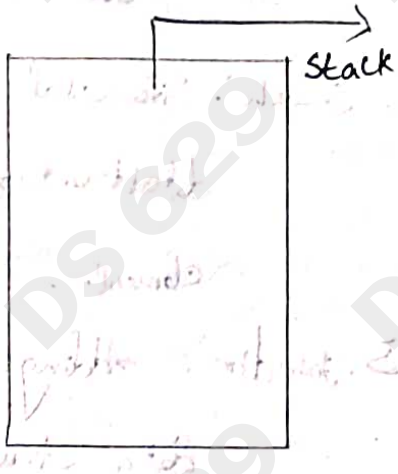


popped:40 popped:30 popped:20 popped:10 underflow.

Overflow:- When a stack size is full then it is shown stack over flow condition.



Underflow :- When the stack is empty, but we can perform again pop operation in to stack then that is said stack underflow condition.



Advantages of stack:-

* Lightening fast
* Auto sharding
* Replication is very easy.
* You can perform rich queries, can create on the fly indexes with a single command.

Disadvantages of stack:-

* Very unreliable
* Indexes take up a lot of RAM.
* Binary tree indexes

Operations on Data structures?

1. Traversal: Travel through the data structure.

2. Search: Traversal through the datastructure for a given element.

3. Insertion:- Adding new elements to the data structure.

4. Deletion:- Removing an element from the data structure.

5. Sorting:- Arranging the elements in some types of order.

6. Merging:- Combining two similar data structures into one.

Space Complexity:- By space complexity we mean the amount of memory it needed to run to completion. It consists of the instruction space, data space.

Time Complexity:- The time complexity of an algorithm is the amount of time it needs to run to completion.

Data type:- A data type is a type of data. Some common data types include integers, floating point numbers, character, strings and arrays. They may also be more specific types such as dates, time stamps, boolean values and varchar formats.

Abstract data type:- An abstract data type is an abstract concept defined by axioms which represent some data and operations on that data. Abstract data types are focused on what, not how. They framed decoratively and do not specify algorithms or data structures.

Algorithm Analysis:- There can be more than one solution to a problem and developing an efficient algorithm requires lot of practice and skill. When we have two or more algorithms for a given solution then it is necessary to compare them and choose better one. This is one of the most important motivation to Analyze an algorithm.