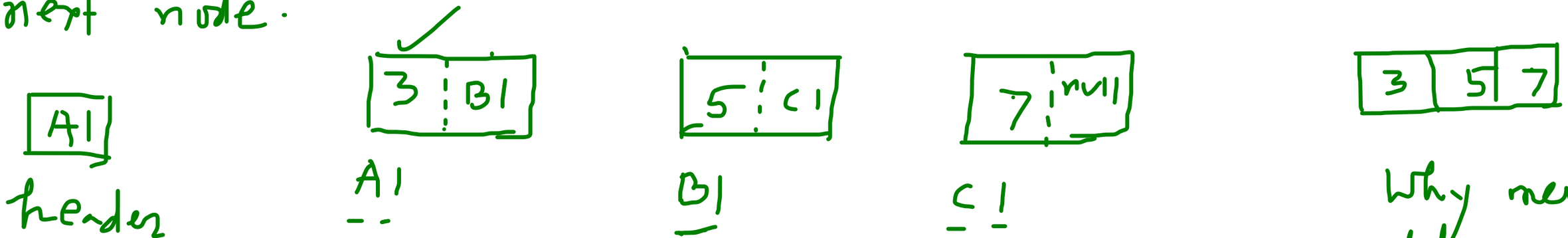
Linked list.

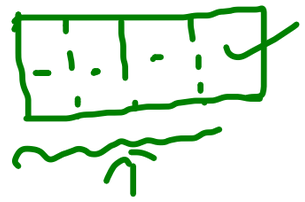
$5\times8^2+2\times8^1+5\times8^0 \quad (525)_8 \rightarrow (341)_{10}$

Linked list is a linear data structure where each node contains data/info and link for the next node.



Linked list is purely a dynamic data structure which is considered as a fundamental data structure.

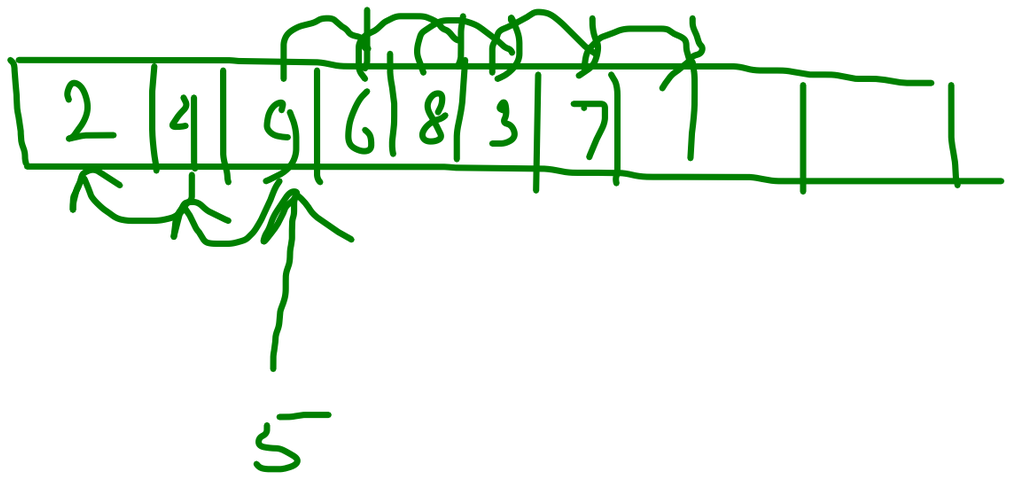
Comparison between array and linked list



1) Linked list is a homogeneous / heterogeneous data structure which can accomodate same or different data types together to create a new datatype where as array is homogeneous data structure in most of the language.

2) Linked list is dynamic data structure where an array is a static data structure.

3) Insertion and deletion of nodes in Linked list is faster than array.

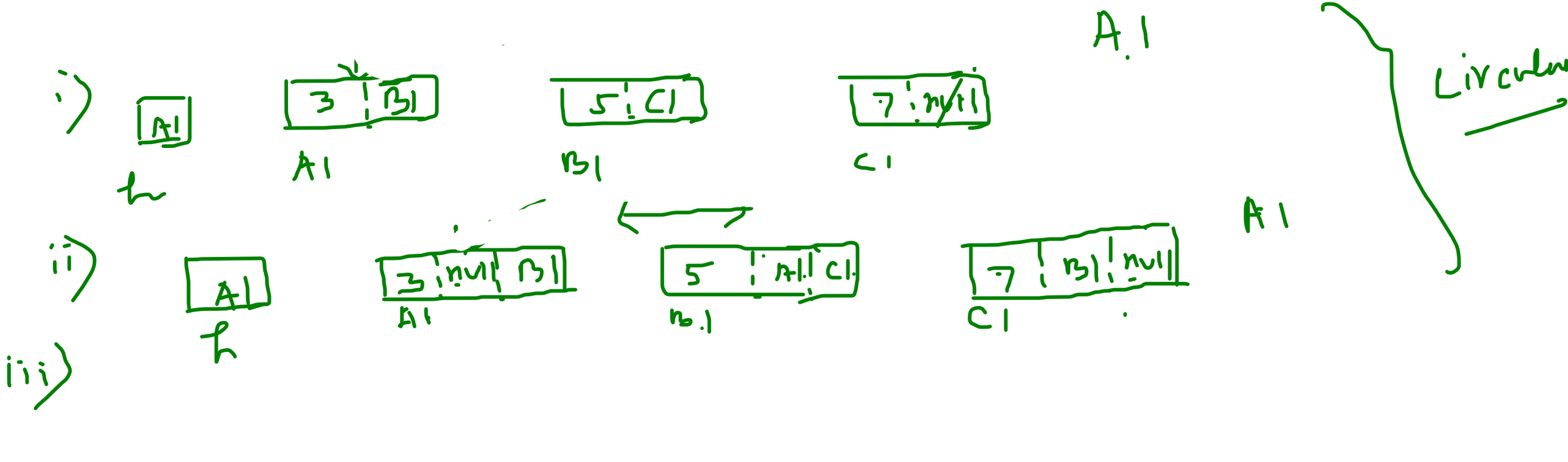


Types of Linked list :

i) Singly Linked list

ii) Doubly " "

iii) Circular " "

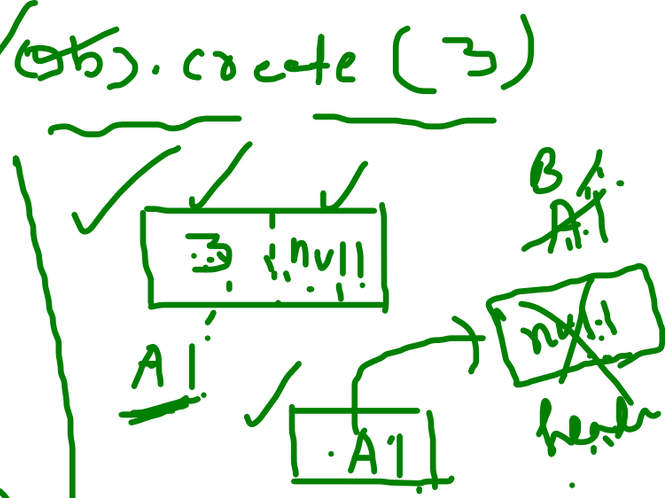


Implementation of Linked list coding

```
class Node
{
    int data;
    Node link;

    Node (int d)
    {
        data = d;
        link = null;
    }
}
```


```
class LinkedList
{
    Node header;

    void create (int d)
    {
        Node newnode = new Node (d);
        header = newnode;
    }

    void insertFirst (int d)
    {
        Node newnode = new Node (d);
        newnode.link = header;
        header = newnode;
    }
}
```

Obj.create (3)



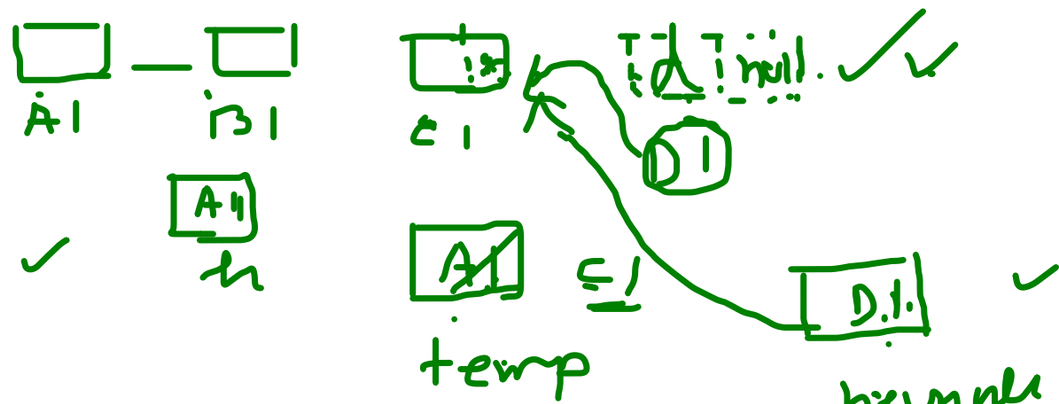
```
void deleteFirst()
{   S.o.p("Deleted node" + header.data);
    header = header.link;
}

void display()
{   Node temp = header;
    while (temp != null)
    {   S.o.p(temp.data);
        temp = temp.link;
    }
}
```


```
int countNodes()
{
    - - - - - - -
    - - - - -
}
```

```
int countNodes()
{
  int c=0;
  Node temp = head;
  while (temp != null)
  {
    c++;
    temp = temp.link;
  }
  return c;
}
```

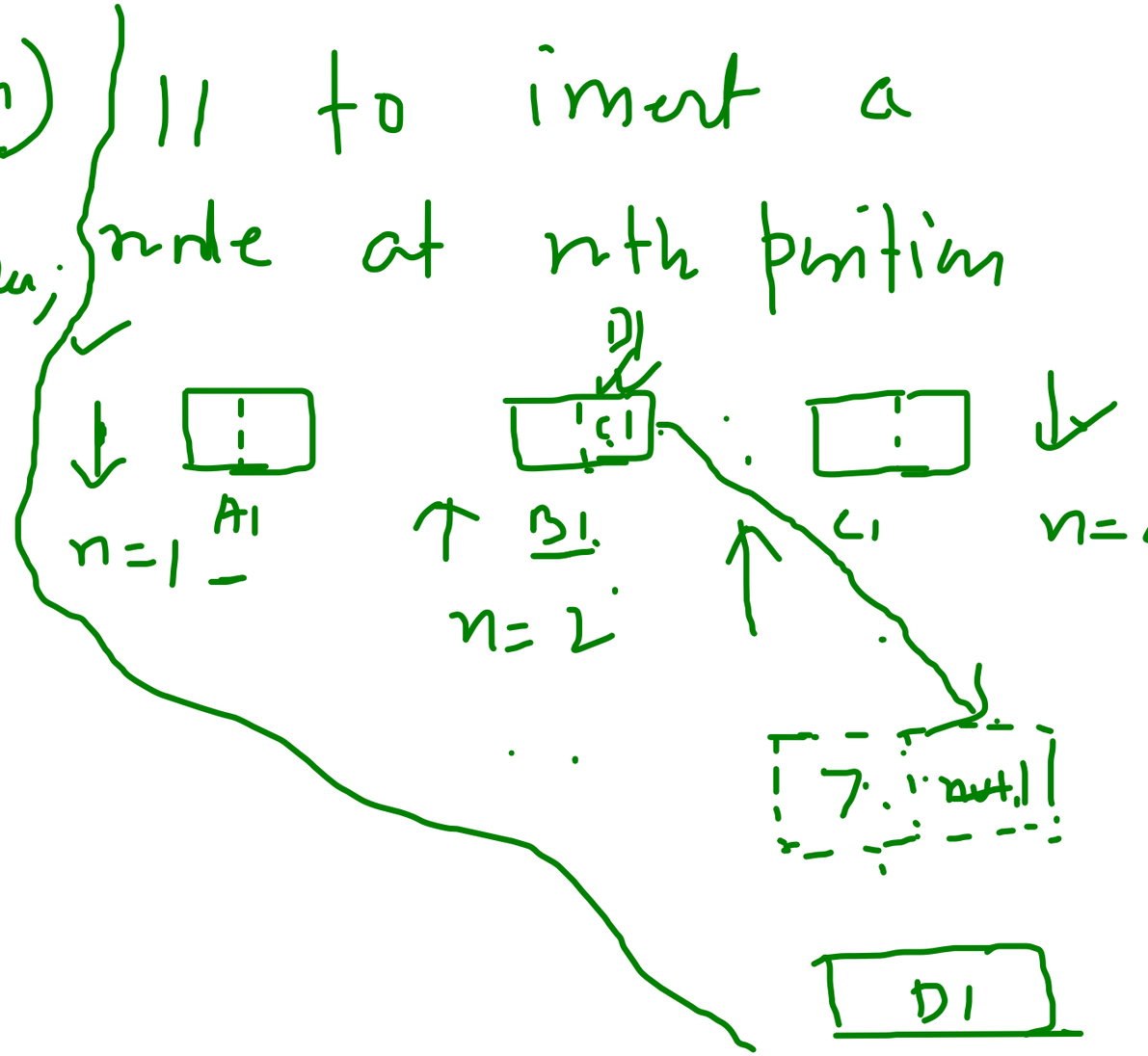
```
void countprime()
{
  Node temp = head;
  int c = 0;
  while (temp != null)
  {
    if (isprime(temp.data) == true)
      c++;
  }
  temp = temp.link;
  S.o.P(c);
}
```

```
void appendNode(int d)
{
    Node newnode = new Node(d);
    Node temp = header;
    while (temp.link != null)
        temp = temp.link;
    temp.link = newnode;
}
```



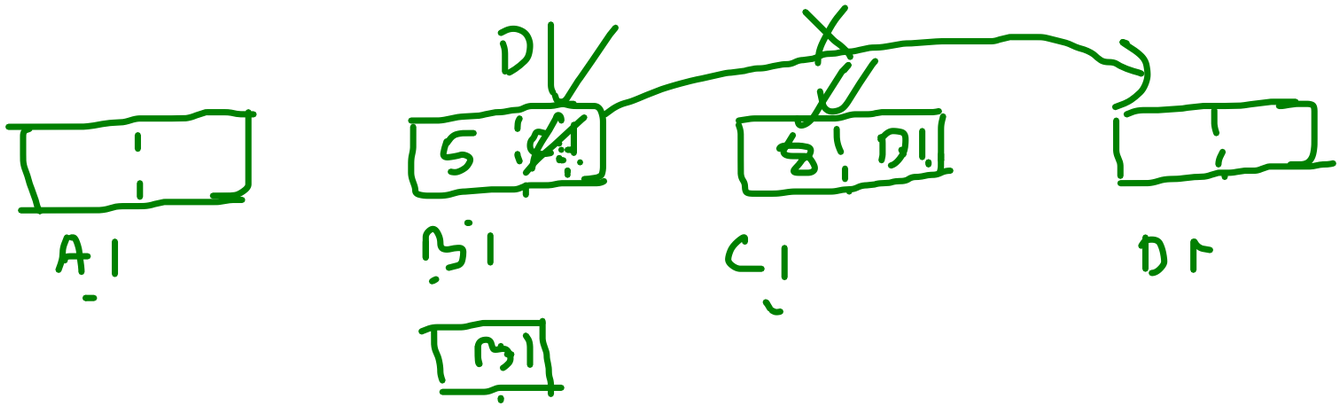
```
void insertnth(int d, int n)
{  int i      ;        Node temp = header;
   Node newnode = new
                  Node(d);
   for (i = 1 ; i <= n-2; i++)
          temp = temp.link;
   newnode.link = temp.link;
   temp.link = newnode;
}
```

// to insert a node at nth position



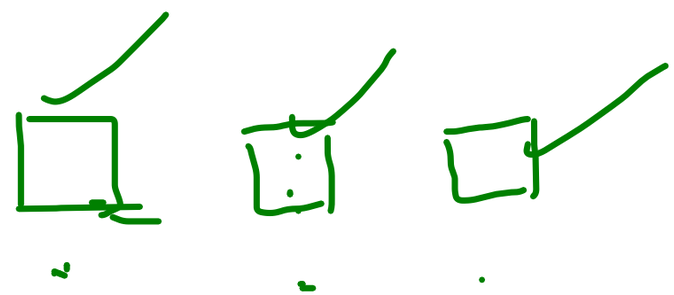
```
void deleteNth(int n)
{
  Node temp = header;
  for (int i = 1; i <= n-2; i++)
     temp = temp.link;
  S.op("Deleted node =" + temp.link.data);
  temp.link = temp.link.link;
}
```

n=3

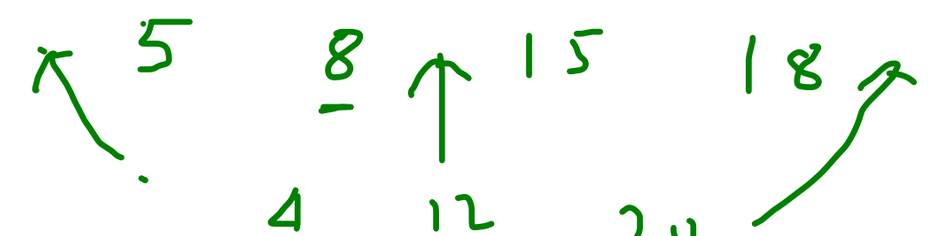


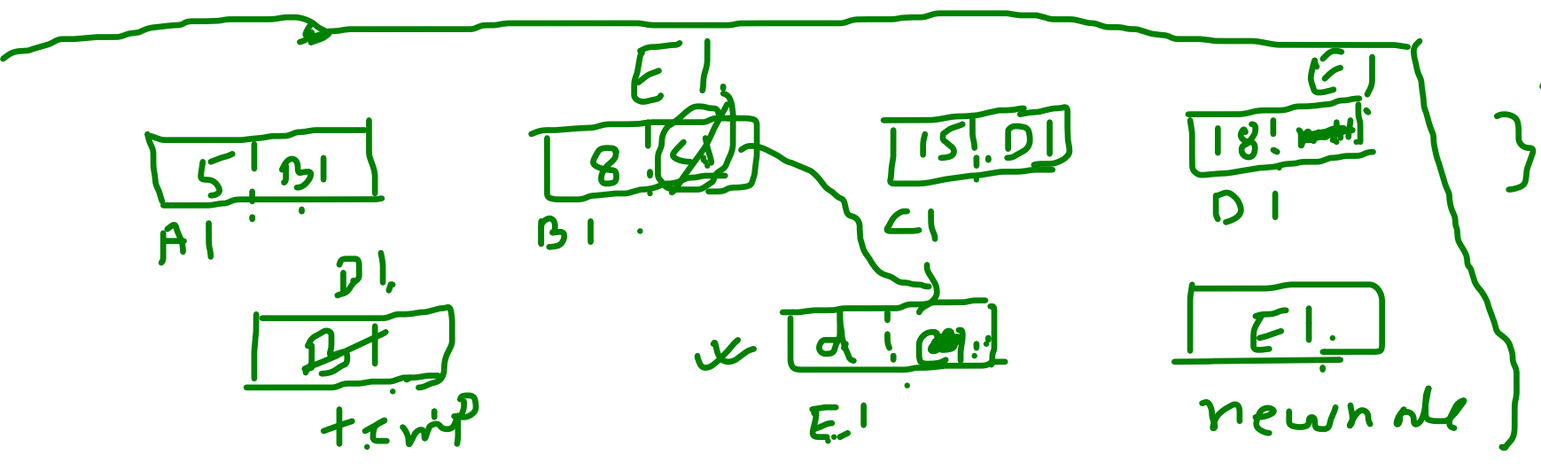
```
void displayReverse( )
{  int i, j, n = countNode( ) ;
   for (i= n. ; i >= 1 ; i--)
  {  Node temp = header ;
     for (j = 1 ; j < i ; j + +)
          temp= temp. link ;
     S.o ptln ( temp . data ) ;
  }
}
```

n = 3



```
void insertAscending(int d)
{
  Node newnode = new Node(d);
  Node temp = header;
  if (temp.data > d)
  {
     insertFirst(d);
     return;
  }
  while (temp.link != null)
  {
    if(d)temp.data <= d <= temp.link.data)
    {
      break;
    }
    temp = temp.link;
  }
  newnode.link = temp.link;
  temp.link = newnode
}
```





```
void splitList (int n) // to split a list from nth position.
{ Node temp = header;
  for (int i = 1; i <= n-2; i++)
      temp = temp.link;
Node
  header1 = temp.link;
  temp.link = null;
}
```

5 4 9 7 8 3

n = 4

5 4 9 ✓

7 8 3 ✓

```
void searchNode ( int num) // Linear Search
{   int i, n = countNode ( ) ;        if found display the
   Node temp = header ;                  position, otherwise display
 for (i= 1 ; i <= n ; i++)                error message.
{   if (temp.data == num)
   {
   S.o.ptln ("Found in position " + i );
        return ;
   }
else
    temp = temp.link ;
 }
   S.o.ptln (" Not Found in the List ");   }
```

```
void bubblesort( )
{  int i, j, n = countNodes( ) ;
   Node temp = header ;
for ( i = 1 ; i < n ; i ++ )
{      temp = header ;
    for ( j = 1 ; j < n - i ; j ++ )
    {     if (temp . data > temp.link. data )
                { int t = temp . data ;
                temp . data = temp . link . data ;
                }. temp . link . data = t ;
temp = temp. link ;
}
                                        display ( ) ;
                                        }
```

→ 4 6 4 6 2 5

WAP to multiply two matrix Test on Link List

↓

8th May

void main (int a[][], int b[][])

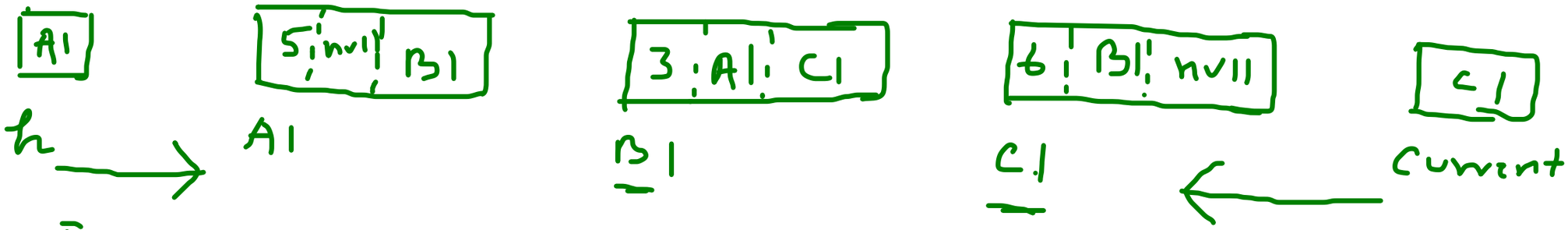
m×n n×p

r1, c1 r2, c2

r1, c2

```
for (i=0; i<m; i++)
{
    for (j=0; j<p; j++)
    {
        sum = 0
        for (k=0; k<n; k++)
            sum += ----- * ---
        c[i][j] = sum;
}
```

Doubly linked list.



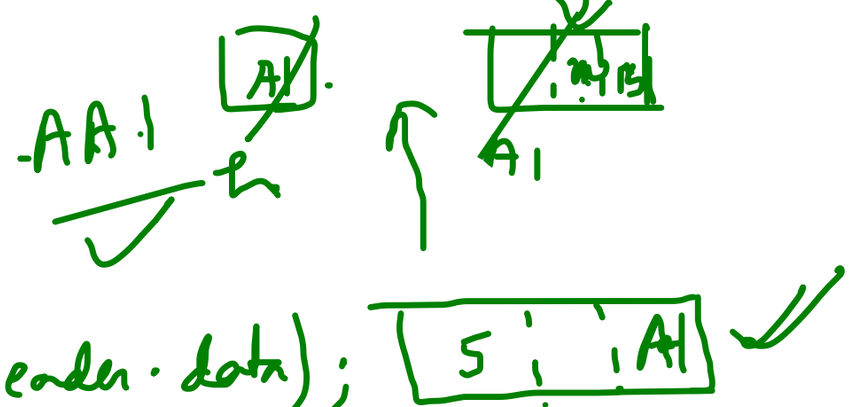
In a doubly linked list there are two pointers to store the address of the previous and next node. Hence traversal in both direction i.e start to end and vice versa is possible.

```
class Dnode
{  int data
   Dnode prev, next ;
   Dnode (int d)
{   data = d ;
    prev = next = null ;
}
}
```

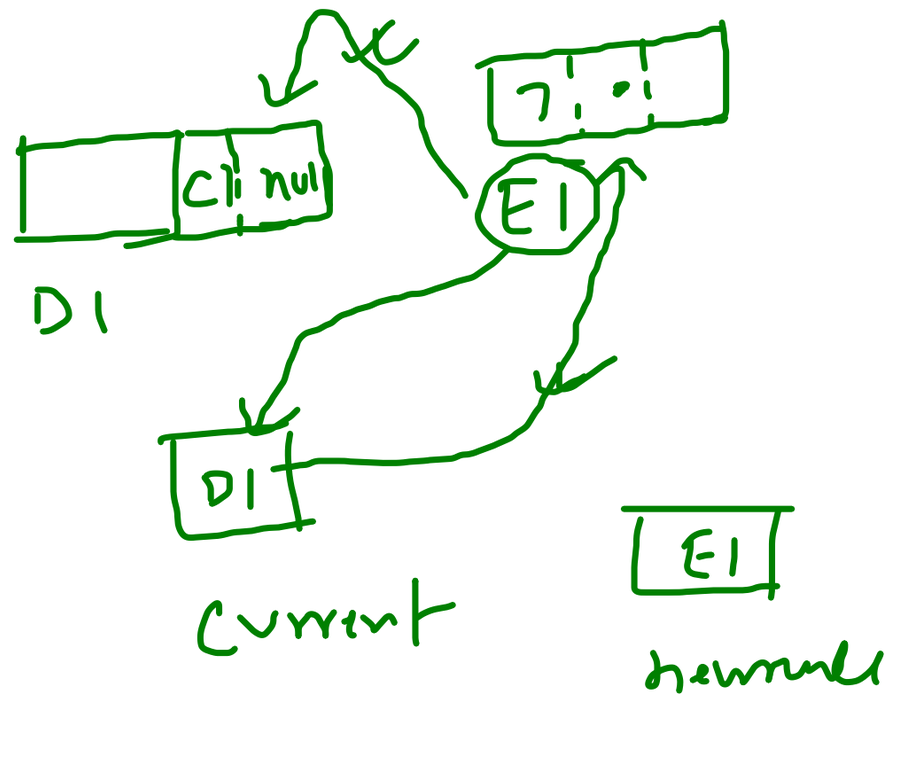
```
class Dlinkedlist
{  Dnode header, current ;

   void createfirst (int d)
{  Dnode newnode =
       new Dnode (d) ;  →
   header = current = →
                    newnode ;
}
```

```
void insertFirst(int d)
{
  Dnode newnode = new Dnode(d);
  header.prev = newnode;
  newnode.next = header;
  header = newnode;
}

void deleteFirst()
{ S.O.ln("Deleted node" + header.data);
  header = header.next;
  header.prev = null;
}
```

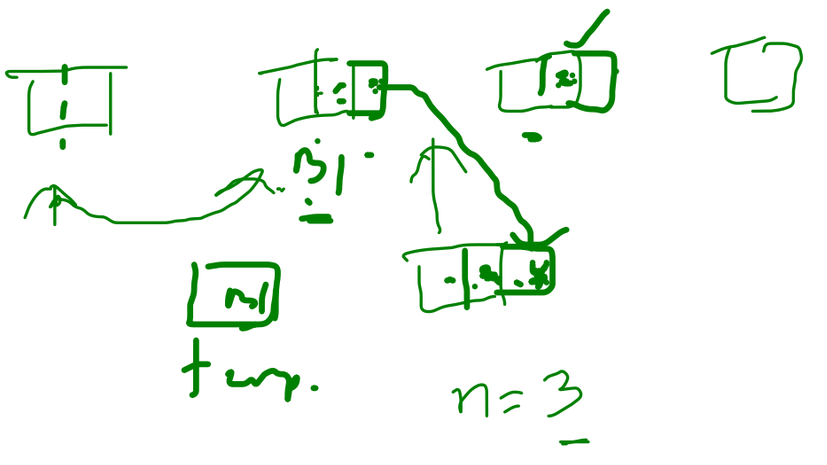
```
void append(int
  d)
{
}
```



```
void appendNode(int d)
{
Dnode newnode = new Dnode(d);
newnode.prev = current;
current.next = newnode;
current = newnode;
}
void displayReverse()
{
   Dnode temp = current;
  while(temp != null)
 { S.o.p(temp.data);
  temp = temp.prev;
 }
}
```



```
void deleteLast()
{
    S.o.println("Deleted node" + current.data);
    current = current.prev;
    current.next = null;
}
void insertnth(int n, int d)
{
    Dnode newnode = new Dnode(d);
    Dnode temp = header;
    for(int i = 1; i <= n-2; i++)
        temp = temp.next;
    newnode.next = temp.next;
    newnode.prev = temp;
    temp.next.prev = newnode;
    temp.next = newnode;
}
```



Recursion in linked list:

```
int countNodes(Node h)
{   if(h == null)
        return 0;
    else
        return 1 + countNodes(h.link);
}
```

return 1 + 2

return 1 + 1

return 1 + 0.

```
void display(Node h)
{   if(h==null)
        return;
    else { s.o.p(h.data);
           display(h.link); }
}
```

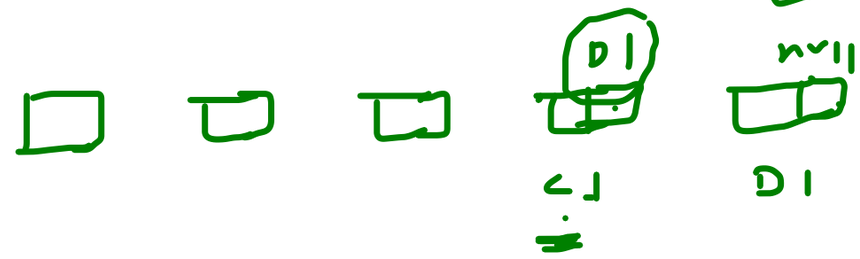
```
void searchNode(int item, Node h)
{
  if (h == null)
      s.o.p("Not found");
  else if (h.data == item)
      s.o.p("Found");
  else
      searchNode(item, h.link);
}
```

```
int countprime(Node h)
{
  if (h == null)
      return 0;
  else if (isprime(h.data))
return 1 + countprime(h.link);
  else
     return countprime(h.link);
}
```

```
boolean isprime(int n)
{

}
```

```
void appendNode (Node h, int data)
{   if(h.link == null)
    {   Node newnode = new Node (data);
        h.link= newnode ;
    }
else
        appendNode (h.link, data);        ✓
}
```

```
void deleteLast (Node h)
{
  if (h.link.link == null)
   { S.o.p(" Deleted node = " + h.link.data);
     h.link = null;
   }
  else
     deleteLast (h.link);
}
```



Revision

1) using NOR construct NAND

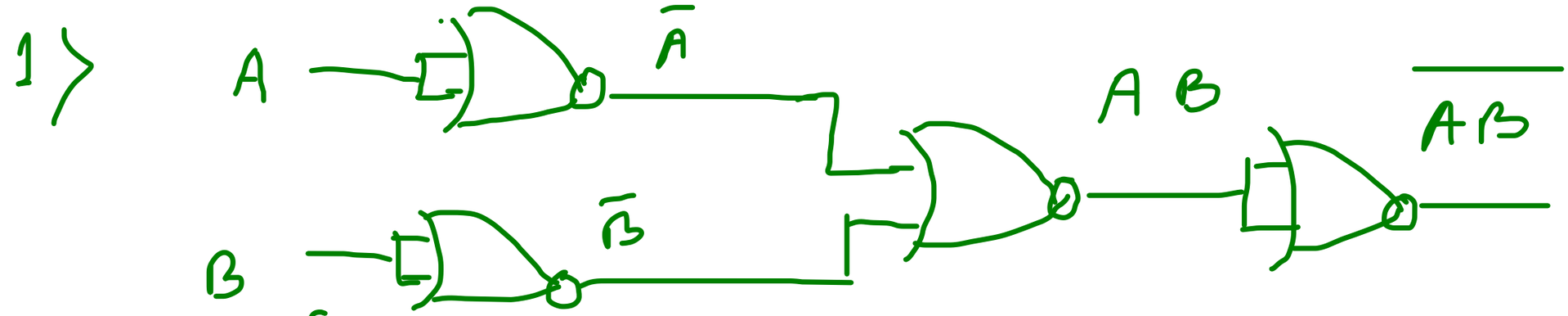
2) $F(A, B, C, D) = \Pi(0, 2, 3, 4, 5, 8, 9, 11, 12, 13)$

3) State and prove law of complement

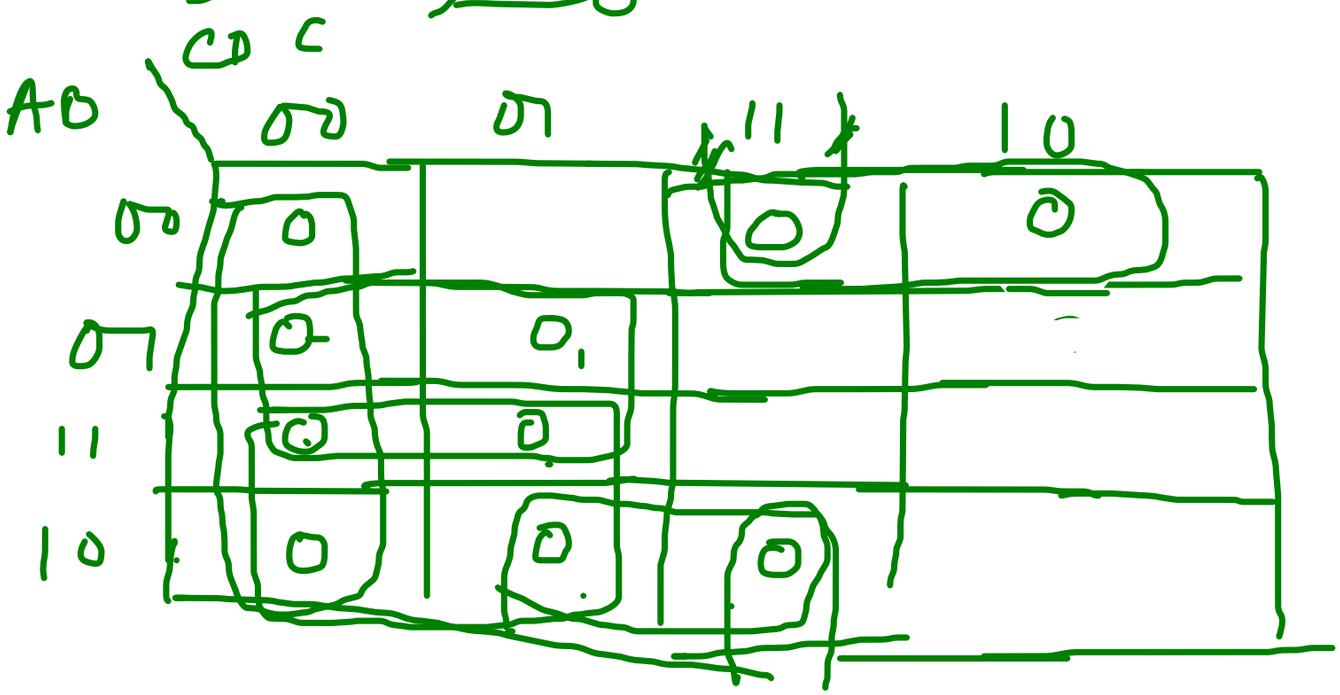
4) Draw a decoder circuit to convert a binary to octal

5) prove $\Sigma 0, 2, 3, 4 = \pi(1, 5, 6, 7)$

1)



2)



2 Quad, 2 pair

3) $A + A' = 1$

L.H.S $A + A'$

$= A + A' \cdot 1$ [Identity law]

$= A + 1$ [2nd complement law]

$= 1$ [Law of unity]

$=$ R.H.S.

L.H.S = $\sum$ 0, 2, 3, 4

$A+B = (\overline{A} \cdot \overline{B})'$

$f = m_0 + m_2 + m_3 + m_4$

$f' = m_1 + m_5 + m_6 + m_7$ [$f' = U - f$]

$= (\overline{m_1} \cdot \overline{m_5} \cdot \overline{m_6} \cdot \overline{m_7})'$ [Apply De Morgan's law]

$\therefore (M_1 \cdot M_5 \cdot M_6 \cdot M_7)'$ [minterm + Maxterm = 1]

Complementing both side

$f = M_1 \cdot M_5 \cdot M_6 \cdot M_7$

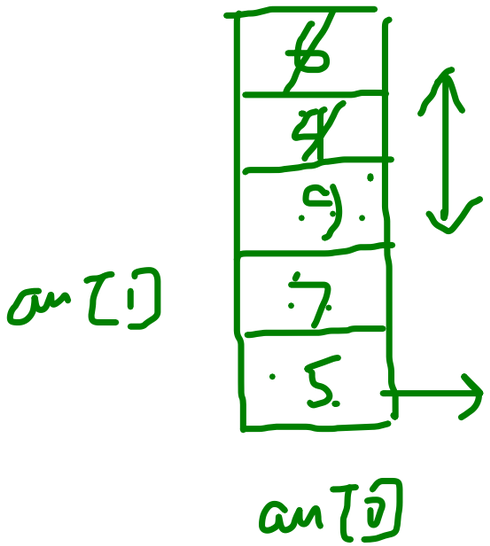
$= \pi(1, 5, 6, 7)$: R.H.S proved.

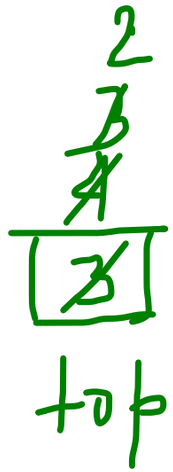
Stack.

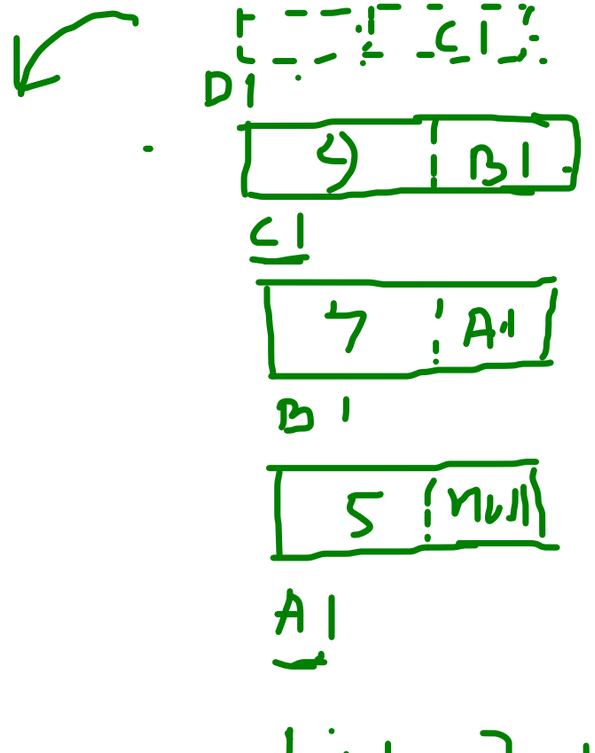
Def: Stack is a linear data structure where insertion and deletion of nodes take place only at one end, known as top of the stack.

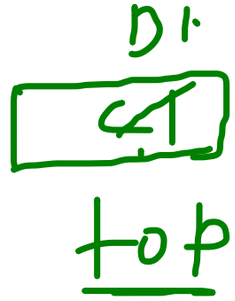
It is a type of LIFO (Last in first out) type of data structure.

Stack can be implemented using array and Linked list.









Array implementation of stack-

Linked list implementation of stack-

Application of stack.

1) To store return address during function call

2) To implemint undo/Redo in word processor.

3) To convert infix expression to postfix (*) → a a 2) a)

4) To evaluate postfix expression (*)

```
class Stack.
{  int arr[], size, top ;
  Stack (int n)
{  arr = new int[n];
  size = n;
   top = -1;
}
  void push (int item)
{   if (top == size - 1)
     System.out.println ("overflow");
else
}arr[++top] = item;
```

```
int pop ( )
{ if (top == -1)
    return -9999 ;
  else
   return arr[top--] ;
  }
{ int peek ( )
  { if (top == -1) return -999;
    else return arr[top] ; }
```

```
boolean isEmpty ()
{
    return (top == -1) ;
}

boolean isFull ()
{
   return (top == size - 1);
}
```

```
void main ()
{
Stack obj = new Stack(5);
obj.push (3);
obj.push (8);
obj.push (10);
System.out (obj.pop ());
```

Stack.

1. push
2. pop
3. peek
4. Display
5. Exit.