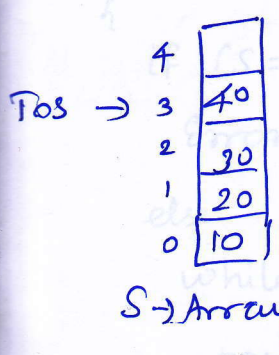
```
void pop(Stack S)
{
  if (IsEmpty(S))
    Error("Empty Stack");
  else
    S->TopofStack--;
}
```

* It decrements topofstack so that it points to the next topmost element in the stack.

c) TopandPop - It returns the top element and deletes the same from the stack.

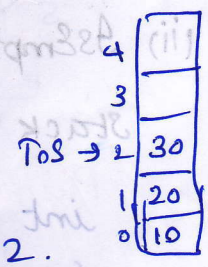
```
ElementType TopandPop(Stack S)
{
  if (!IsEmpty(S))
    return S->Array[S->TopofStack--];
  Error("Empty Stack");
  return 0;
}
```



return S->Array[Tos--]

S->Array[3] => 40.

Tos-- (ie) TopofStack = 2.



returns 40 & removes it. TopofStack is 2.

2 Linked List Implementation of Stack:

* Single Linked List is used to implement Stack Push is performed by inserting an element at front of the list and pop is performed by deleting the element at the front of the list. Top examines the element at the front of the list, returning its value.

Algorithm:

ii) Type declaration:

```
struct node;
typedef struct node * PtrToNode;
typedef PtrToNode Stack, Position;
int IsEmpty (Stack S);
Stack CreateStack (Void);
void DisposeStack (Stack S);
void MakeEmpty (Stack S);
void Push (ElementType X, Stack S);
ElementType Top (Stack S);
void pop (Stack S);
```

Structure:

```
struct node
{
  ElementType Element;
  Position next;
};
```

(iii) IsEmpty - This routine is used to check whether the Stack is empty or not.

```
int IsEmpty (Stack S)
{
  return S->next == NULL;
}
```

* Stack S contains header node's address. The IsEmpty function returns true (1) value if header's next is NULL else returns false (0) value.

(iii) CreateStack - This routine is used to create a stack. It creates a header node for stack and makes the header's next pointer to be NULL (ie) stack empty.

```
Stack CreateStack()
{
 Stack S;
 S = malloc(sizeof(struct node));
 if (S==NULL)
   FatalError("out of space");
 S->next = NULL;
 return S;
}
```

(iv) MakeEmpty - This routine is used to make the stack empty. It removes all the elements from the stack using pop and thus makes header node's next pointer to be NULL.

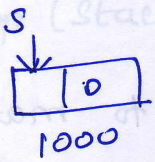
```
void MakeEmpty(Stack S)
{
  if (S==NULL)
    Error("Must create the stack first");
  else
    while(!IsEmpty(S))
      pop(S);
}
```

(v) Push - This routine is used to push an element at the top of the stack.

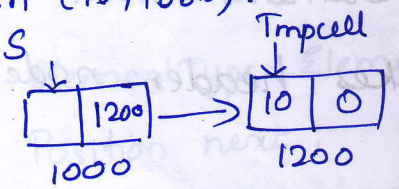
```
void push(ElementType X, Stack S)
{
```

```
TmpCell = malloc (sizeof (struct node));
if (TmpCell == NULL)
    FatalError ("Out of Space");
else
{
    TmpCell -> Element = x;
    TmpCell -> next = S -> next;
    S -> next = TmpCell;
}
}
```

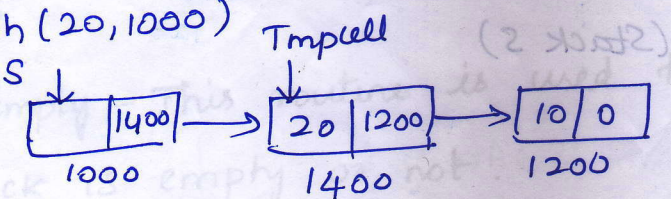
eg. Initially, by CreateStack, a header node gets created and its next is NULL.



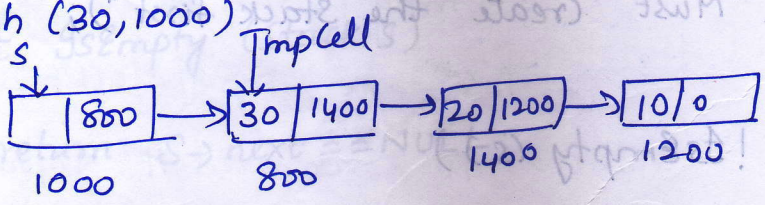
Push (10, 1000).



Push (20, 1000)



Push (30, 1000)



(vi) Top - The routine is used to return the top element in the stack.

```
ElementType Top (Stack S)
{
    if (!IsEmpty (S))
        return S -> next -> element;
    Error ("Empty Stack");
```

Top(1000) ⇒ returns ~~1000 → element~~

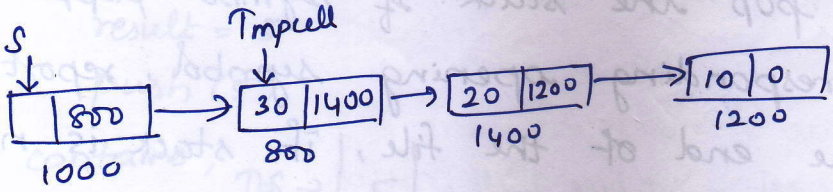
1000 → next → element

800 → element (ie) 30.

(vii) Pop - The routine is used to remove the topmost element from ~~the~~ the stack

```
void pop (Stack S)
{
  Position TmpCell;
  if (IsEmpty (S))
    Error ("Stack is empty");
  else
    {
      TmpCell = S→next;
      S→next = TmpCell→next;
      free (TmpCell);
    }
}
```

eg.



Pop(1000) ⇒ Tmpcell = S→next

(ie) TmpCell = 800

S→next = Tmpcell→next

(ie) S→next = 1400

free (800)