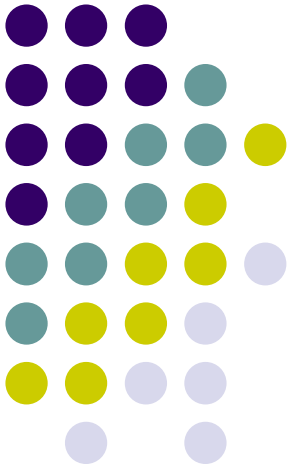
UNIT - 2

LINEAR DATA STRUCTURES

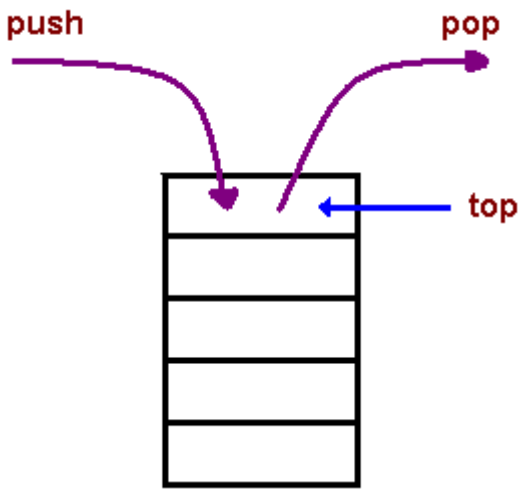


Stack ADT

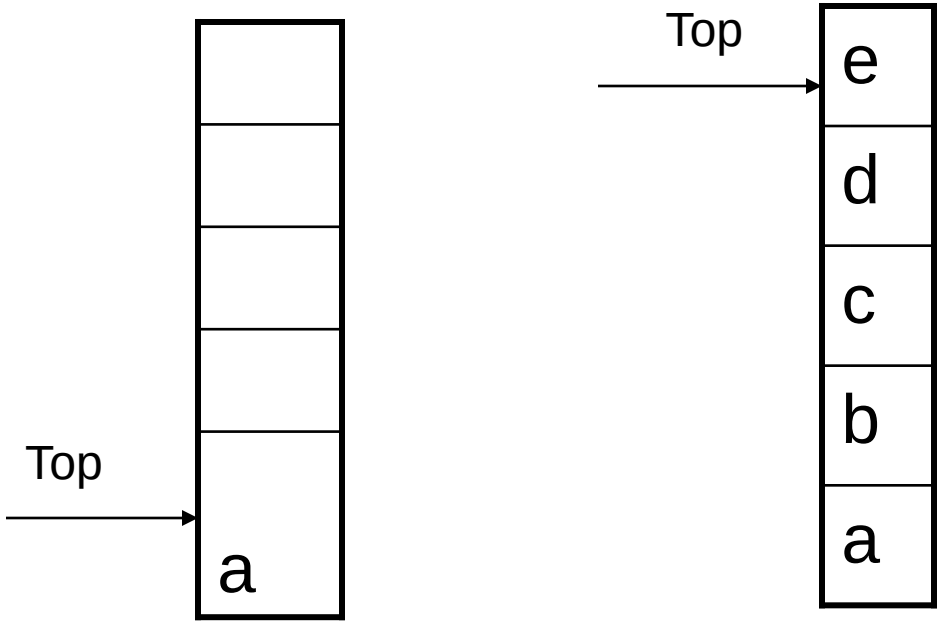
- A Stack is a linear data structure where the collection of items are accessed by Last In First Out (LIFO) or First In Last Out (FILO) order.
- A Stack represents an ordered collection of homogeneous (same data type) elements.
- Both insertion , deletion operations can be done only at one end (last element), called the top of the stack.
- We can implement Stack using Array or Linked List.
- Stack also referred as “Push-down lists”.
- Real-Time Examples of Stack :

  - Car Parking
  - Pile of Coins
  - Stack of trays
  - Pile of Notebooks
  - Coaches of Train

Representation of Stack Model:



- The first element placed in the stack will be the bottom of the stack.
- The last element placed in the stack will be at the top of the stack.
- The last element added to the stack is the first element to be popped out.
- Hence stacks are referred to as Last In First Out or First In Last Out.



Exceptional Conditions:

Overflow Condition:

- An attempt to insert an element when the stack is full is said to be Overflow and the new can not be pushed.

Underflow Condition:

- An attempt to delete an element when the stack is empty is said to be Underflow and

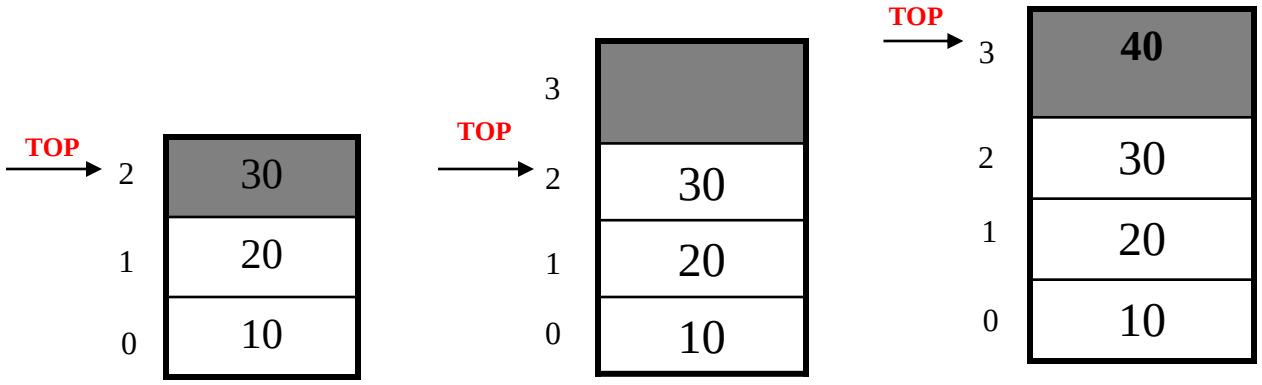
Ways of Implementation of Stack :

- → Array Implementation
- → Linked List Implementation

Implementation of Stack

- A stack represents an ordered collection of homogeneous elements.
- It can be implemented using arrays / linked list.
- Arrays – Better implementation when the number of elements are known.
- Linked list – Better implementation when the number of elements are not known.

Array Implementation of Stack:

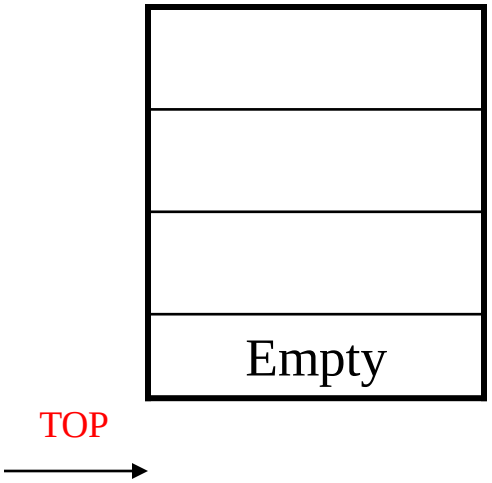


Array Implementation of Stack (Contd.,)

- Create (S) – Create an empty Stack
- Push (S, element) – Inserts an element
- Pop (S) – Deletes an element
- ShowTop (S) – Returns the Top element

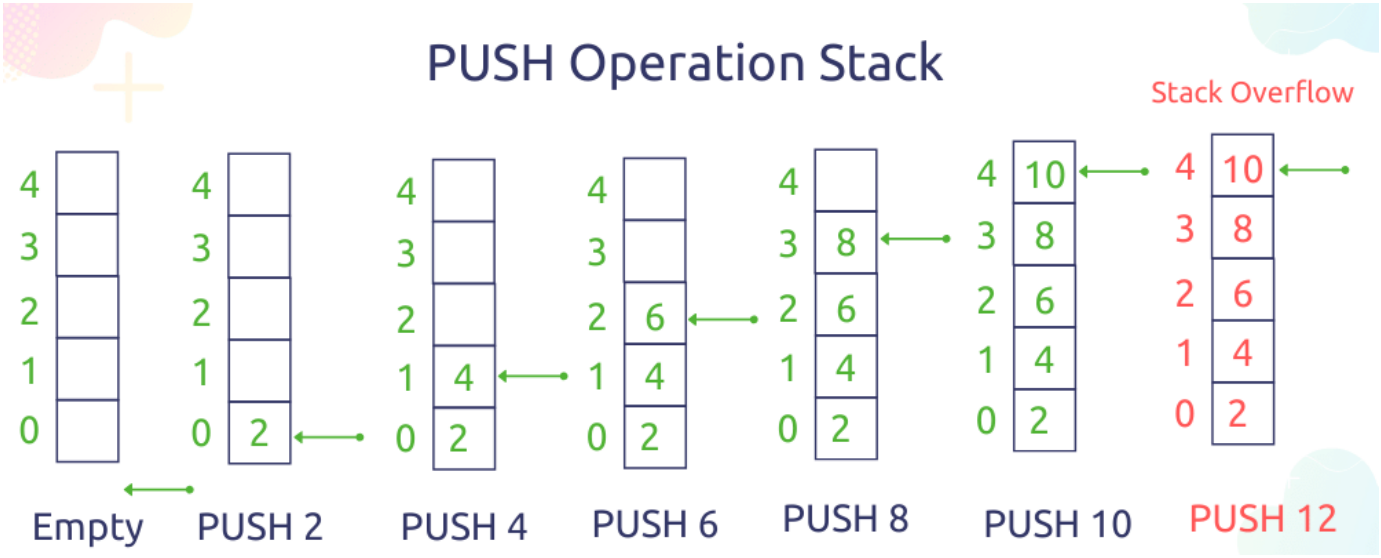
Create (S):

```
void Create (STACK *S)
 {
  S[TOP] = -1;     // S→Top
 }
```



- → Create an empty stack.
- → Initializes the space to be used for elements to be pushed.
- → Allocation of space can be Static using arrays or Dynamic using linked list.
- → As all operations are performed on the top element, we need to have the index or the pointer to it (Top pointer).
- → Top pointer should be properly initialized to denote an empty stack.

Push (S, Element):



Algorithm:

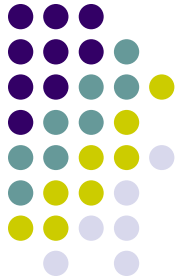
Step 1: Check whether the Stack is full or not. If yes, print the message that the Stack is full. i.e., Overflow. Then the Push operation can not be performed.

Step 2: Else,

i) **TOP** pointer is incremented by 1.

ii) new element is pushed at the position **TOP.**

Routine for Push operation:

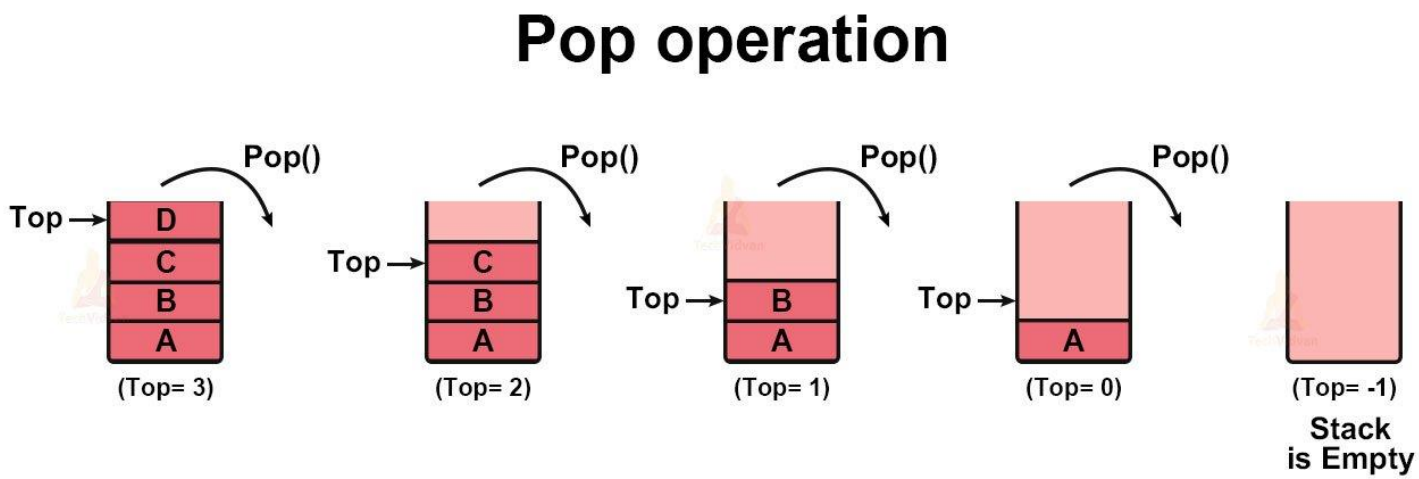


```
#define MAX 50
int TOP = -1;
int stack [MAX];

void Push (int data)
{
 if(TOP = =MAX-1)
{
 printf("Stack Overflow");
 return;
 }
 else
{
  // Push an element into a stack

  ++TOP;
  printf("\n Enter the element to be pushed");
  scanf("%d", &data);
  stack [TOP] = data;
 }
}
```

Pop (S):



Algorithm:

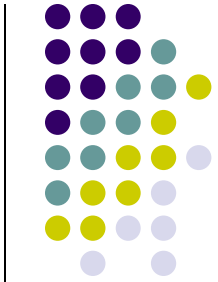
Step 1: Check whether the Stack is empty or not. If yes, print the message that the stack is empty. i.e., Underflow. Then the pop operation can not be performed.

Step 2: Else,

i) The element at the position of **TOP** is deleted.

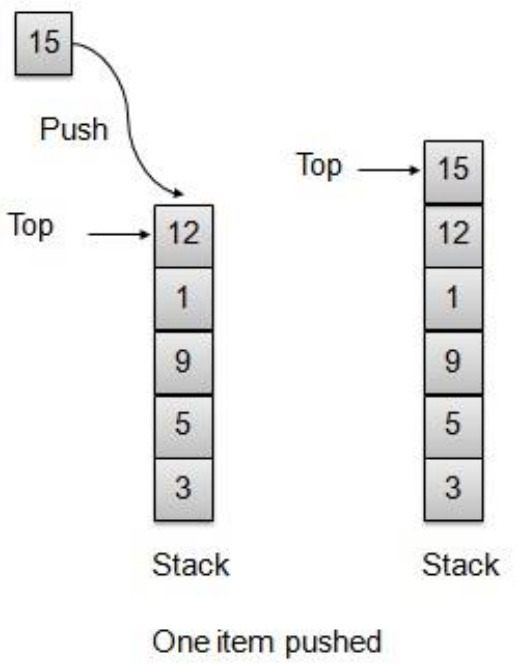
ii) **TOP** is decremented by 1.

Routine for Pop Operation:



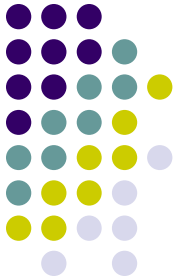
```
void Pop( )
{
 if ( TOP == -1)
 {
   printf ( " Stack Underflow");
   return;
  }
  else
  {
  printf ( "The Popped item is", stack [ TOP]);
  --TOP;
  }
}
```

ShowTop (S) or Peek (S) :



- Check whether the stack is empty.
- Return / Print the top element as pointed by the top pointer.

Routine for ShowTop( ):



```
Void ShowTop ( )
{
int TOP = S→TOP;
If (TOP == -1)
{
printf (“ Stack Underflow”);
Return NULL;
}
return S → Element [TOP];
}
```

Linked List Implementation of Stack:

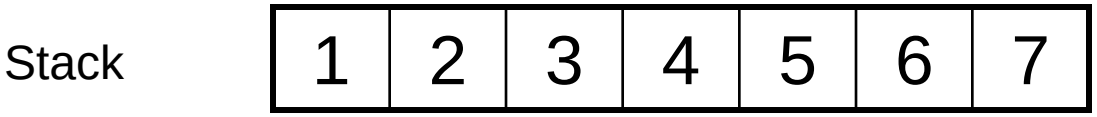
- In the stack Implementation, a stack contains a **TOP** pointer, which is the **"head"** of the stack where pushing and popping data items happens at the head of the list.

- The first node has a **NULL** in the link field and second node-link has the first node address in the link field and so on and the last node address is in the **"TOP"** pointer.

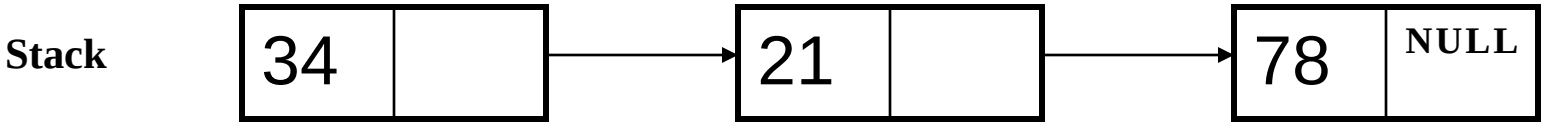
- The main **advantage** of using a linked list over arrays is that it is possible to implement a stack that can shrink or grow as much as needed.
- Using an array will put a restriction on the maximum capacity of the array which can lead to stack overflow. Here each new node will be dynamically allocated. so **overflow is not possible**.

Why Linked list Representation?

- Use linked list when the size of the stack is not known in advance.

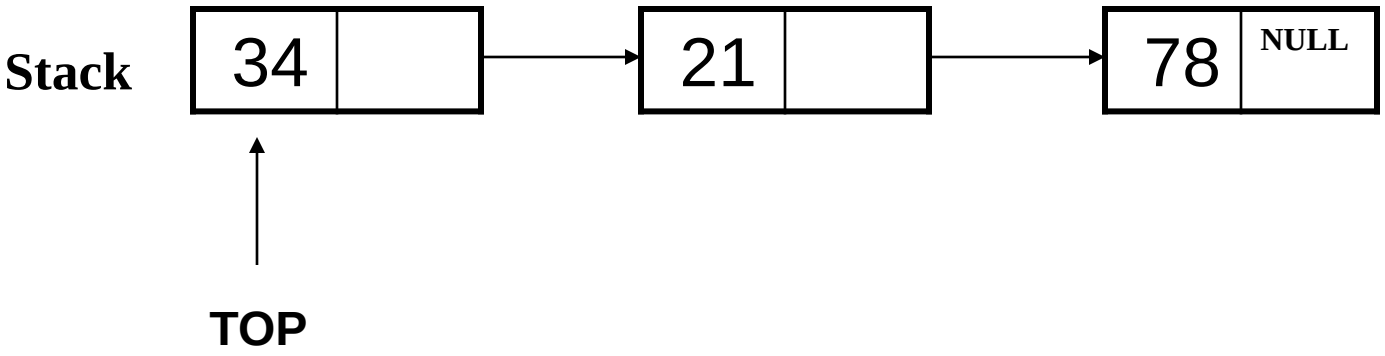


Size is known in advance



No limitation on the number of nodes we can create

Why should we prefer the beginning of the linked list as the Top of the Stack ?



→We will select the beginning of the linked list as the top of the stack.

→For Push operation , a node will be inserted at the beginning of the linked list.

→For Pop operation, every time the first node of the linked list will be deleted.

→ This automatically holds because the TOP pointer is pointing to the first node of the linked list.

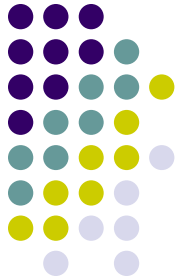
Why we prefer Adding and Removing the first node of the linked list?

Time complexity of adding a node at the beginning : **O(1)**

Time complexity of removing the first node : **O(1)**

But what if we take the end of the linked list as the TOP of the stack?

- Deleting the last node of the singly linked list requires **Traversal.**
- Time Complexity of adding a node at the end : **O(1)**
- Time Complexity of removing the last node: **O(n)**



- The code of the Push ( ) function must be similar to the code of inserting the node at the beginning of the singly linked list.
- The code of the Pop ( ) function must be similar to the code of deleting the first node of the singly linked list.

- **Stack Overflow** occurs when there is no space left to dynamically allocate the memory.
- In that case, malloc ( ) function will return NULL.
- Stack Underflow occurs when top is equal to NULL.

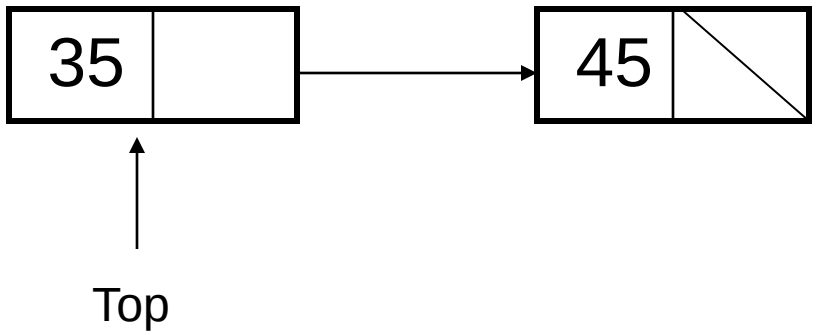
Structure of the Node:

- Structure of the node representing a stack element is same as the structure of the node of the singly linked list. The top pointer must always point to the first node of the linked list.

```
Struct node
 {
  int data;
  Struct node* next;
 } * top = NULL;          // Top pointer is globally declared
```

Push Operation

- Similar to the insert at beginning function of the singly linked list with small changes.

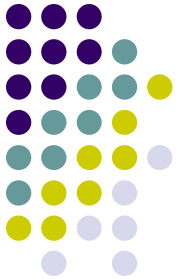


- We have to add a new node with a value of 50 in its data part at the beginning of the list.

Procedure:

- → Create a new node.
- → Allocate memory space for the new node.
- → Store the data in the data field.
- → Put the address of the first node of the linked list in the next part of the new node.
- → Update the top pointer and make it point to the new node of the linked list.

Routine for Push Operation



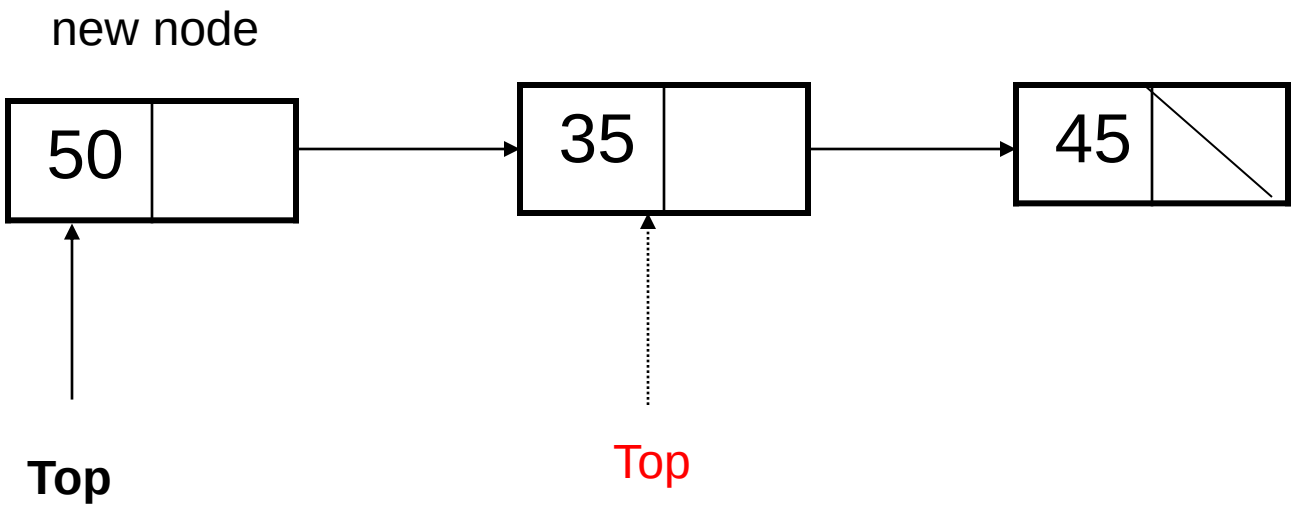
```
sturct node* newnode;
newnode = malloc ( sizeof (sturct node));
if( newnode = = NULL)
```

/* malloc( ) function retruns NULL when the requested memory can't be allocated */

```
{
 printf("Stack Overflow");
 exit (1);
 }
 newnode → data = Element;
 newnode → next = NULL;

  newnode → data = Top;
  Top = newnode;
```

After inserting a new node at the front,



Routine to Print all the elements in the List

```
void Print ( )
 {
   struct node* Temp;
   Temp = Top;
   printf (" The Stack elements are:");
   While ( Temp!= NULL)
   {
    printf ("%d", Temp → data);
    Temp = Temp → next;
    }
   }
```

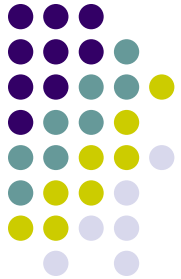
Pop Operation

- Pop function of stack is similar to the delete a node at the front function of the singly linked list with small changes.

Procedure

- → **Create a temporary pointer for the purpose of deletion.**
- → **Update the temporary pointer so that it can point to the first node of the linked list.**
- → **Store the value of the first node somewhere.**
- → **Update the top pointer so that it can point to the next node of the linked list.**
- → **Delete the node pointed by the temporary pointer.**
- → **Return the value of the first node.**

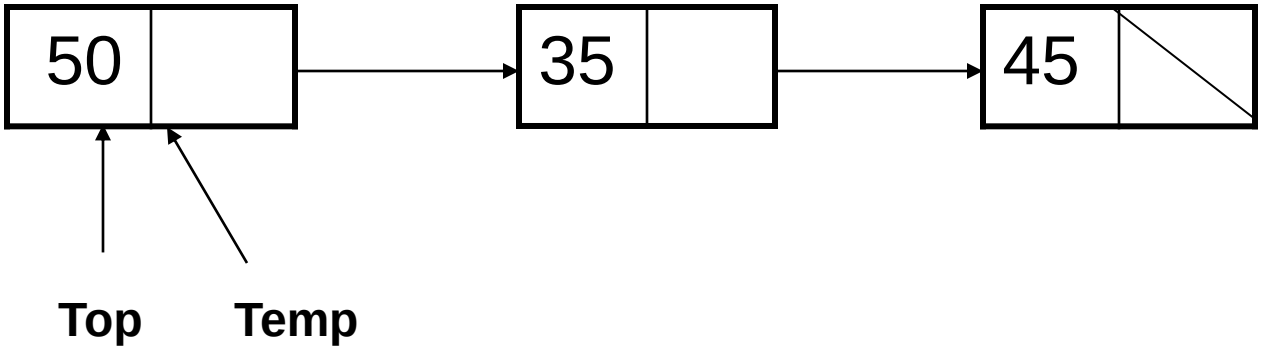
Routine for Pop Function

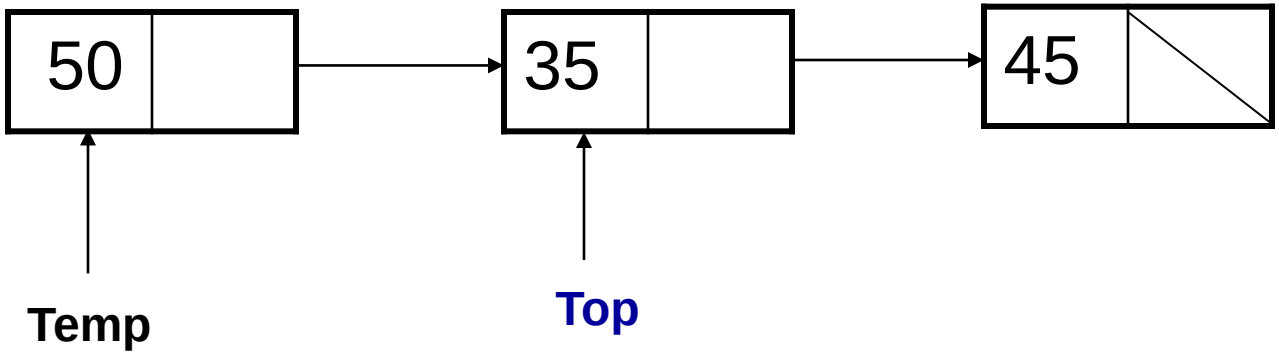


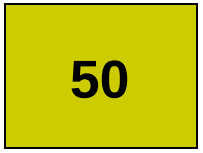
```
int Pop ( )
{
 struct node* Temp;
 Temp = Top;
 int val = Temp → data;

  Top = Top → next;
  free (Temp);
  Temp = NULL;
  return val;      // return the popped element
  }
```

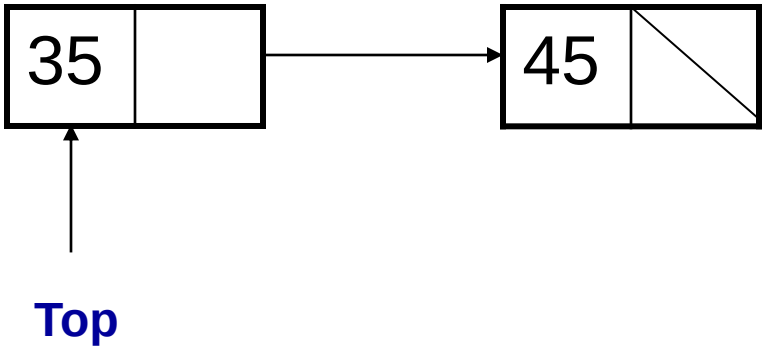
/* Store the value of the first node (data = 50) somewhere and update the top pointer so that it can point to the next node of the linked list */



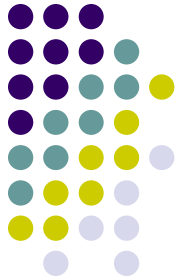




After Popped out 50 from the list,



isEmpty Function

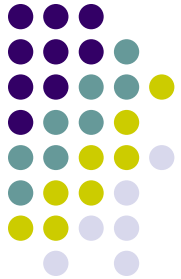


```
int isEmpty ( )
{
 if (Top == NULL)
```

/* Top pointer will the hold NULL when there is no node in the linked list */

```
    return 1;        // Stack is empty
 else
    return 0;        // Stack is not empty
 }
```

ShowTop or Peek Function



```
int ShowTop ( )
{
  if (isEmpty( ))
  {
   printf (“ Stack Underflow”);
   exit (1);
  }
  return Top → data;
 }
```