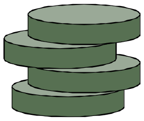
CSE 2215:

Data Structures and Algorithms-I

Stacks



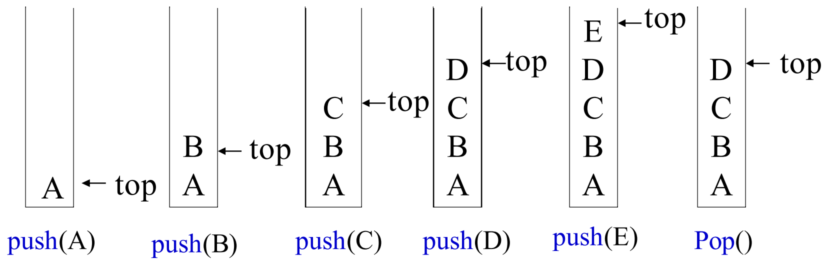
Queues

Stacks and Queues

- A stack is a last in, first out (LIFO) data structure
  - Items are removed from a stack in the reverse order from the way they were inserted
- A queue is a first in, first out (FIFO) data structure
  - Items are removed from a queue in the same order as they were inserted

Stack: Last In First Out

- A *stack* is a list with the restriction that insertions and deletions can be performed in only one position, namely, the *top* of the stack.
- The operations: **push** (insert) and **pop** (delete)



Stack: Last In First Out

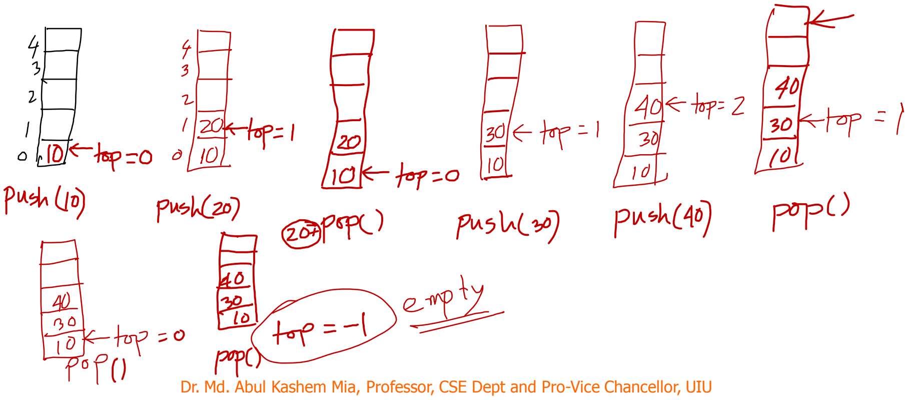
Sample Question:

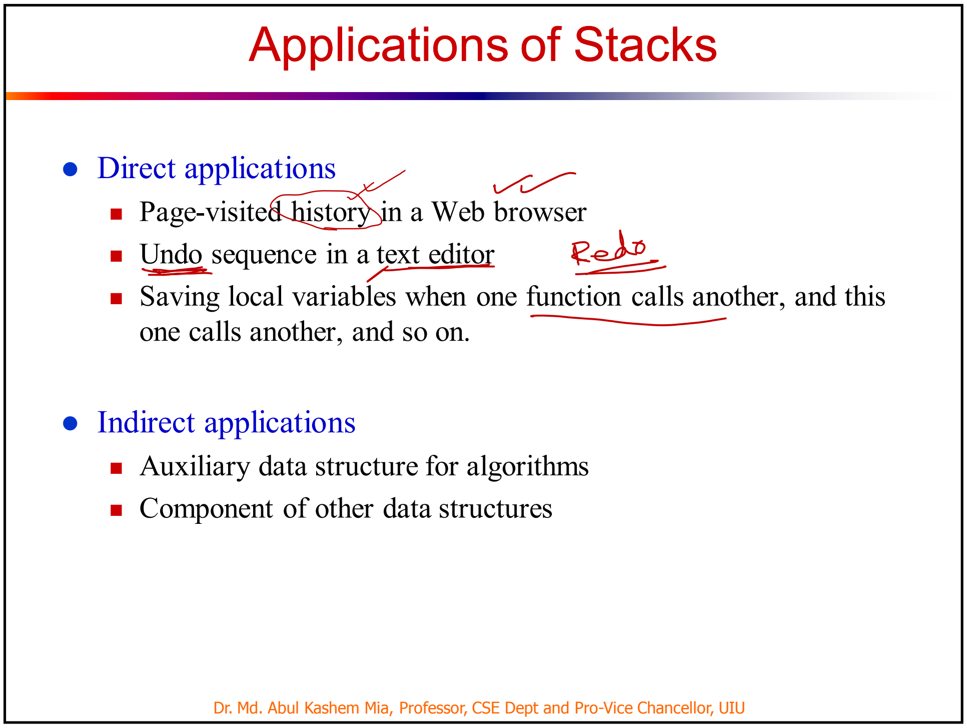
4

Show the status of a STACK implemented by an array of size, m=5 for the operations: push(10), push(20), pop(), push(30), push(40), pop(), pop(), pop().

full

top = m-1

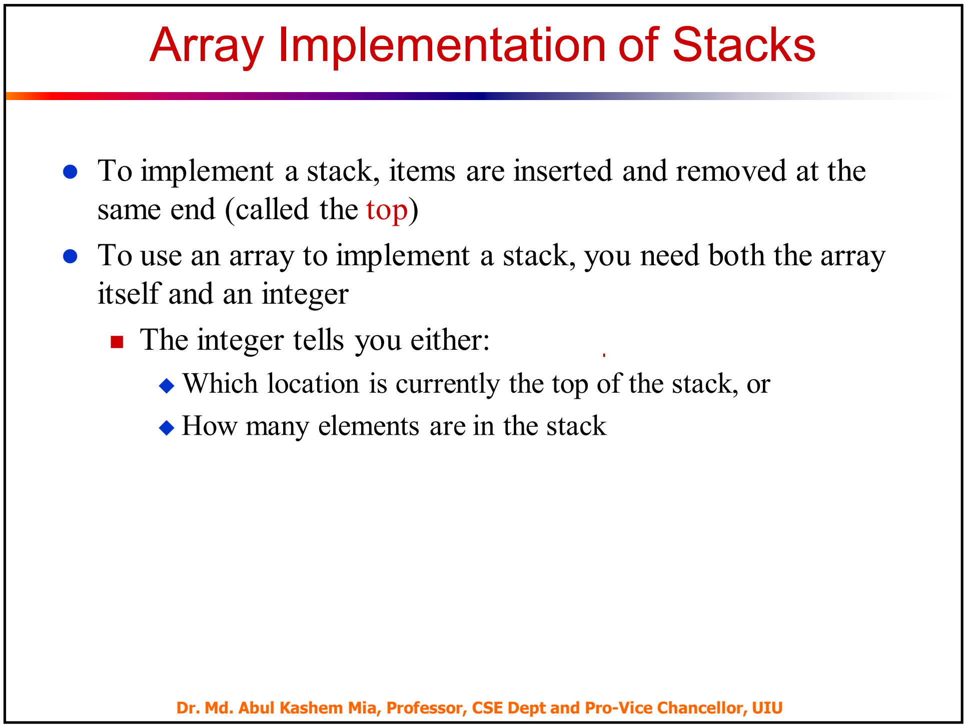




Applications of Stacks

- Direct applications
  - Page-visited history in a Web browser
  - Undo sequence in a text editor Redo
  - Saving local variables when one function calls another, and this one calls another, and so on.
- Indirect applications
  - Auxiliary data structure for algorithms
  - Component of other data structures

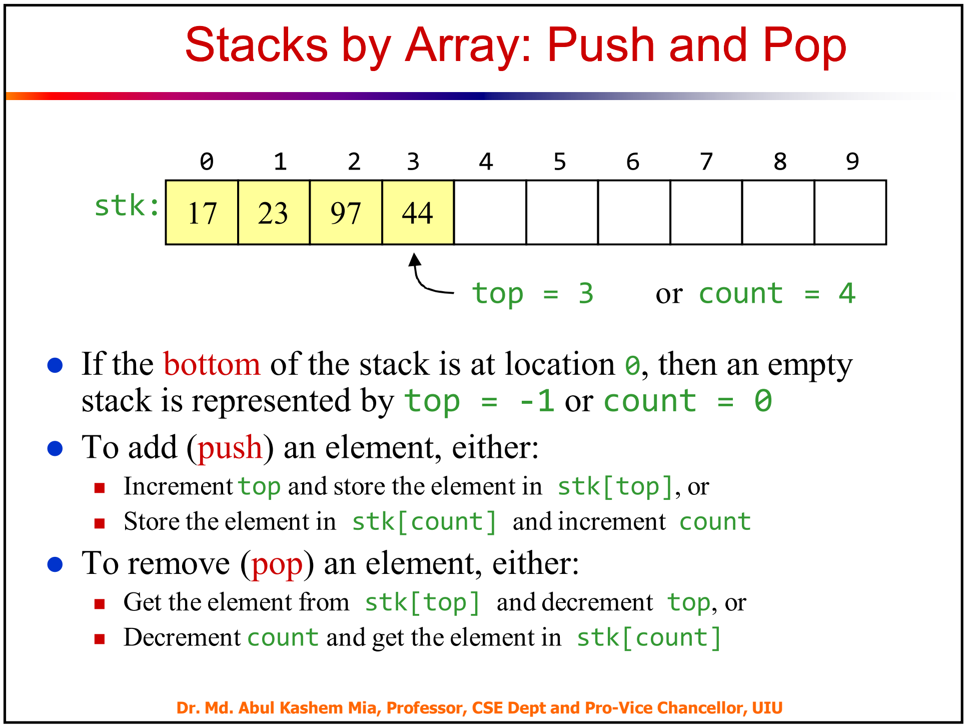
Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU



Array Implementation of Stacks

- To implement a stack, items are inserted and removed at the same end (called the top)
- To use an array to implement a stack, you need both the array itself and an integer
  - The integer tells you either:
    - Which location is currently the top of the stack, or
    - How many elements are in the stack

Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU



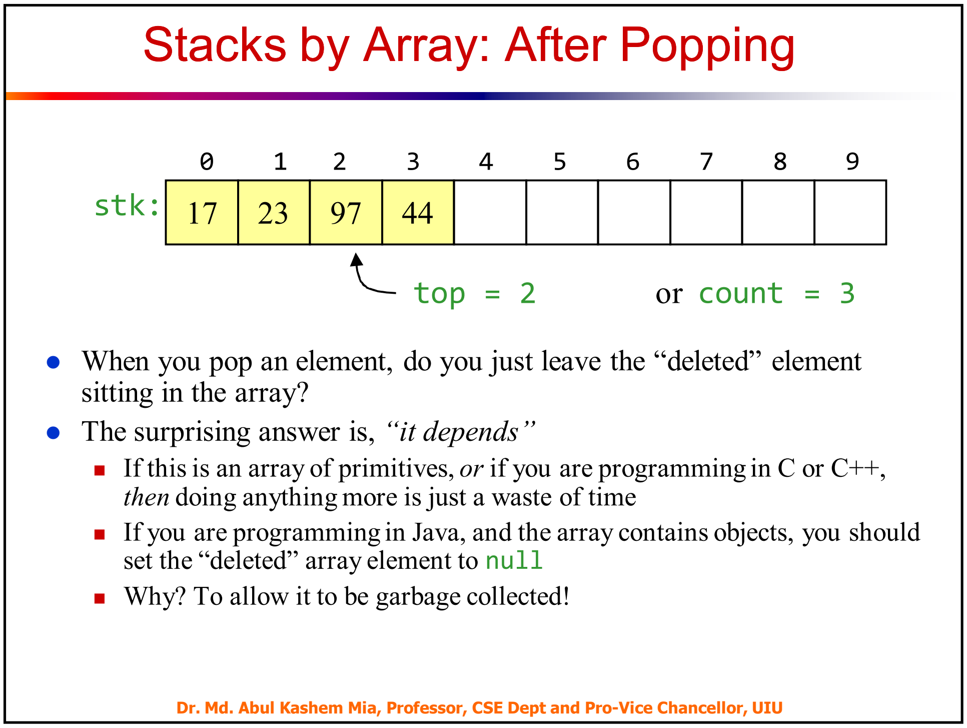
Stacks by Array: Push and Pop

	0	1	2	3	4	5	6	7	8	9
stk:	17	23	97	44						

top = 3 or count = 4

- If the bottom of the stack is at location 0, then an empty stack is represented by top = -1 or count = 0
- To add (push) an element, either:
  - Increment top and store the element in stk[top], or
  - Store the element in stk[count] and increment count
- To remove (pop) an element, either:
  - Get the element from stk[top] and decrement top, or
  - Decrement count and get the element in stk[count]

Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU



Stacks by Array: After Popping

	0	1	2	3	4	5	6	7	8	9
stk:	17	23	97	44						

top = 2 or count = 3

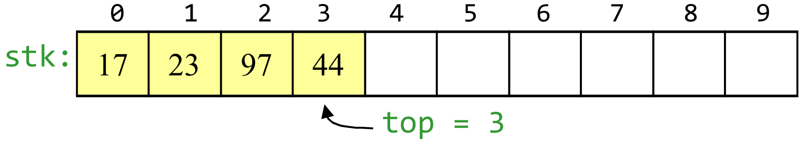
- When you pop an element, do you just leave the "deleted" element sitting in the array?
- The surprising answer is, *"it depends"*
  - If this is an array of primitives, *or* if you are programming in C or C++, *then* doing anything more is just a waste of time
  - If you are programming in Java, and the array contains objects, you should set the "deleted" array element to null
  - Why? To allow it to be garbage collected!

Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU

Stacks by Array: Error Checking

- There are two stack errors that can occur:
  - Underflow: trying to pop (or peek at) an empty stack
  - Overflow: trying to push onto an already full stack
- For underflow, you should throw an exception
  - You could create your own, more informative exception
- For overflow, you could do the same things
  - Or, you could check for the problem, and copy everything into a new, larger array

Stacks by Array: Push and Pop

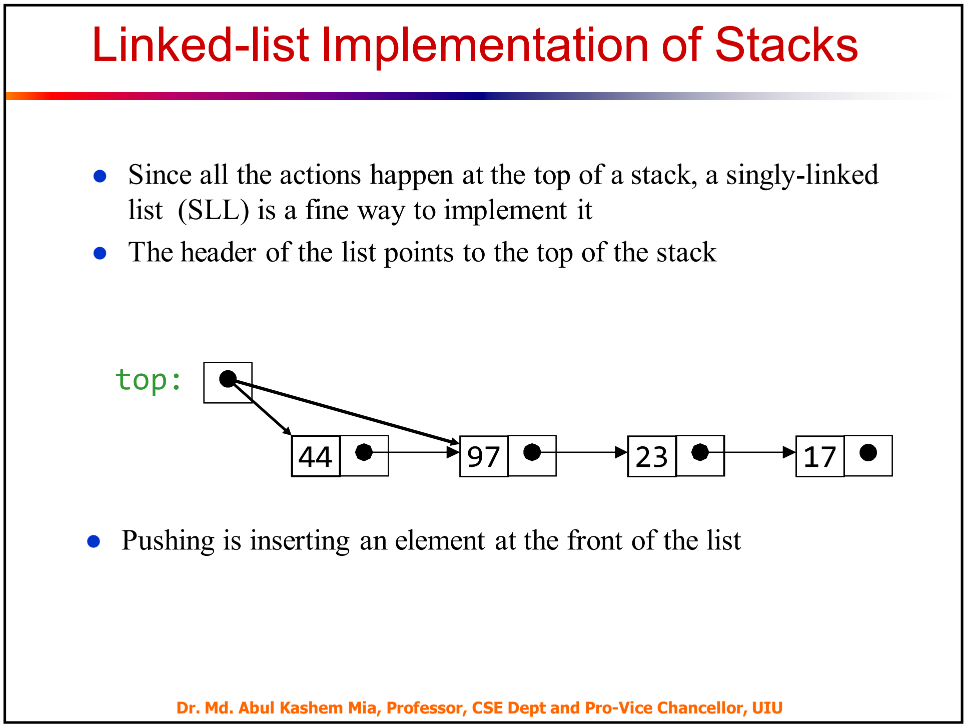


	0	1	2	3	4	5	6	7	8	9
stk:	17	23	97	44						

top = 3

```
void push(int  x){
   if(top >= n-1)
      printf("\n STACK is over flow");
   else {
      top++;
      stk[top] = x;
   }
}
```

```
int pop() {
   int y;
   if(top <= -1)
      printf("\n Stack is under flow");
   else {
      y = stk[top];
      top--;
      return y;
   }
}
```

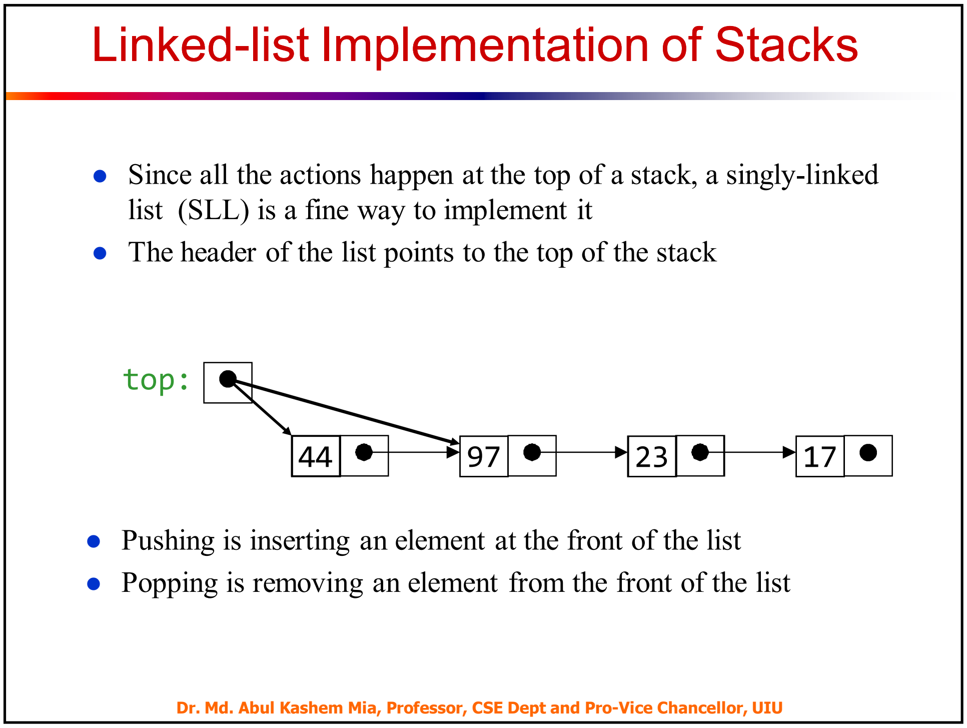


Linked-list Implementation of Stacks

- Since all the actions happen at the top of a stack, a singly-linked list (SLL) is a fine way to implement it
- The header of the list points to the top of the stack

top:
44
97
23
17

- Pushing is inserting an element at the front of the list

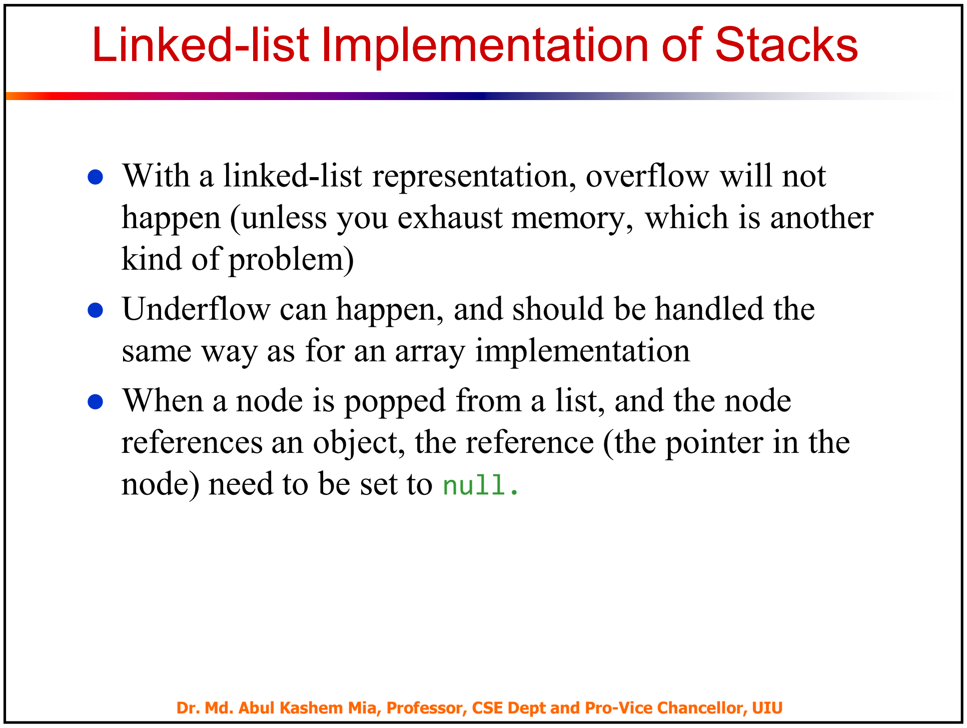


Linked-list Implementation of Stacks

- Since all the actions happen at the top of a stack, a singly-linked list (SLL) is a fine way to implement it
- The header of the list points to the top of the stack

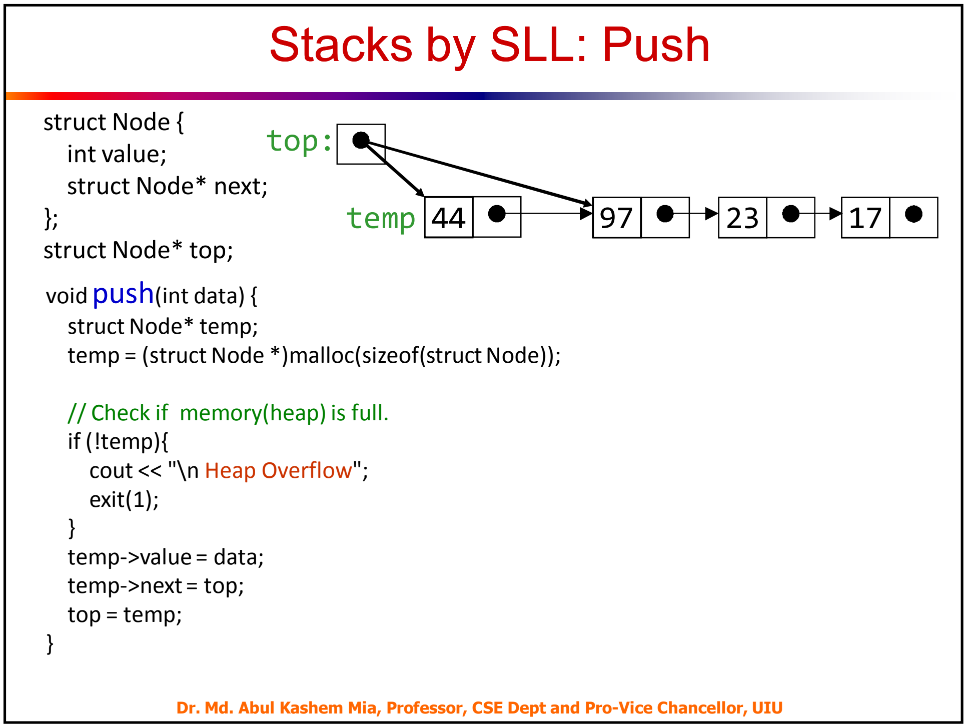
top:
44
97
23
17

- Pushing is inserting an element at the front of the list
- Popping is removing an element from the front of the list



Linked-list Implementation of Stacks

- With a linked-list representation, overflow will not happen (unless you exhaust memory, which is another kind of problem)
- Underflow can happen, and should be handled the same way as for an array implementation
- When a node is popped from a list, and the node references an object, the reference (the pointer in the node) need to be set to `null.`



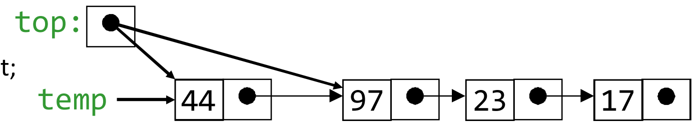
Stacks by SLL: Push

```
struct Node {
   int value;
   struct Node* next;
};
struct Node* top;

void push(int data) {
   struct Node* temp;
   temp = (struct Node *)malloc(sizeof(struct Node));

   // Check if  memory(heap) is full.
   if (!temp){
      cout << "\n Heap Overflow";
      exit(1);
   }
   temp->value = data;
   temp->next = top;
   top = temp;
}
```

Stacks by SLL: Pop



```
struct Node {
   int value;
   struct Node* next;
};
struct Node* top;
int pop(){
   struct Node* temp;
   int data;
   if (top == NULL) {
      cout << "\n Stack Underflow" << endl;
      exit(1); }
   else {
      data = top->value
      temp = top;
      top = top->next;
      free(temp);
      return data;
   }
}
```