Stack

SCSJ2013 Data Structures & Algorithms

Nor Bahiah Hj Ahmad & Dayang Norhayati A. Jawawi

Faculty of Computing

Course Objectives

At the end of the lesson students are expected to be able to:

- Understand stack concept and its structure.
- Understand operations that can be done on stack.
- Understand and know how to implement stack using array and linked list.

Introduction to Stack

What is stack?

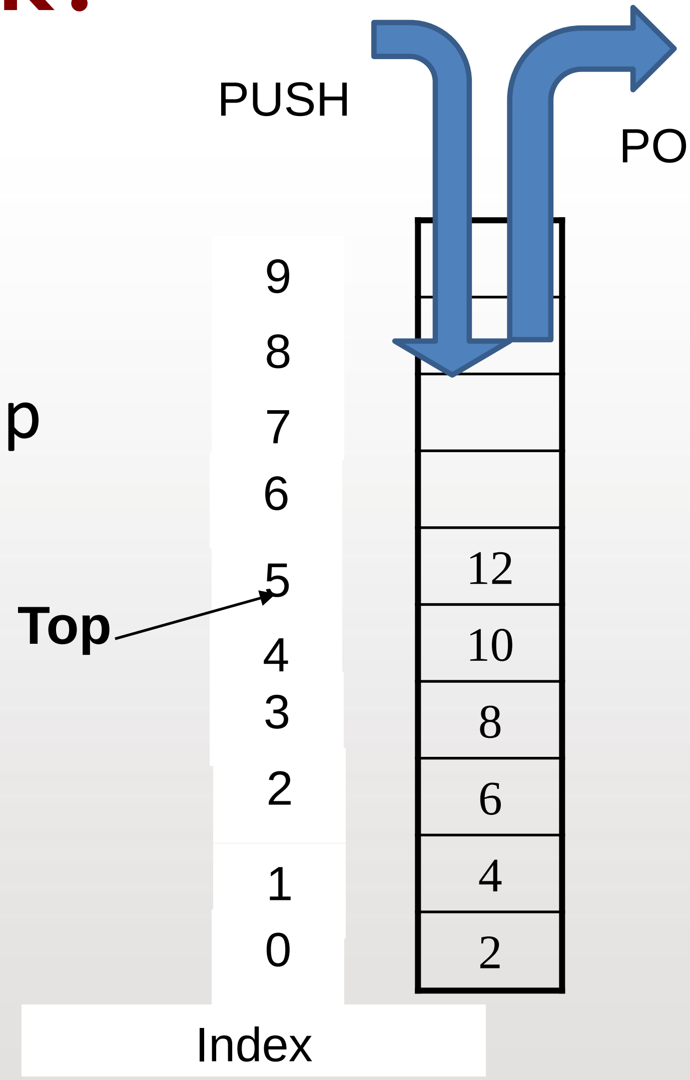
- Stack is a collection of items which is organized in a sequential manner.
- Stack examples: stack of books or stack of plates.

Stack operations

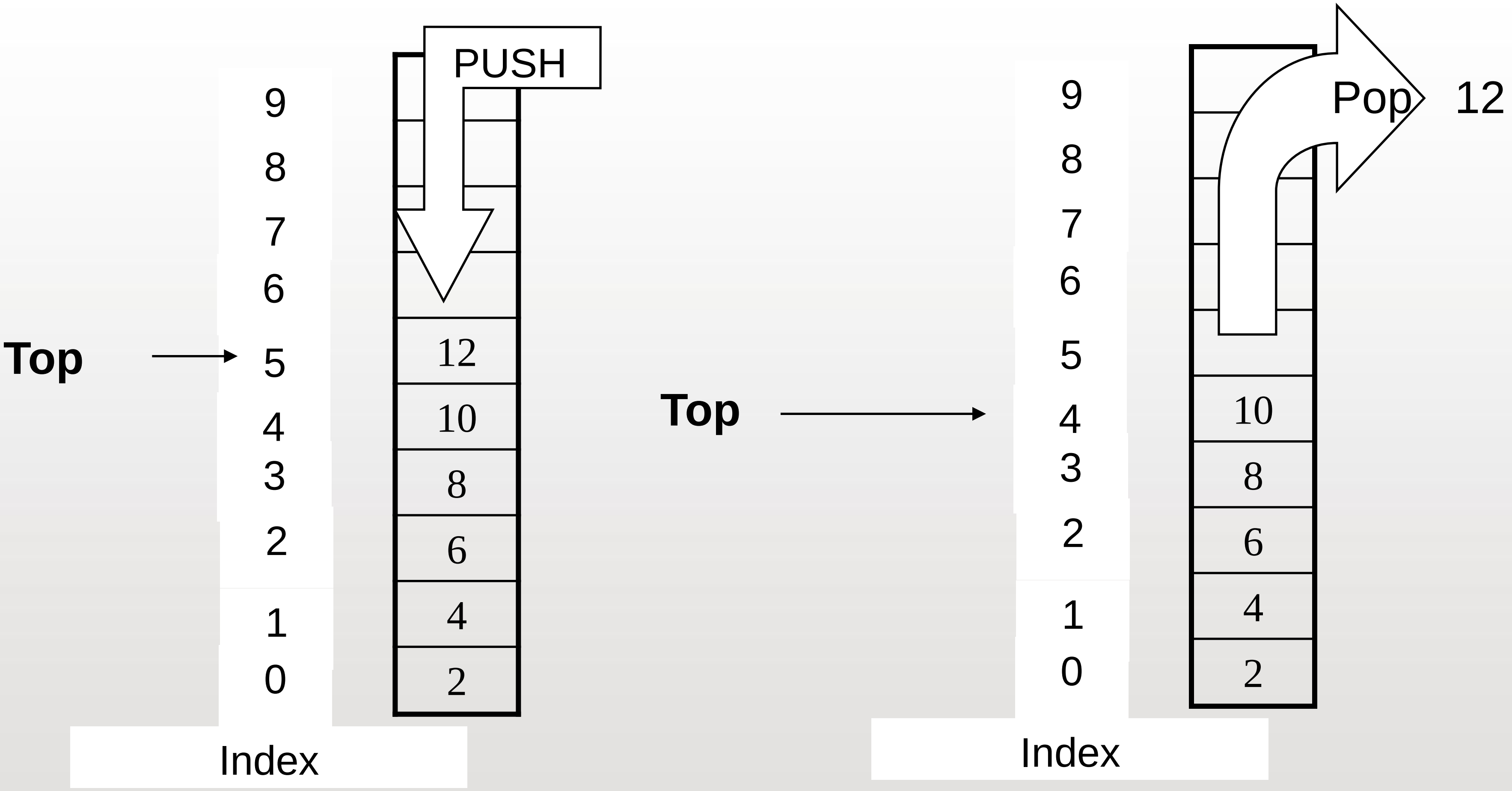
All additions and deletions are restricted at one end, called top. Known as LAST IN FIRST OUT (LIFO) data structure.

What Is Stack?

- Stack is an abstract data type
- Adding an entry on the top (push)
- Deleting an entry from the top (pop)



Last-in First-out (LIFO)



Top → 5

9
8
7
6
5
4
3
2
1
0

Index

PUSH

12
10
8
6
4
2

Top → 4

9
8
7
6
5
4
3
2
1
0

Index

Pop 12

10
8
6
4
2

Implementation for Stack

Array

- Size of stack is fixed during declaration
- Item can be pushed if there is some space available, need isFull( ) operations.
- Need a variable called, top to keep track the top of a stack.
- Stack is empty when the value of top is –1.

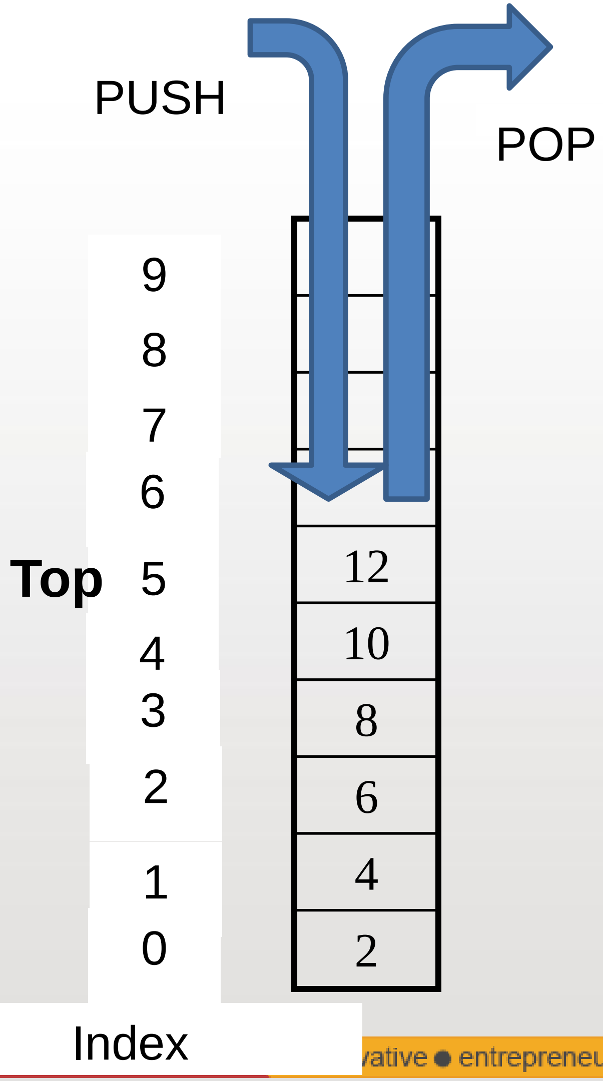
Linked List

- Size of stack is flexible. Item can be pushed and popped dynamically.
- Need a pointer, called top to point to top of stack.

Array Implementation of Stack

Stack Operations:

- createStack()
- push(item)
- pop( )
- isEmpty( )
- isFull( )
- stackTop()



“Stack can be visualized as array, BUT the operations can be done on top stack only. “

Array Implementation of Stack

3 things to be considered for stack with array

1. Stack Empty : when top is -1.
2. Push operations: To insert item into stack

   2 statements must be used

   ```
   top = top + 1;
   stack[top] = newitem;
   ```
3. pop operations: To delete item from stack.

   2 statements should be used

   ```
   Item = stack[top]; or stackTop();
   top = top – 1;
   ```

- `Item = stack[top];` statement is needed if we want to check the value to be popped.

Array Implementation of Stack

Stack declaration:

```
const int size = 100;
class stack
{
  private : // data declaration
   int top ;
   char data[size] ;
  public : // function declaration
  void createStack();
  void push(char) ;    // insert operation
  void pop() ;     // delete operation
  char stackTop() ; // get top value
  bool isFull() ;  // check if stack is Full
  bool isEmpty();  // check if stack is empty
} ;
```

Array Implementation of Stack

- We need two data attributes for stack:
  1. Data : to store item in the stack, in this example data will store char value
  2. top : as index for top of stack, integer type

- Size of the array that store component of stack is 100. In this case, stack can store up to 100 char value.

- Declaration of stack instance:

```
stack aStack;
```

Array Implementation of Stack

`createStack()` operation

- Stack will be created by initializing top to -1.
- `createStack()` implementation:

```
void stack:: createStack();
{
        top = -1;
}
```

- Top is –1 :- means that there is no item being pushed into stack yet.

Array Implementation of Stack

`isFull()` Operation

- This operation is needed ONLY for implementation of stack using array.
- In an array, size of the array is fixed and to create new item in the array will depend on the space available.
- This operation is needed before any push operation can be implemented on a stack.
- `bool isFull()` implementation

```
bool stack::isFull()
{
        return (top == size-1 );
}
```

- Since the size of the array is 100,

  `bool isFull()` will return *true*, If top is 99 (100 – 1).

  `bool isFull()` will return *false*, if there is some space available, top is less than 100.

Array Implementation of Stack

`bool isEmpty()` operation

- This operation will check whether the array for stack is empty.
- This operation is needed before ANY pop operation can be done. If the stack is empty, then pop operation cannot be done.
- `bool isEmpty()` will return *true* if top –1 and return *false* if top is not equal to -1, showing that the stack has element in it.
- `bool isEmpty()`implementation :

```
bool stack::isEmpty()
{
        return ( top == -1 );
}
```

Array Implementation of Stack

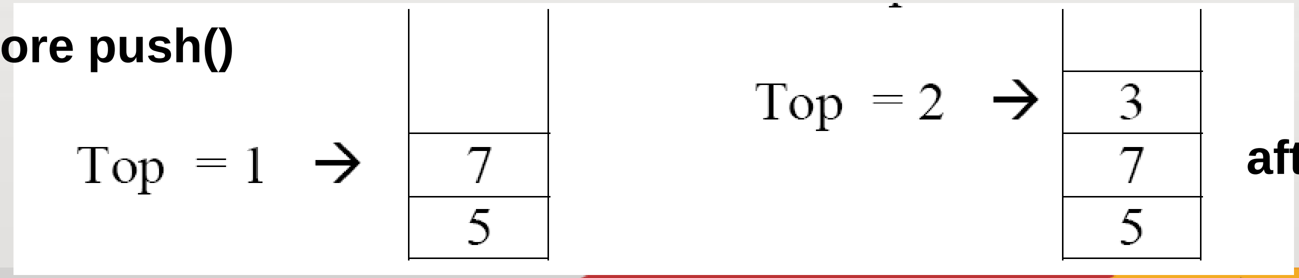
`push(newItem)` operation : Insert item onto stack

- **push()** operation will insert an item at the top of stack. This operation can be done only if there is space availbale in the array
- Before any item can be inserted into a stack, isFull() operation must be called first.
- Insertion operation involve the following steps:
  - Top will be increased by 1.

    top = top + 1;
  - New item will be inserted at the top

    data[Top] = newItem;

before push()



after push()

Array Implementation of Stack

```
void stack::push(char newitem)
{
  if (isFull()) // check whether stack is full
     cout << “Sorry,Cannot push item.
                 Stack is now full!"<< endl;
  else
  {   top = top + 1    // Top point to next index
      data[top] = newitem; //assign new item at top
   }//end else
}//end push()
```

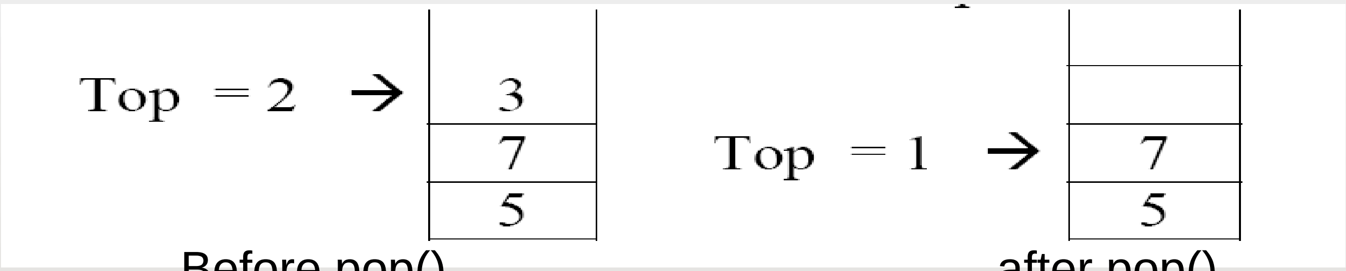
Top will be increased first before item is inserted in order to avoid inserting item at the current top value.

Array Implementation of Stack

pop() Operation

- This operation will delete an item at top of scak.
- Function isEmpty() will be called first in order to ensure that there is item in a stack to be deleted.
- pop() operation will decrease the value of top by 1:

```
top = top - 1;
```



Before pop() after pop()

Array Implementation of Stack

```
void stack::pop()
{
  char item;
  if ( isEmpty() )
   cout << “Sorry, Cannot pop item.
                   Stack is empty!” << endl;
  else
  { //display value at top to be deleted
   cout << “Popped value :” << data[top];
      top = top - 1;
       // top will hold to new index
  }// end if
}//end pop
```

Array Implementation of Stack

`stackTop()` **operation : to get value at the top**

```
 char stackTop()
 { //function to get top value
    if (isEmpty())
       cout <<“Sorry, stack is empty!”<<
endl;
     else
       return data[top];
  } // end stackTop
```

Linked List Implementation of Stack

Linked List Implementation of Stack

- Stack implemented using linked list – number of elements in stack is not restricted to certain size.
- Dynamic memory creation, memory will be assigned to stack when a new node is pushed into stack, and memory will be released when an element being popped from the stack.
- Stack using linked list implementation can be empty or contains a series of nodes.
- Each node in a stack must contain at least 2 attributes:
- i) data – to store information in the stack.
- ii) pointer next (store address of the next node in the stack

Linked List Implementation of Stack

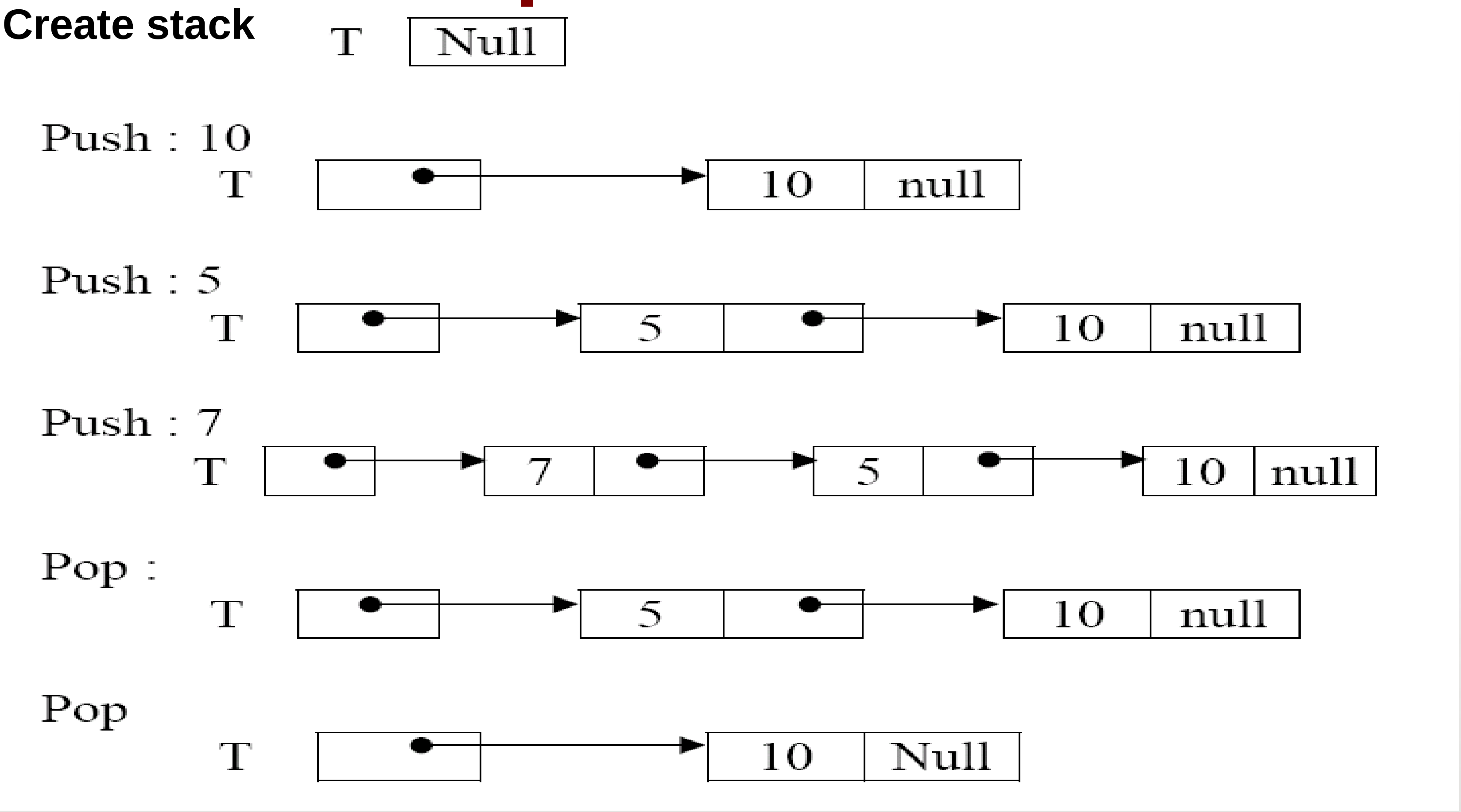
Basic operations for a stack implemented using linked list:

- `createStack()` – initialize top
- `push(char)` – insert item onto stack
- `pop()` – delete item from stack
- `isEmpty()` – check whether a stack is empty.
- `stackTop()` – get item at top

`isFull()` operation is not needed since elements can be inserted into stack without limitation to the stack size.

- Push and pop operations can only be done at the top ~ similar to add and delete in front of the linked list.

Linked List Implementation of Stack: push() and pop() operations



Linked List Implementation of Stack

```
class nodeStack
{
   int data;
   nodeStack *next;
};
class stack
{
   private: // pengisytiharan ahli data
      nodeStack *top;
   public : // pengisytiharan ahli fungsi
      void createStack(); // set Top to NULL
      void push(int) ; // insert item into stack
      void pop() ; // delete item from stack
      int stackTop() ; // get content at top stack
      bool isEmpty(); // check whether stack is empty
};
```

Create Stack and isEmpty()

- Creating a stack will initialize top to NULL - showing that currently, there is no node in the stac

```
void stack::createStack()
{
     top = NULL;
}
```

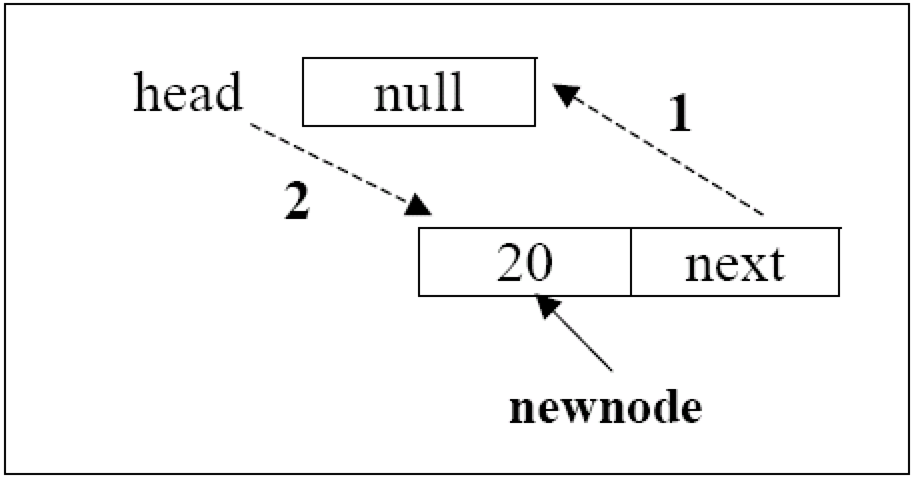
- Is Empty() stack will return true if stack is empty, top is NULL.

```
bool stack::isEmpty()
{
 return (top == NULL);
}
```

push() operations

- 2 conditions for inserting element in stack:
  - Insert to empty stack.
  - Insert item to non empty stack : stack with value.

push () to empty stack



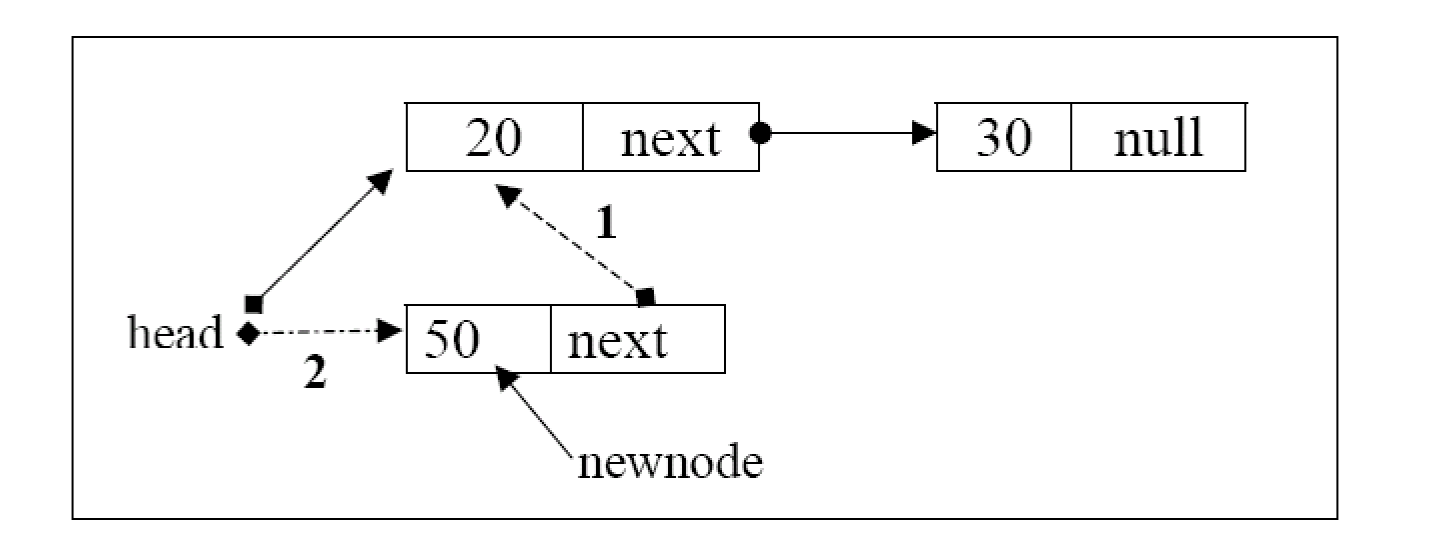
In this situation the new node being inserted, will become the first item in stack.

STEP 1 : newnode-> next = head;

STEP 2 : head = newnode;

push() to non-empty stack

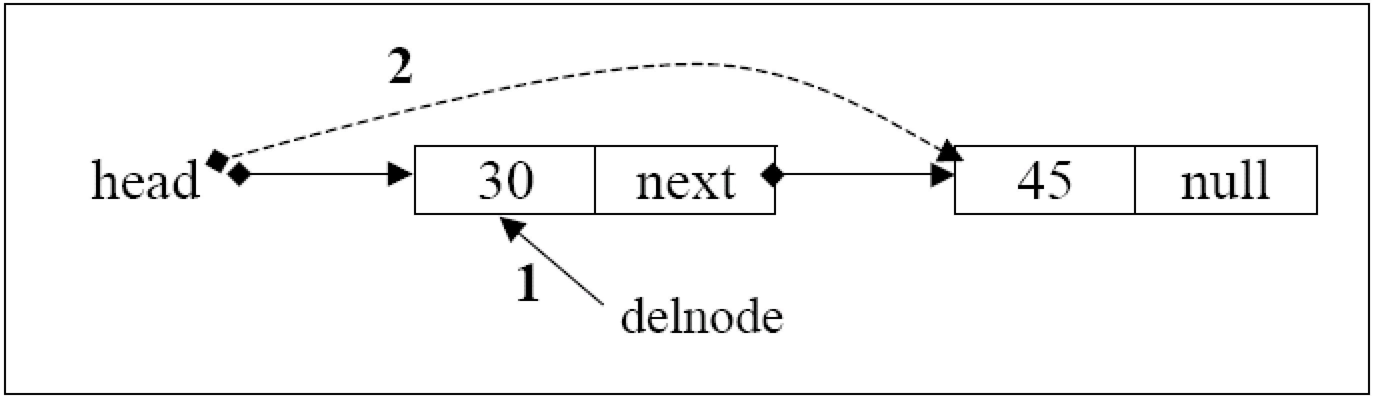
This operation is similar to inserting element in front of a linked list. The next value for the new element will point to the top of stack and head will point to the new element.



STEP 1 : newnode-> next = head;
STEP 2 : head = newnode;

Delete item from stack (pop)

- Pop operation can only be done to non-empty stack. Before pop() operation can be done, operation must be called in order to check whether the stack is empty or there is item in the stack. If `isEmpty()` function return true, pop() operation cannot be done.
- During pop() operation, an external pointer is needed to point to the delete node. In the figure below, `delnode` is the pointer variable to point to the node that is going to be deleted.



STEP 1 : delnode = head;

STEP 2 : head = delnode -> next; or **head = head->next;**

STEP 3 : delete(delnode);

Summary

What we have learned so far….

- Stack is a LIFO data structure
- Can be implemented using array and link list
- Basic Operation for a stack are follows:
  - `createStack(),Push(),Pop()`
  - `stackTop(),isEmpty(),isFull()`

Thank You

http://comp.utm.my/