SLIIT

Discover Your Future

DS before mid

AI after mid

IT2070 – Data Structures and Algorithms

Lecture 01

Introduction to Stack

Subject Group

Malabe Campus

- Mr. Samantha Rajapaksha
- Ms. Dinuka Wijendra
- Ms.Jenny
- Ms.Namali Walgampaya
- Dr.Charika Weerasiriwardena

Metro Campus

- Mr. Samantha Rajapaksha

Kandy Center

- Ms. Chathurika Pinnaduwage

Mathara Center

- Mr.Ravi Supunya

Teaching Methods

- Lectures – 2 hours/week
- Tutorials – 1 hour/week
- Labs – 2 hours /week

Student Evaluation

- Assessments (Two exams) - 40 %
  - 20 % mid
  - 20 % assignment before final
- Final Examination - 60 %
  - 4 questions

Lectures will cover

Data Structures

- Stack data structure
- Queue data structure
- Linked list data structure
- Tree data structure

} before mid term

Algorithms

- Asymptotic Notations
- Algorithm designing techniques
- Searching and Sorting algorithms

} after mid term

- optimization lesson

Tutorials and Labs will cover

- Solve problems using the knowledge acquired in the lecture
- Get hands on experience in writing programs
  - Java – for data part (Eclipse IDE for practicals)
  - Python – for algorithm part

Data Structures and Algorithms

- **Data Structures**

  - Data structure is an arrangement of data in a computer's memory or sometimes on a disk.

Ex: stacks, queues, linked lists, trees

- **Algorithms**

  - Algorithms manipulate the data in these structures in various ways.

Ex: searching and sorting algorithms

Data Structures and Algorithms

- **Usage of data structures**

  - Real world data storage
  - Real world modeling
    - queue, can model customers waiting in line
    - graphs, can represent airline routes between cities
  - Programmers Tools
    - stacks, queues are used to facilitate some other operations

Data Structures and Algorithms

Algorithms → a sequence of rules to perform a task

Algorithm is a well defined computational procedure that takes some value or set of values as input and produce some value or set of values as output.

An algorithm should be

- correct. mathematically correct
- unambiguous. should be unique
- give the correct solution for all cases. situations
- simple. & not long unnecessarily
- terminate.

Academic Integrity Policy

Are you aware that following are not accepted in SLIIT???

Plagiarism - using work and ideas of other individuals intentionally or unintentionally

Collusion - preparing individual assignments together and submitting similar work for assessment.

Cheating - obtaining or giving assistance during the course of an examination or assessment without approval

Falsification – providing fabricated information or making use of such materials

From year 2018 the committing above offenses come with serious consequences !

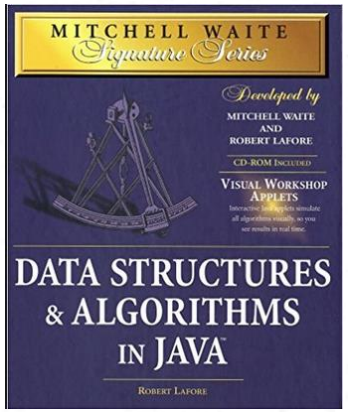
See General support section of Courseweb for full information.

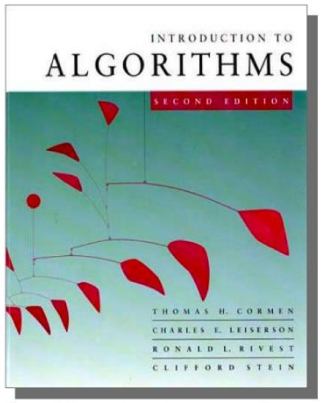
References

available online

1. Mitchell Waite,Robert Lafore, Data Structures and Algorithms in Java,2nd Edition, Waite Group Press,1998.

2. T.H. Cormen, C.E. Leiserson, R.L. Rivest, Introduction to Algorithms,3rd Edition, MIT Press, 2009.

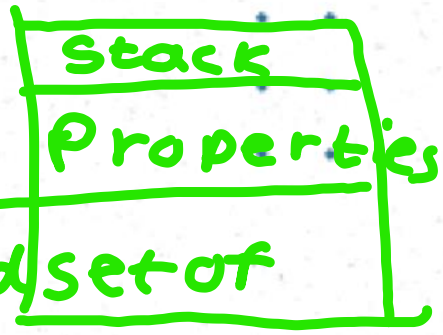




Lecture 1

Stack Queue Tree Linked List

Abstract Datatype = with fixset of operations the values can't be changed



Data Structures and Algorithms

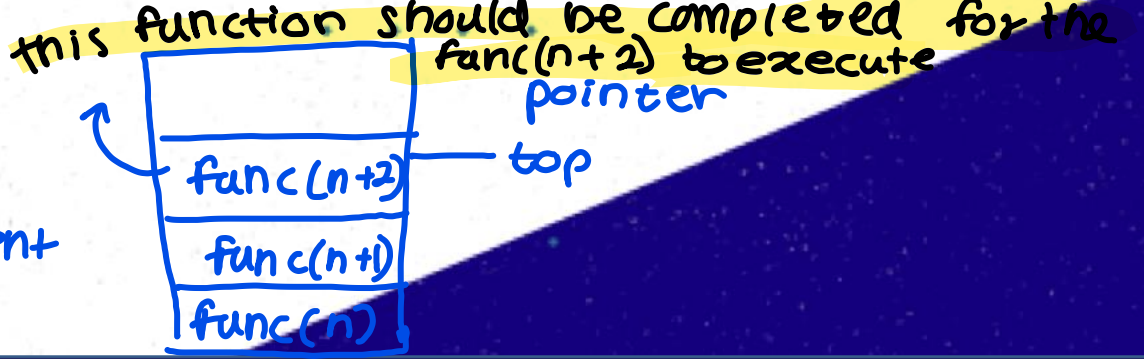
Stacks

Recursive function calling

↳ a function calling itself within a function

```
func(){
    func();
}
```

the internal parameters are different everytime they call it.



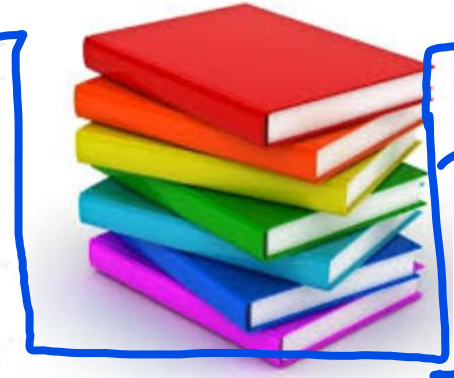
Stack

Stack is a LAST IN FIRST OUT Data algorithm

we must access the top plate (top element) which is the last element inserted to the stack

example-: the undo button (CTRL Z)

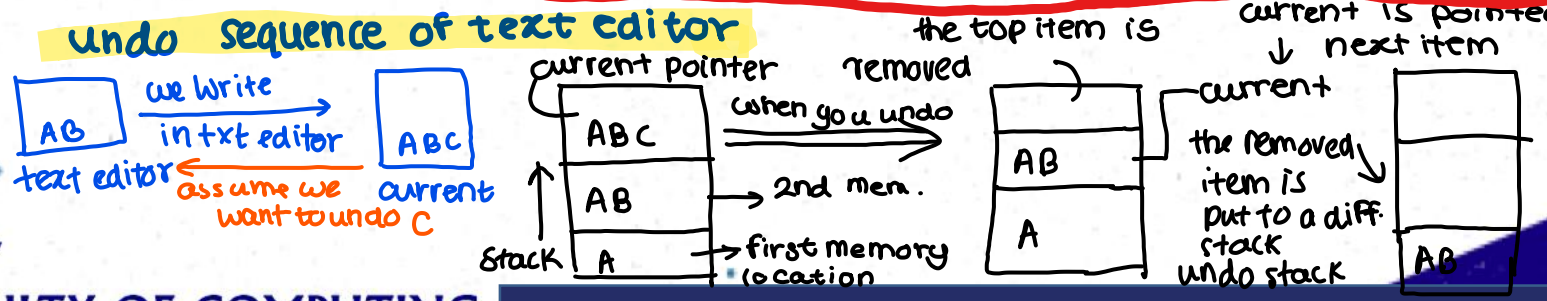
- back button in browsers



if we can't access this we must first access top book (last by) example it has a jar inserted book

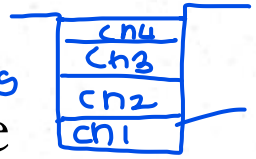
in here orange book

- Allows access to only one data item; the last item inserted
- If you remove this item, then you can access the next-to-last item inserted



Application of Stacks

assume this is a stack of strings



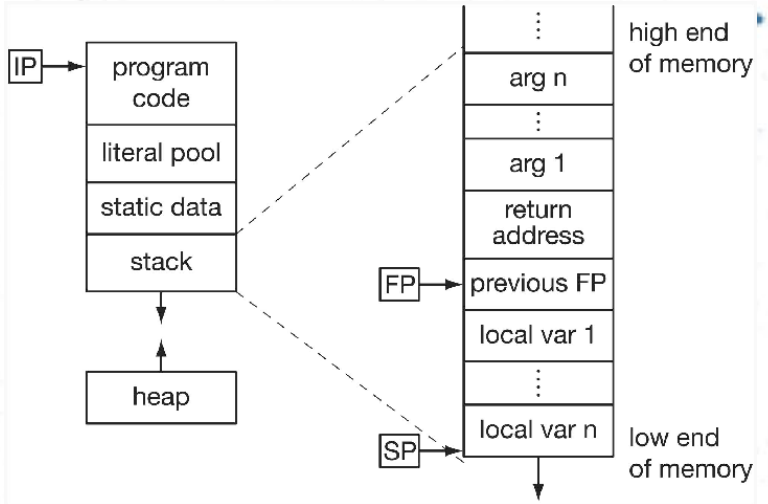
if you want to remove this you have to remove as ch4 / ch3 / ch2 / ch1 which is the reverse order of ch they were put inside in the begining

these two are similar

- String Reverse
- Page visited history in Web browser.
- Undo sequence of text editor.
- Recursive function calling.

additional

- Auxiliary data structure for Algorithms.
- Stack in memory for a process



arrays contain element of the same data type while stacks may contain elements of different data types

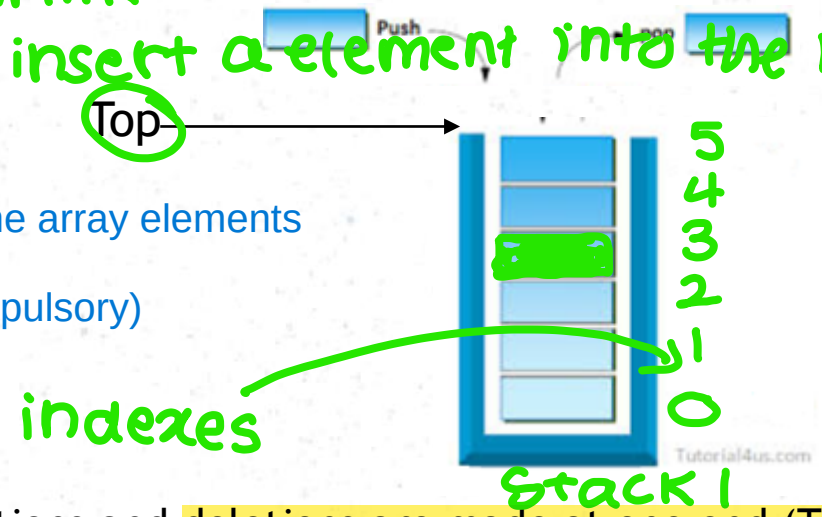
Stack

limitations
- can't extend /shrink
- we can't easily insert a element into the middle of the array

these problems can be solved using linked list

stacks solve the problem of being able to extend at run time whereas in arrays it is constant through out the run time

the key point here is "easily" the array elements can be accessed but we need to provide the array index(compulsory)



in a stack you have a container
Stack has 2 implementations
↓ Array
↓ Linked List

- In a stack all insertions and deletions are made at one end (Top). Insertions and deletions are restricted from the Middle and at the End of a Stack

we can't access a middle element in stack

- Adding an item is called Push
- Removing an item is called Pop

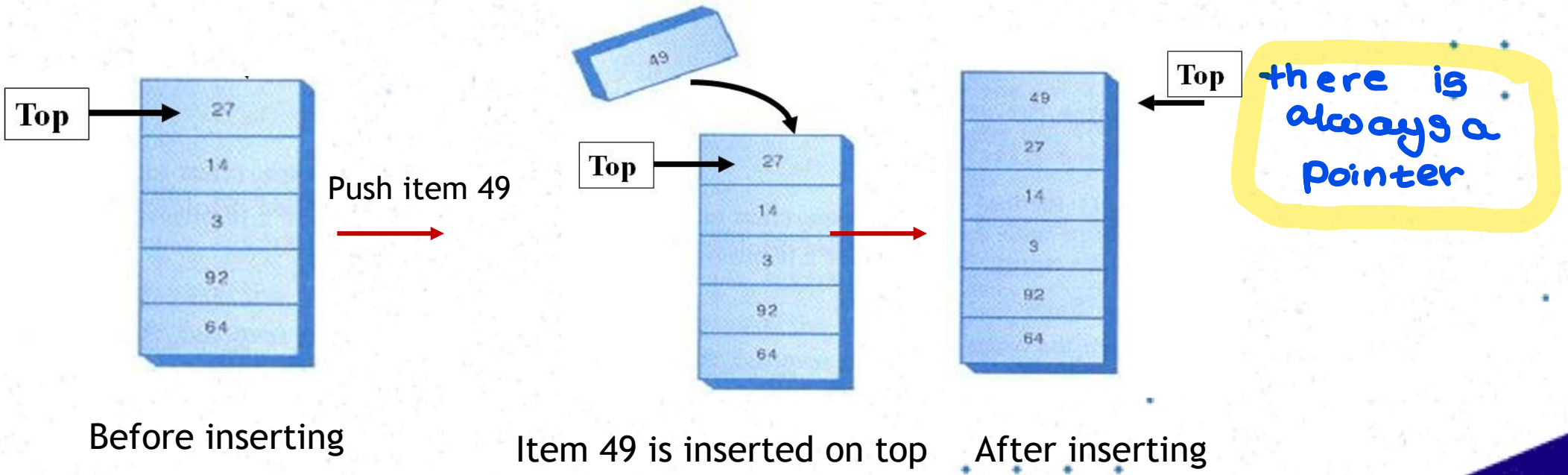
```
Stack1.push()
Stack1.pop()
```

- Elements are removed from a Stack in the reverse order of that in which the elements were inserted into the Stack
- The elements are inserted and removed according to the Last-In-First-Out (LIFO) principle.

basic principal of Stacks

Stack - Push

Stack
- push()
- pop()
- peek()

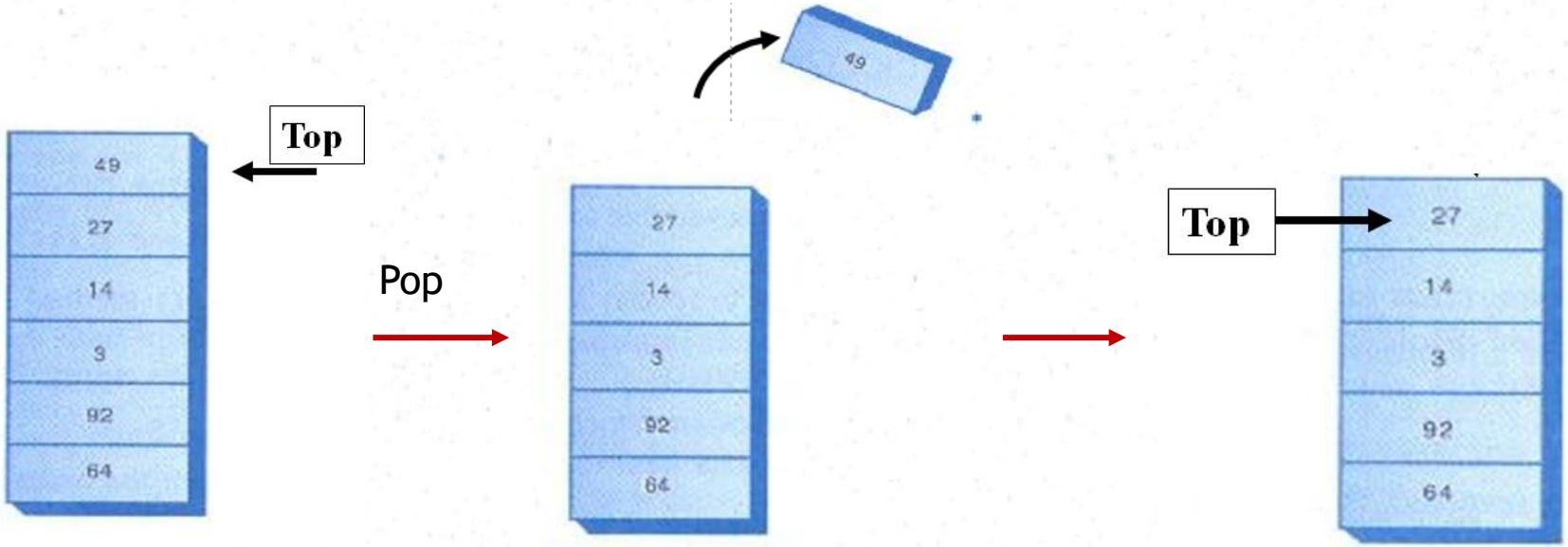


Before inserting

Item 49 is inserted on top

After inserting

Stack - Pop



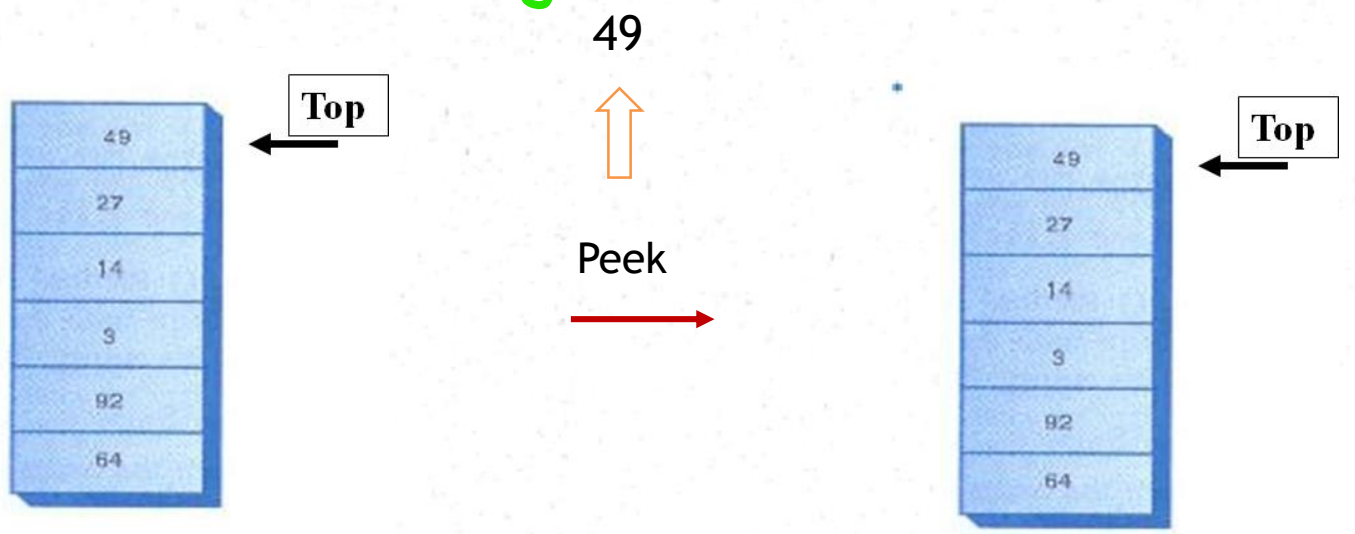
Before removing

Top item is removed

After removing

Stack - Peek

go and have a look at the topmost element



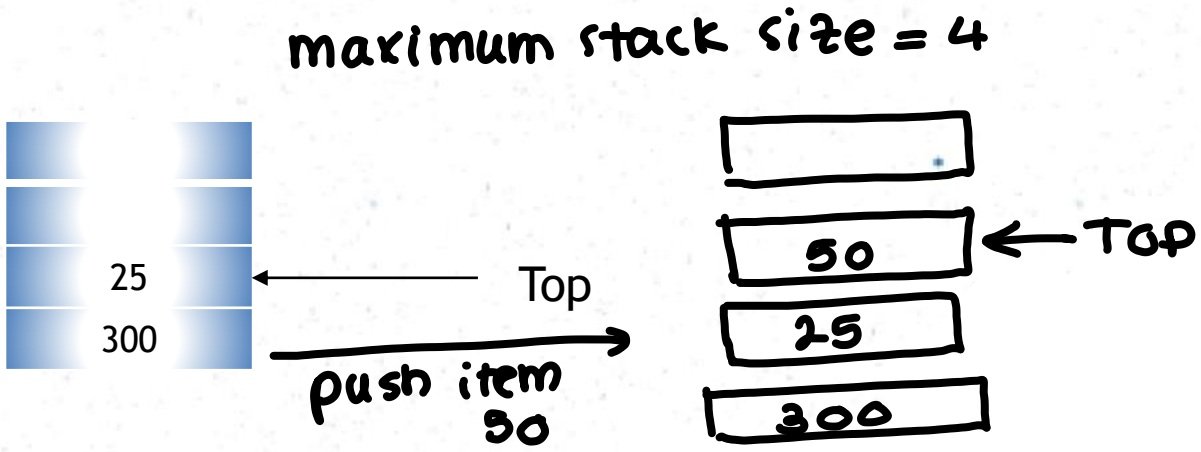
Stack remains the same

No changes to the stack here

Peek is used to read the value from the top of the stack without removing it. You can peek only the Top item, all the other items are invisible to the stack user.

Question

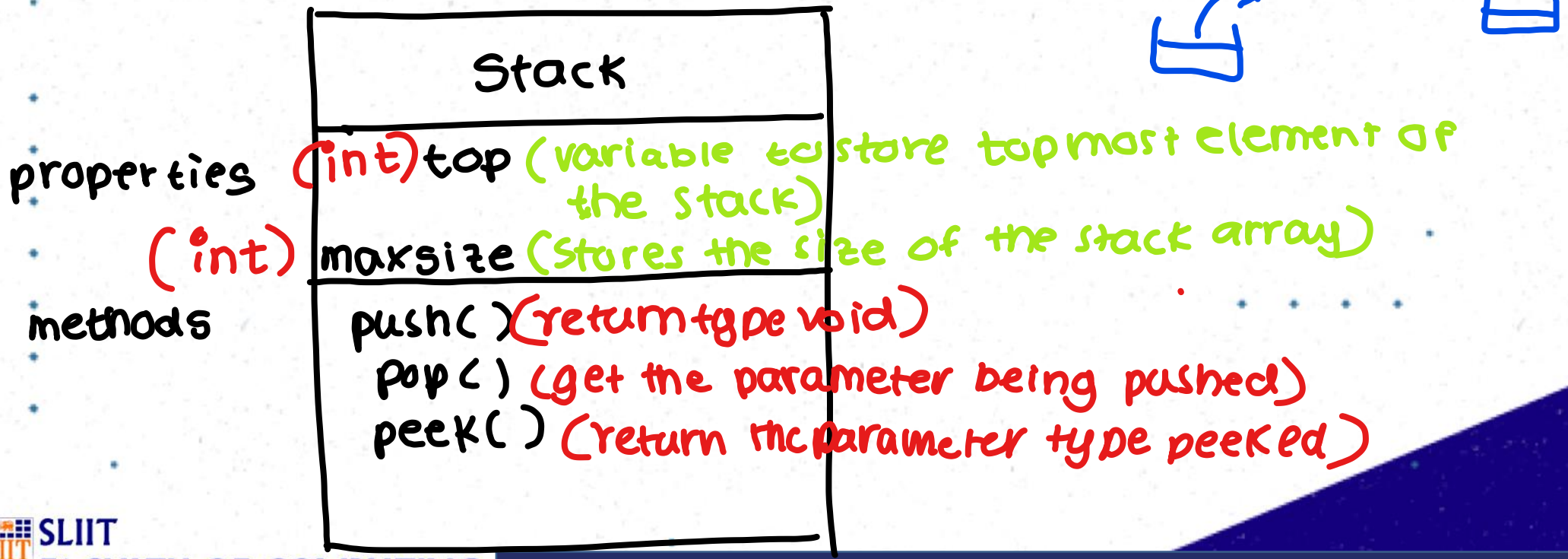
Draw the stack frame after performing the below operations to the stack given below.



i) Push item 50
ii) Push item 500
iii) Peek
iv) Push item 100
v) Pop
vi) Pop
vii) Pop
viii) Pop

Uses of Stack

- The stack operations are built into the microprocessor.
- When a method is called, its return address and arguments are pushed onto a stack, and when it returns they're popped off.



Stack - Implementation

Stack implementation using an **array**

- Constructor creates a new stack of a size specified in its argument.

- Variable *top*, which stores the index of the item on the top of the stack.

```
class StackX {

    private int maxSize;     // size of stack array
    private double[] stackArray;
    private int top;          //top of the stack
    publc StackX(int s) {    // constructor

        maxSize = s;         // set array size
        stackArray = new double[maxSize];
        top = -1;            // no items
    }
}
```

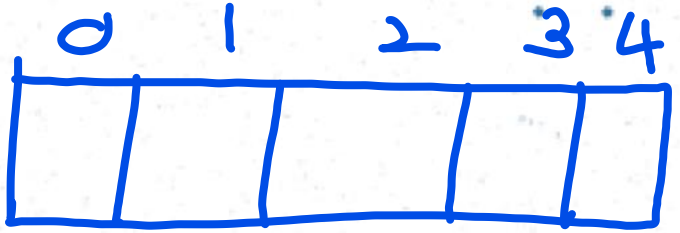
Index of top element

Not pointing to 0 element

at the initial state the stack is empty so this is pointing to (-1)

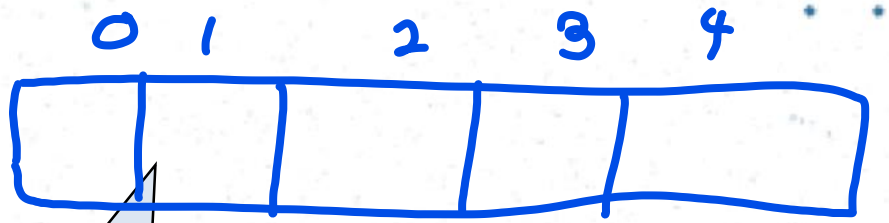
Stack – Implementation - push

```
class StackX{

    private int maxSize;    // size of stack array
    private double[] stackArray;
    private int top;          //top of the stack

    publc StackX(int s) { // constructor

          maxSize = s;       // set array size
          stackArray = new double[maxSize];
          top = -1;           // no items
    }
  public void push(double j) {

        // increment top
        // insert item
  }
```



Stack – Implementation - push

```
class StackX {
        private int maxSize;     // size of stack array
        private double[] stackArray;
        private int top;          //top of the stack

        publc StackX(int s) {// constructor

                maxSize = s;       // set array size
                stackArray = new double[maxSize];
                top = -1;           // no items
        }
      public void push(double j) {

            // increment top.  insert item
           stackArray[++top] = j;
      }
```



it is pointing to(-1)

insert item

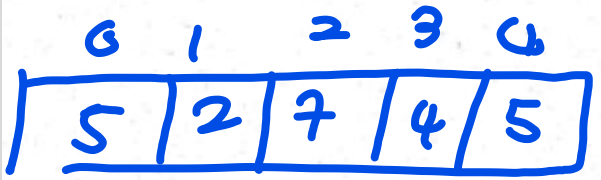
this function incharge of the top pointer

What if the stack is full ?

pre increment is to increase pointer value

Stack – Implementation - push

```
class StackX
{
        private int maxSize;    // size of stack array
        private double[] stackArray;
        private int top;          //top of the stack

        public StackX(int s) {// constructor

                maxSize = s;       // set array size
                stackArray = new double[maxSize];
                top = -1;           // no items
        }
      public void push(double j) {

                // check whether stack is full
                if (top == maxSize - 1)
                        System.out.println("Stack is full");
                else
                        stackArray[++top] = j;
      }
```



if pointer is at the top of the stack

total number of element

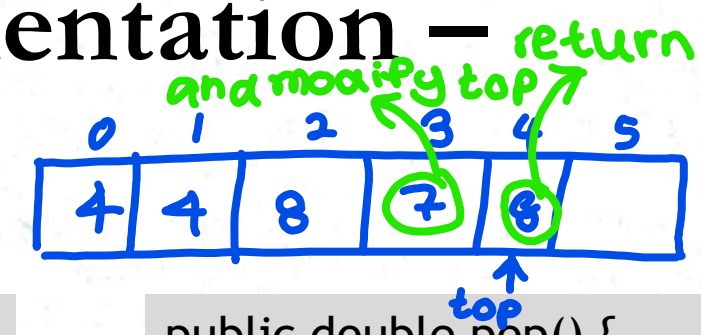
increment pointer

Stack – Implementation – pop/peek

```
class StackX
{
        private int maxSize;    // size of stack array
        private double[] stackArray;
        private int top;          //top of the stack

        public StackX(int s) {// constructor

                maxSize = s;       // set array size
                stackArray = new double[maxSize];
                top = -1;            // no items
        }
        public void push(double j) {

                // check whether stack is full
                if (top == maxSize - 1)
                     System.out.println("Stack is full");
                else
                     stackArray[++top] = j;
       }
```

```
public double pop() {
            // check whether stack is empty
            // if not
               // access item and decrement top
     }

public double peek() {

            // check whether stack is empty
            // if not
               // access item
     }
```

Stack – Implementation – pop/peek




```
class StackX {
        private int maxSize;    // size of stack array
        private double[] stackArray;
        private int top;          //top of the stack

        public StackX(int s) { // constructor

                maxSize = s;       // set array size
                stackArray = new double[maxSize];
                top = -1;            // no items
        }
        public void push(double j) {

                // check whether stack is full
                if (top == maxSize - 1)
                        System.out.println("Stack is
full");
                else
                        stackArray[++top] = j;
        }
```

modifies the top pointer and return value

```
public double pop() {
        if (top == -1)
                return -99;
                else
                return stackArray[top--];
        }

public double peek() {
        if (top == -1)
                return -99;
                else
                return stackArray[top];
    }
```

checking

can be any value to show it is an error

if not full decrement

Question

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

isEmpty() method returns true if the stack is empty and isFull() method return true if the Stack is full.
Implement isEmpty() and isFull() methods of the stack class.

```
public boolean isFull(){
    if(maxSize == top){
       return true;
    }
    else{
       return false;
    }
}
```

```
public boolean isEmpty(){
    if(top==(-1)){
    return true;
    }
    else {
       return false;
    }
}
```

Creating a stack

Question

Using the implemented StackX class, Write a program to create a stack with maximum size 10 and insert the following items to the stack.

30 80 100 25

Delete all the items from the stack and display the deleted items.

Creating a stack

```
class StackApp {
        public static void main(String[] args) {
                StackX  theStack = new StackX(10);  // create a stack with max size 10

                theStack.push(30);  // insert given items
                theStack.push(80);
                theStack.push(100);
                theStack.push(25);

                while( !theStack.isEmpty() ) {    // until it is empty, delete item from stack

                        double val = theStack.pop();
                        System.out.print(val);
                        System.out.print(" ");
                }
        }
} // end of class
```

size

while stack is not empty

printing value

References

1. Mitchell Waite,Robert Lafore, Data Structures and Algorithms in Java,2nd Edition, Waite Group Press,1998.