What is a data structure?

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

Ex: stacks, queues, linked lists, trees

What is an algorithm?

Algorithms manipulate the data in these structures in various ways.

A sequence of steps we can follow to complete a certain 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.

Ex: searching and sorting algorithms

An algorithm should be

- correct.
- unambiguous.- should be only one meaning (the solution should be clear)
- give the correct solution for all cases.
- simple.
- terminate.

Stack

Operation policy - Always last in first out.

Application of Stacks

•String Reverse

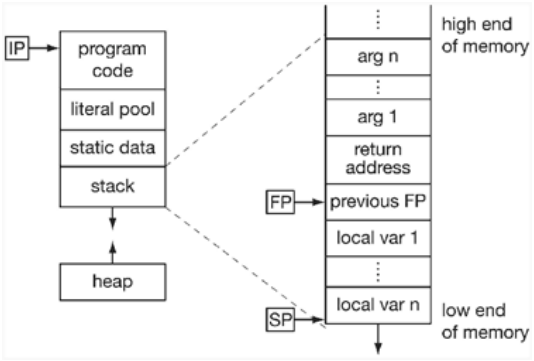
•Page visited history in Web browser.

•Undo sequence of text editor.

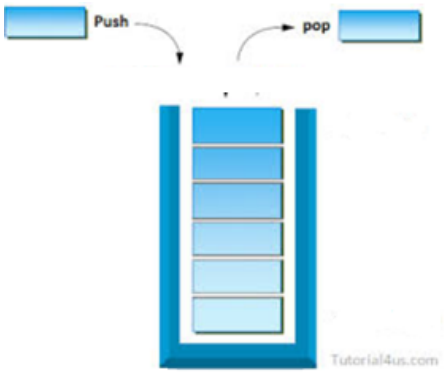
•Recursive function(functions that refer to the same function) calling.-

•Auxiliary data structure for Algorithms.

•Stack in memory for a process



As discussed under OSSA , the stack which stores function parameters , return values, temporary variables are stored can borrow memory from the heap.(unused storage)

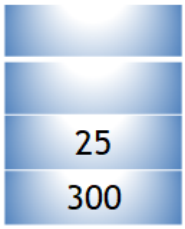


All insertions and deletions are done from the top
Restricted access to the middle or the end of the stack.
Add items- push
Remove items from the stack and return it-pop
To get the first element without deleting it from the stack - peek

When a new element is added , the pointer called the top pointer will update with the reference to the most recent element.

What is the difference between the pop and the peek?
Pop will remove the element from the stack and then return the element while peek will simply return the element.

(Q)



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

Since the stack size is 4 , if more than 4 elements are inserted, a error message will be given.

Uses of the stack

When a program is loaded, all the , function parameters , return values, temporary variables are stored in the stack
Stack operations are built into the microprocessor.

Stack Implementation

```
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];// creating a double array of the size and
initialing it to the variable
        top = -1;             // no items
       }

     public void push(double j) {
        //check the stack full
        // increment top
        // insert item
     }
      public double pop() {
          // check whether stack is empty
         // access item
         //decrement top
       }
       public double peek() {
          // check whether stack is empty
         // access item
       }
```

In this the order of incrementing the top variable matters,when inserting an item the top variable is subjected to a pre increment while when removing an item , the top varible is subjected to a post decrement.

```
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() {
  if (top == -1)
        return -99;
                 else
        return stackArray[top--];
     }
public double peek() {
  if (top == -1)
        return -99;
                 else
        return stackArray[top];
     }
Public boolean isFull(){
        if(top == maxSize - 1){
                return true;
        }
        else
                Return false;
}
```

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

Using the 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
```

```
// Stack.java
// demonstrates stacks
// to run this program: C>java StackApp
import java.io.*;                  // for I/O
////////////////////////////////////////////////////////////////
class StackX
   {
   private int maxSize;        // size of stack array
   private double[] stackArray;
   private int top;            // top of stack

//--------------------------------------------------------------
-
   public StackX(int s)         // constructor
      {
      maxSize = s;             // set array size
      stackArray = new double[maxSize];  // create array
      top = -1;                // no items yet
      }

//--------------------------------------------------------------
-
   public void push(double j)  // put item on top of stack
      {
      stackArray[++top] = j;   // increment top, insert item
      }

//--------------------------------------------------------------
-
   public double pop()         // take item from top of stack
      {
      return stackArray[top--]; // access item, decrement top
      }

//--------------------------------------------------------------
-
   public double peek()        // peek at top of stack
      {
      return stackArray[top];
      }

//--------------------------------------------------------------
-
   public boolean isEmpty()    // true if stack is empty
      {
```

```
      return (top == -1);
      }

//--------------------------------------------------------------
-
   public boolean isFull()     // true if stack is full
      {
      return (top == maxSize-1);
      }

//--------------------------------------------------------------
-
   }  // end class StackX

////////////////////////////////////////////////////////////////

class StackApp
   {
   public static void main(String[] args)
      {
      StackX theStack = new StackX(10);  // make new stack
      theStack.push(20);                 // push items onto stack
      theStack.push(40);
      theStack.push(60);
      theStack.push(80);

      while( !theStack.isEmpty() )       // until it's empty,
         {                               // delete item from
stack
         double value = theStack.pop();
         System.out.print(value);        // display it
         System.out.print(" ");
         }  // end while
      System.out.println("");
      }  // end main()

   }  // end class StackApp
```