BUILDING BINARY TREE DATA STRUCTURE

REQUIREMENT:

The requirement is to create a binay tree data structure having following details:

- Running a driver Script
- Perform CRUD Operations on Binary tree
  1. Add elements into the tree(multiple elements comma separated)
  2. Search an element
  3. Remove an element
- Perform Traversals in Binary Tree
  4. Print inorder Traversal
  5. Print preorder Traversal
  6. Print postorder Traversal
  7. Print level order Traversal
- Perform MinMax Element Search in Binary Tree
  8. Print the largest element
  9. Print the smallest element
- Find Root to leaf paths traversal
  10. Print all the paths i.e starting from root to the leaf
- Perform Input Output Related query
  11. Write all the elements to a file on quit command
  12. Option to start script with a file input and load BST with integers inside that file

APPROACH:

1. Creating a Driver Class :

This Class will start the initial script which will run until the user enters quit. The user will be given choices among the 10 options mentioned as a requirements inside the script. Also when the quit is entered a call for the I/O class (7^{th} approach) will be made which will save all the tree details in the file.

Driver class will look like below:

Pseudo Code :

```
Class Driver

  def start
    root=nil;                          # root of the binary tree
```

```
    scriptloop=”looping”            # for running the script until the user presses quit

    while scriptloop!=”quit”
      option=gets.chomp                        # user Input for different requirements
      case option
      when 1: Crud.add_Element(val , root)  #Take input val, add element in binary tree
      when 2: Crud.search_Element(val, root)       # will search element in binary tree
      when 3: Crud.remove_Element(val, root)      # will remove element from tree
      when 4: Traversal.in(root)
      when 5: Traversal.pre(root)
      when 6: Traversal.post(root)
      when 7: Traversal.level(root)
      when 8: MinMax.largest(root)
      when 9: MinMax.smallest(root)
      when 10: Paths.find(root)
      when 11: InputOutput.read_from_File(root,file)   # will read tree from file

      scriptloop=gets.chomp                       #srcipt input
    end of while loop
    InputOutput.write_In_File(root,file)            #when the user quits it writes in file
end
```

2. Creating a Data Structure for Tree : A class will be created with data members for value, left node, right node.

```
Class BinaryTree
  attr_reader :left, :right, :val

  def initialize(val)
    @val= val
    @left= nil
    @right= nil
```

3. Creating a Class for CRUD operation : This class will handle the 1st, 5th, 6th cases which involvels modification in the binary tree.

```
Class CRUD
 def self.add_Element(element, root)       #will accept root of tree and add element


  def self.remove_Element(element, root)  # remove a element


  def self.search_Element(element, root)    # search for an element
```

4. Creating a Class for Traversal : It will contain 4 methods for each Inorder, Preorder, Postorder and levelorder Traversal. Its member functions will be initialized with root of the tree for the data member.

```
Class Traversal
  def self.pre(root)                        #  perform preorder traversal

  def self.post(root)                       #  perform postorder traversal
```

```
def self.in(root)                    # perform inorder traversal

def self.level(root)                 #  perform levelorder traversal
```

5. Creating a MinMax Class for finding minimum and maximum elements : It will contain 2 member functions for finding minimum and maximum elements. Its member functions will be initialized with root of the Tree.

```
Class MINMAX

  def self.largest(root)

  def self.smallest(root)
```

6. Creating a pathVisitor Class for printing all root to leaf path : It will contain a single member function for finding and printing all the path from root to leaf.

```
Class Path

  def self.find(root)                    # find all root to leaf node path
```

7. Creating a Class for file I/O related cases : It will have 2 member functions for reading from the file or for writing into the file for the tree input.

```
Class InputOutput

  def self.write_In_File(root, nameOfFile)     # will save the tree details in a file

  def self.read_From_File(root, nameOfFile) # will read tree from file if found, else
                                              exception
```

Note : Some basic operations will be handled by another utility class. Also user will create an object for the driver class and then input/output for user requests will occur.