A Mini Project Report
on

Programming Language With Data Structures Visualization

Course: DS Lab
CLASS: B.E CSE-B Sem: III

By

Nishanth .T

1602-21-733-102

Siddhartha Reddy .D

1602-21-733-118

Department of Computer Science & Engineering

Vasavi College of Engineering (Autonomous)
Ibrahimbagh, Hyderabad-31
2023

ACKNOWLEDGEMENT

We take this opportunity with pride and enormous gratitude, to express the deeply embedded feeling and gratefulness to our respectable guide **Dr. V.Sireesha ma'am,** Department of Computer Science and Engineering, whose guidance was unforgettable and filled with innovative ideas as well as her constructive suggestions has made the presentation of our mini project for the **3rd Semester** a grand success.

We are thankful to **Dr. T. Adilakshmi,** Head of Department (CSE), **Vasavi College of Engineering** for the help during our course work.

Finally we express our gratitude to the management of our college, **Vasavi College of Engineering** for providing the necessary arrangements and support to complete our project work successfully.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Data Structures are necessary for storing, accessing and performing various operations such as insertion, deletion and updation in an efficient manner. Most Data Structures such as Binary Search Trees and Linked Lists are difficult to understand and implement without a proper visualization of how data is manipulated. With the help of the Data Structures Visualization Tool, the different operations which can be performed are visualized in a step-by-step manner so that the user can clearly understand how they work without any misunderstanding. The user can also call their own methods by typing in a custom and simple programming language to work on different Data Structures. By using a simple language, the programming language-specific barrier of syntax and semantics can be reduced significantly so that the user can focus solely on the Data Structures and their functioning.

1. INTRODUCTION

Data Structures are one of the most important topics for anyone interested in working in a Computer Science related field. It just so happens that data structures can be incredibly difficult and stressful to understand clearly. This prevents many from effectively using data and performing operations which are of minimum Time and Space Complexity. By visualizing the Data Structures, users can get a clear understanding of what actually happens and this can help them better understand their implementation.

The Data Structures Visualization Tool can be used by anyone who is familiar with programming and does not require understanding complex syntaxes and structures. The user can easily program their required Data Structures and perform different operations on them. After coding, they can look at the Visualization of all the Data Structures mentioned and different operations performed in a step-by-step manner. The user will be able to understand how they work by looking at visual representations instead of the implementation code itself, and by using the simple programming language to create them, they will become more familiar with implementing different Data Structures.

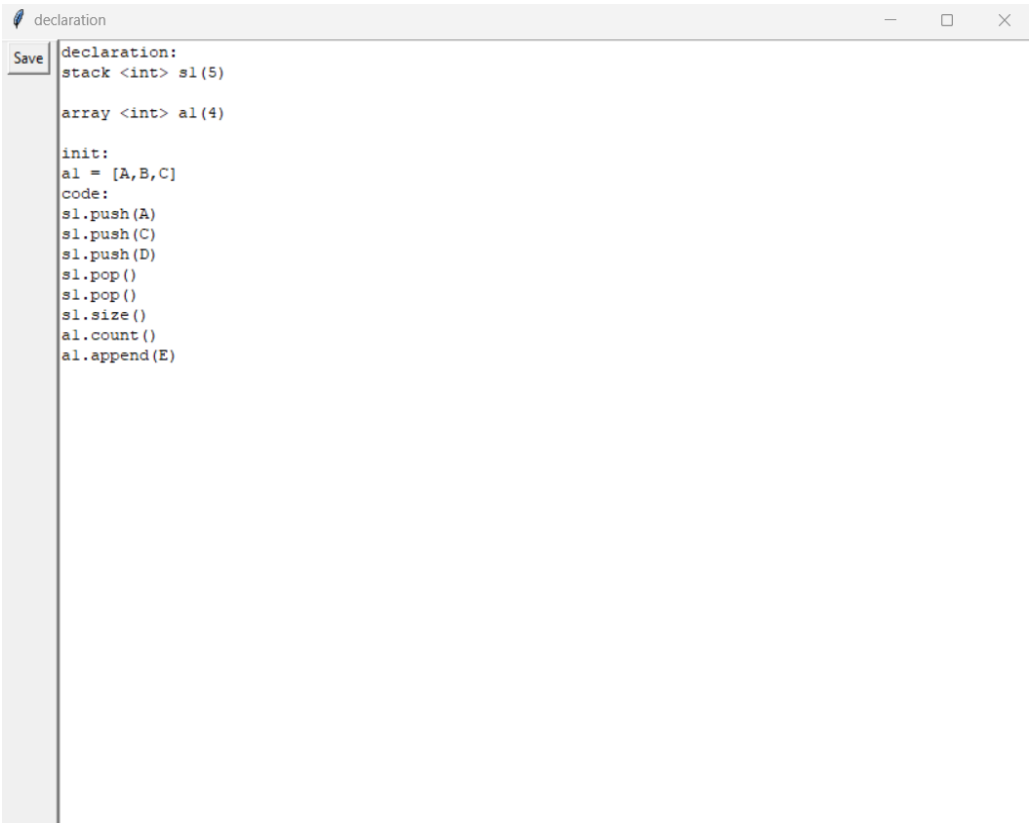

Fig 1. Programming Data Structures

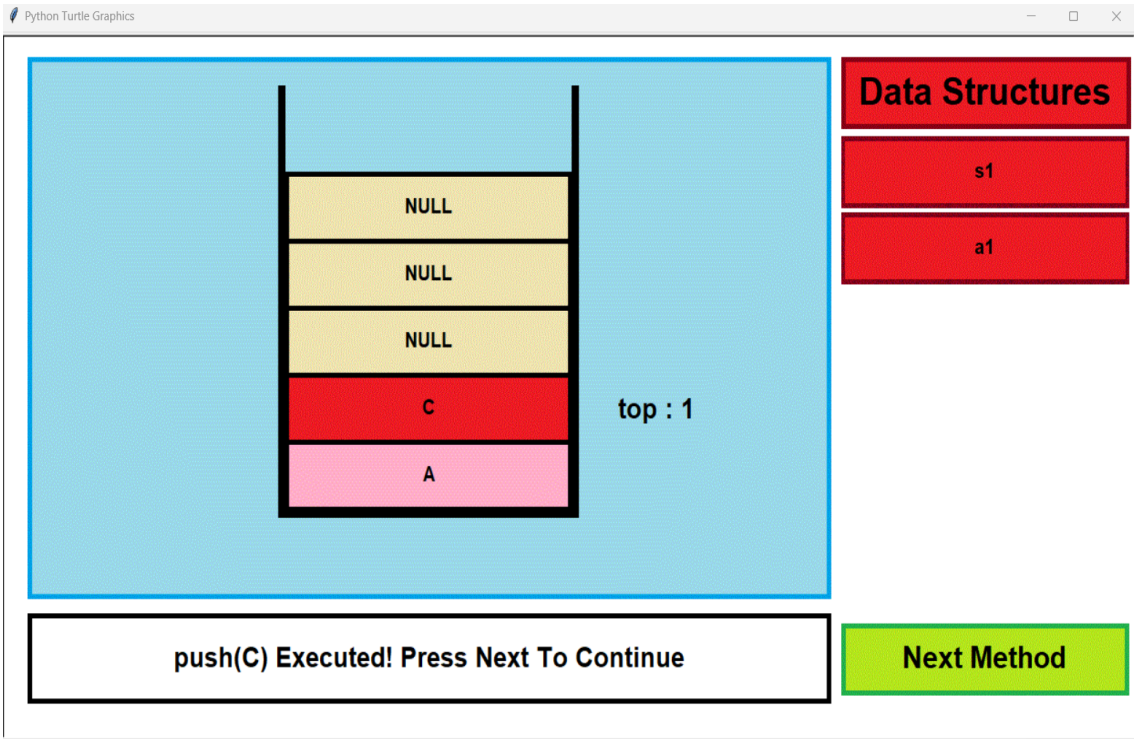

Fig 2. Data Structures Visualization

1.1. OVERVIEW

Data Structures are important to learn because they are most frequently asked in interviews, coding rounds and online assessments. They can be hard to understand without a proper visualization tool to guide the user on how different operations affect various parameters of a structure. Along with visualization, if the user can interact with the tool by typing custom programs of their liking, they can better understand it and clear any doubts that they may have. The programming language should be simple so that there are no unnecessary barriers to learning Data Structures. We have created a tool with a simple programmable interface which the user can type in custom programs along with a visualization tool to provide a graphical representation of all Data Structures and functions mentioned.

1.2. MOTIVATION

There exists a need to provide visualization of Data Structures so that people can better understand them. Many university students find it difficult to implement different Data Structures and do not understand how different aspects of the structure are manipulated for every operation. There are a few visualization tools available, but they require a Network Connection, and/or implement a subscription based program. In some tools, the user cannot program their own values into the Data Structure and they only work on a random set of values. Hence, a tool with the ability to be used offline, be easy to understand, and have a programmable interface for custom programs is required.

1.3. PROBLEM STATEMENT

The problem statement is to provide a Data Visualization Tool that teaches the user how a Data Structure is manipulated and a Programmable Interface in which the user can create their own custom programs and visualize them using the tool. We have created a tool which includes Data Structures such as Stack, Queue, Linked List, Binary Tree and Binary Search Tree. We have also included all the methods from their respective ADT's and implemented a Programmable Interface. It is simple to understand and requires the user to declare a Data Structure, with a variable, and use it to call different methods on the respective Data Structure. The user can optionally initialize the Data Structure with some initial values before calling the required methods.

1.4. PROJECT OBJECTIVES

The objectives of our project include providing a Data Structures and Visualization Tool which provides a simple and easily understandable visual representation of different operations performed on various Data Structures along with a Programmable Interface so that the user can type in their required programs and use the required Data Structures.

1.5. PROJECT SCOPE

With this project, our main aim is to provide a tool which can be used for visualizing different Data Structures. It is incredibly hard for teachers to explain different structures without visualization and just through code snippets. Most students find the Data Structures course to be intimidating and difficult to understand. With the help of a visualization tool, Students will be able to understand different Structures and clear any misunderstandings.

With the provision of a Programmable Interface, students will be able to create their own custom programs and visualize them on a screen. This makes the tool more interactive and different operations can be performed on custom data. This tool is necessary for helping students and anyone interested to understand how different Data Structures operate on data.

2. SOFTWARE REQUIREMENT SPECIFICATION

The user is required to install the latest version of Python, and the following Python Modules,

- Turtle
- Tkinter
- Cmd
- Clrprint

Along with the above-mentioned modules, a Python Interpreter, such as IDLE, is required to provide an interactive Programmable Interface, File Management System, and a Data Structures Visualization Tool to represent the following Data Structures,

- Array
- Stack
- Queue
- Linked List
- Binary Tree
- Binary Search Tree

3. SYSTEM DESIGN

The project was implemented using Python's Turtle module, a tool for producing interactive GUI by using "turtles" which can be repositioned, and modified to provide different elements of the UI. A list of turtles has been assigned to depict the different elements of a Data Structure, with every turtle being assigned an image to depict on the screen. There are different Classes which encapsulate all the attributes and methods of a Data Structure, with each method manipulating the List of Turtles in such a way that it represents the specific task being performed.

Using Python's Tkinter module, a File Management System has been created which the user can utilize to create new scripts, remove existing scripts, edit scripts, view the contents of a script, and visualize the script.

The programs written are converted into tokens using Regular Expressions, and based on the type of token, the required Data Structures are created, and respective methods are called during visualization.

4. PROJECT IMPLEMENTATION

```
class Array_Turtle:
    def __init__(self, x=0, y=0, data = "NULL"):
        self.data = data
        self.turtle = t.Turtle()
        self.turtle.turtlesize(40)
        self.pen = t.Turtle()
        self.turtle.up()
        self.pen.up()
        self.pen.ht()
        self.turtle.shape("arr"+str(random.randint(1, 9))+".gif")
        self.move(x, y)
        self.turtle.onclick(fun = self.get_data)
        self.pen.onclick(fun = self.get_data)
    def move(self, x, y):
        self.turtle.setposition(x, y)
        self.pen.setposition(x, y-11)
    def write(self, msg):
        self.pen.write(msg, align = "center", font = ("Arial", 15, "bold"))
    def clr(self):
        self.pen.clear()
        self.turtle.ht()
    def hide(self):
        self.pen.ht()
        self.turtle.ht()
    def set_data(self, data):
        self.data = data
    def change_color(self):
        self.turtle.shape("arr"+str(random.randint(1, 9))+".gif")
    def get_data(self, x, y):
        self.turtle.onclick(fun = None)
        self.pen.onclick(fun = None)
        write(self.data)
        time.sleep(3)
        undo()
        self.turtle.onclick(fun = self.get_data)
        self.pen.onclick(fun = self.get_data)
```

Implementation of Array Template

```
class Stack_Turtle:
    def __init__(self, x=0, y=0, data = "NULL"):
        self.data = data
        self.turtle = t.Turtle()
        self.turtle.turtlesize(40)
        self.pen = t.Turtle()
        self.turtle.up()
        self.pen.up()
        self.pen.ht()
        self.turtle.shape("stack-1.gif")
        self.move(x, y)
        self.turtle.onclick(fun = self.get_data)
        self.pen.onclick(fun = self.get_data)
    def move(self, x, y):
        self.turtle.setposition(x, y)
        self.pen.setposition(x, y-11)
    def write(self, msg):
        self.pen.write(msg, align = "center", font = ("Arial", 15, "bold"))
    def clr(self):
        self.pen.clear()
        self.turtle.ht()
    def hide(self):
        self.pen.ht()
        self.turtle.ht()
    def set_data(self, data):
        self.data = data
    def get_data(self, x, y):
        self.turtle.onclick(fun = None)
        self.pen.onclick(fun = None)
        write(self.data)
        time.sleep(3)
        undo()
        self.turtle.onclick(fun = self.get_data)
        self.pen.onclick(fun = self.get_data)
```

Implementation of Stack Template

```
class Linked_Turtle:
    def __init__(self, x=0, y=0, data = "NULL", address = "NULL", next_node = "NULL"):
        self.data = data
        self.address = address
        self.next = next_node
        self.turtle = t.Turtle()
        self.link = t.Turtle()
        self.link.width(7)
        self.turtle.turtlesize(40)
        self.data_pen = t.Turtle()
        self.next_address_pen = t.Turtle()
        self.address_pen = t.Turtle()
        self.turtle.up()
        self.data_pen.up()
        self.next_address_pen.up()
        self.address_pen.up()
        self.link.up()
        self.data_pen.ht()
        self.next_address_pen.ht()
        self.address_pen.ht()
        self.link.ht()
        self.turtle.shape("linked"+str(random.randint(1, 9))+".gif")
        self.move(x, y)
        self.turtle.onclick(fun = self.get_data)
        self.data_pen.onclick(fun = self.get_data)
        self.next_address_pen.onclick(fun = self.get_data)
        self.address_pen.onclick(fun = self.get_data)
    def move(self, x, y):
        self.turtle.setposition(x, y)
        self.data_pen.setposition(x-31, y-11)
        self.next_address_pen.setposition(x+17, y-11)
        self.address_pen.setposition(self.turtle.xcor(), self.turtle.ycor()-70)
        self.link.setposition(x+49, y)
    def get_data(self, x, y):
        self.turtle.onclick(fun = None)
        self.data_pen.onclick(fun = None)
        self.next_address_pen.onclick(fun = None)
        self.address_pen.onclick(fun = None)
        write(str(self.address)+" :: "+str(self.data)+" -> "+str(self.next))
        time.sleep(3)
        undo()
        self.turtle.onclick(fun = self.get_data)
        self.data_pen.onclick(fun = self.get_data)
        self.next_address_pen.onclick(fun = self.get_data)
        self.address_pen.onclick(fun = self.get_data)
```

```
    def write(self, index = -1):
        self.data_pen.clear()
        self.address_pen.clear()
        self.next_address_pen.clear()
        self.data_pen.write(str(self.data)[:2], align = "center", font = ("Arial", 15, "bold"))
        self.next_address_pen.write(str(self.next)[:4], align = "center", font = ("Arial", 15, "bold"))
        self.address_pen.write(str(self.address), align = "center", font = ("Arial", 15, "bold"))
    def clr(self):
        self.hide()
        self.turtle.clear()
        self.data_pen.clear()
        self.link.clear()
        self.next_address_pen.clear()
        self.address_pen.clear()
    def hide(self):
        self.data_pen.ht()
        self.turtle.ht()
        self.address_pen.ht()
        self.next_address_pen.ht()
    def set_data(self, data, address):
        self.data = data
        self.address = address
        self.write(data, address)
    def link_draw(self):
        if(self.next != None):
            self.link.down()
            self.link.forward(20)
            self.link.up()
            self.link.backward(20)
```

Implementation of Linked List Template

```
class Tree_Turtle:
    def __init__(self, x=0, y=0, data = "NULL"):
        self.data = data
        self.turtle = t.Turtle()
        self.link = t.Turtle()
        self.link.width(7)
        self.turtle.turtlesize(40)
        self.pen = t.Turtle()
        self.turtle.up()
        self.pen.up()
        self.link.up()
        self.link.right(90)
        self.pen.ht()
        self.link.ht()
        self.turtle.shape("tree"+str(random.randint(1, 9))+".gif")
        self.move(x, y)
        self.turtle.onclick(fun = self.get_data)
        self.pen.onclick(fun = self.get_data)
    def move(self, x, y):
        self.turtle.setposition(x, y)
        self.pen.setposition(x, y-11)
        self.link.setposition(x, y)
    def get_data(self, x, y):
        self.turtle.onclick(fun = None)
        self.pen.onclick(fun = None)
        write(str(self.data))
        time.sleep(3)
        undo()
        self.turtle.onclick(fun = self.get_data)
        self.pen.onclick(fun = self.get_data)
    def write(self, index = -1):
        self.pen.clear()
        self.pen.write(str(self.data), align = "center", font = ("Arial", 15, "bold"))
    def clr(self):
        self.pen.clear()
        self.turtle.ht()
        self.link.clear()
    def hide(self):
        self.data_pen.ht()
        self.turtle.ht()
    def set_data(self, data):
        self.data = data
        self.write(data)
```

Implementation of Tree Template

```
def do_visualize(self, inp):
      """Compile and Show Visualization"""
      global scripts, cur_name, txt_edit, instructions, instructiondeclaration
      instructiondeclaration.clear()
      scripts = os.listdir()
      clrprint("\nExisting Scripts:", clr = "blue")
      for i in scripts:
            if(".py" in i[-3:]):
               clrprint(i[:-3], clr = "blue")
      clrprint("Enter Script To Visualize:", clr = "blue")
      name = input()
      name += ".py"
      if name not in scripts:
         clrprint("Error: File Not Found!", clr = "blue")
      else:
         cur_name = name
      instructions=[(re.sub(r'\s+','',i)) for i in filter(lambda x : x.strip(),open(os.getcwd()+"/"+name,"r").readlines())]
      for i in range(len(instructions)):
          instructiondeclaration.append(instructions[i])
          if(instructions[i+1]=="init:" or instructions[i+1]=="Init:" or instructions[i+1]=="INIT:"):
              break
      declaration = instructiondeclaration[1:]
      instructioninitialisation=[]
      for i in instructions:
          if i not in instructiondeclaration:
              if i=="Code:" or i=="code:" or i=="CODE:":
                  break
              instructioninitialisation.append(i)
      initialisation=instructioninitialisation[1:]
      code=[i for i in instructions if i not in instructiondeclaration and i not in instructioninitialisation][1:]
      structures=[i.split('<')[0] for i in declaration]
      objects=[i.split(">")[1].split("(")[0] for i in declaration]
      data_types=[i.split("<")[1].split(">")[0] for i in declaration]
      capacity=[i.split("(")[1].split(")")[0] for i in declaration]
      methods={}
      for i in code:
          j=i.split('.')[0]
          k=i.split('.')[1]
          if j in methods:
              methods[j].append(k)
          else:
              methods[j]=[k]
      final={}
      for i in range(len(structures)):
          if structures[i] in final:
              final[structures[i]].append([objects[i],data_types[i],capacity[i]])
          else:
              final[structures[i]]=[[objects[i],data_types[i],capacity[i]]]
      for key1, value1 in methods.items():
          [v.append(value1) for key2, values in final.items() for v in values if key1 == v[0]]
      run(objects, final)
```

Implementation of Programmable Interface

5. RESULTS

The Visualization of different Data Structures has been achieved along with the implementation of a Programmable Interface to program different Structures and call their respective methods.

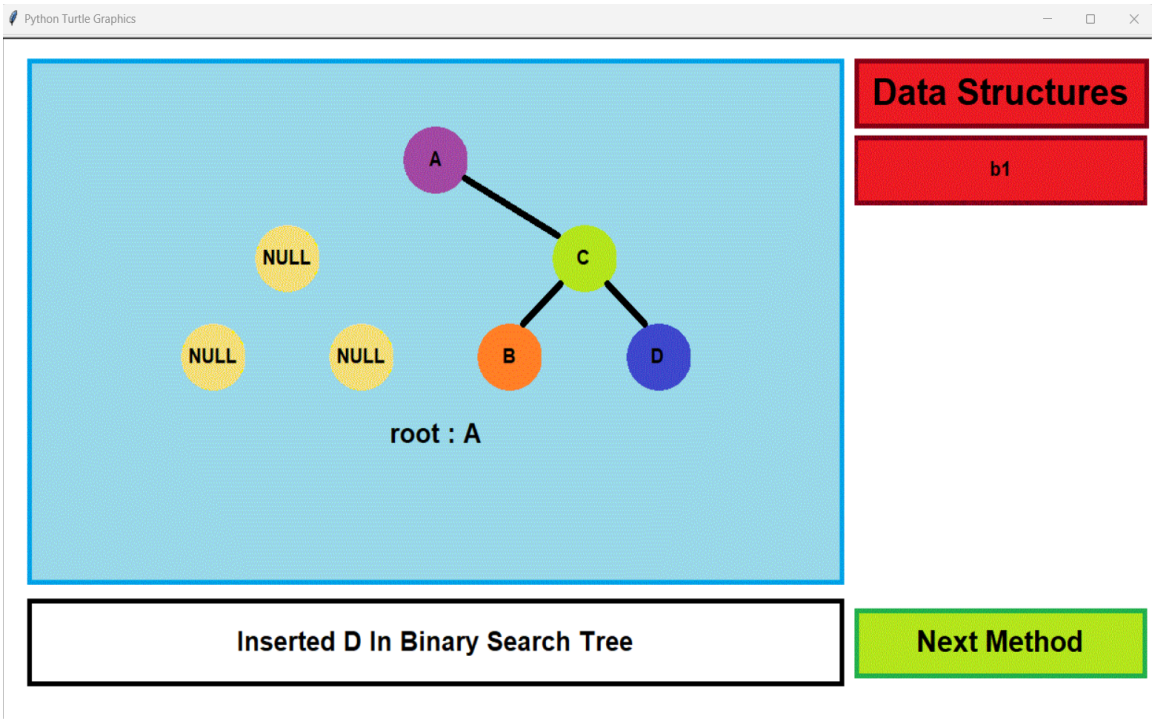

Fig 3. Binary Search Tree Visualization

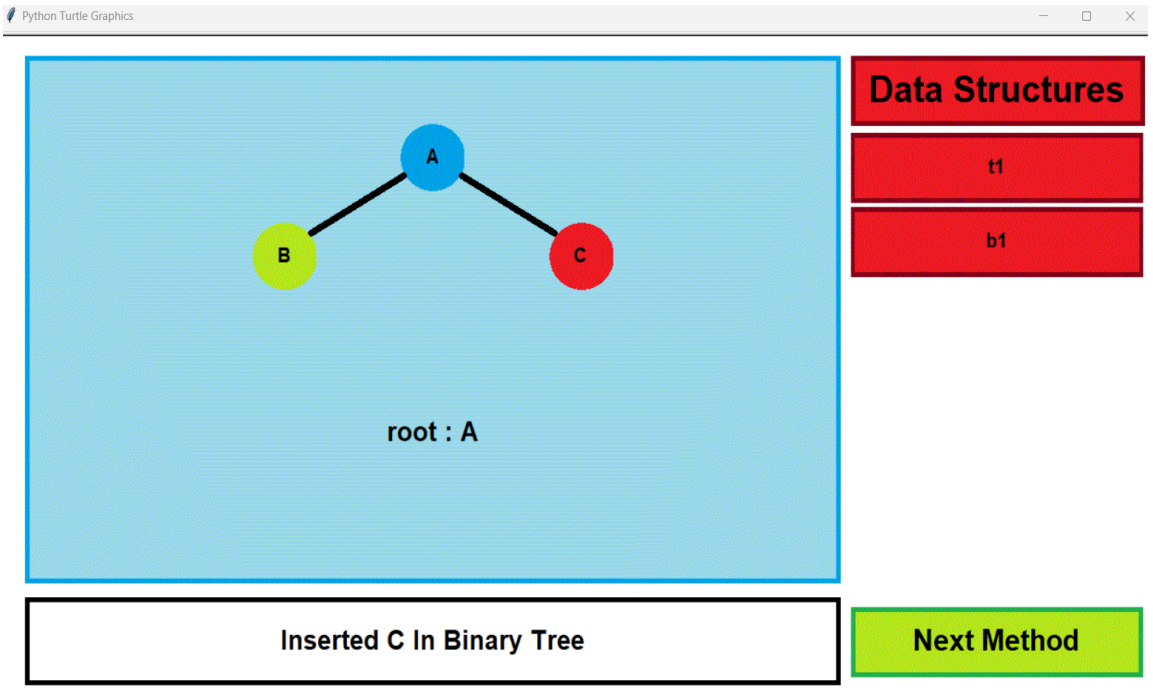

Fig 4. Binary Tree Visualization

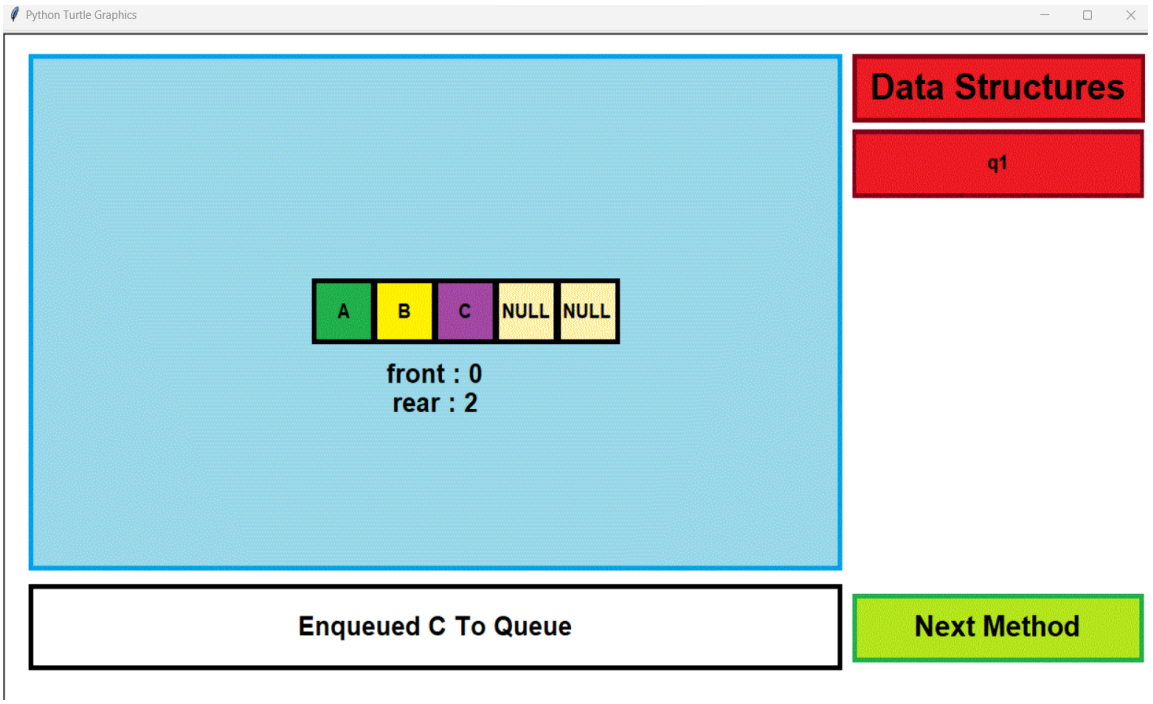

Fig 5. Queue Visualization

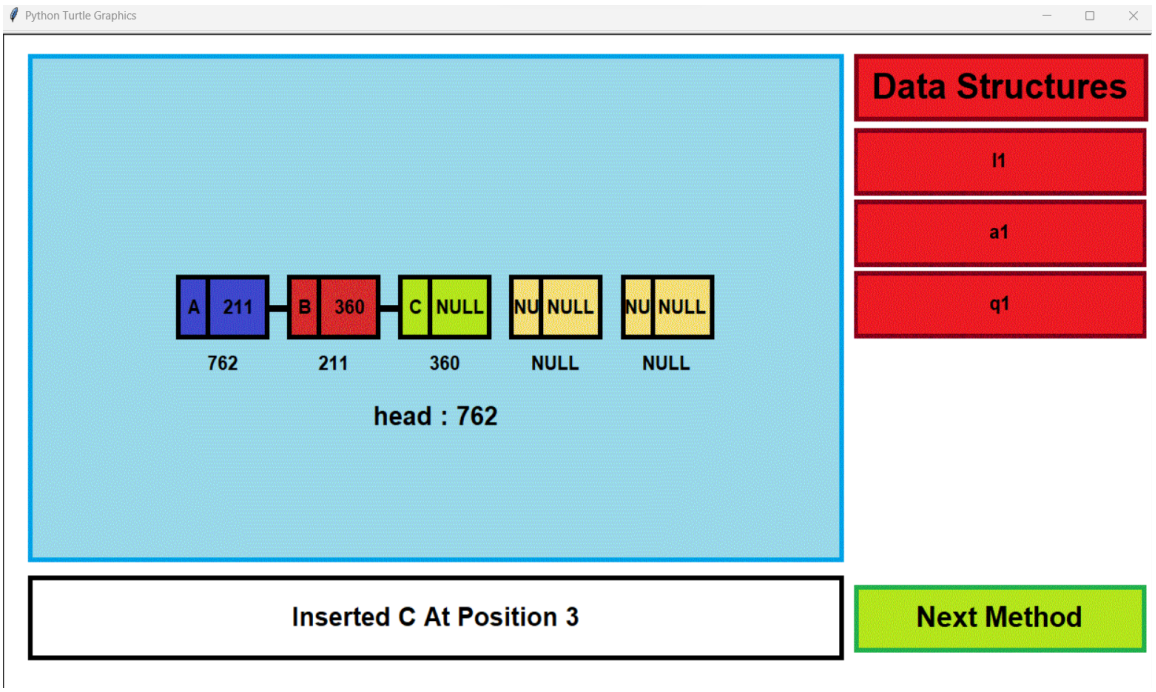

Fig 6. Linked List Visualization

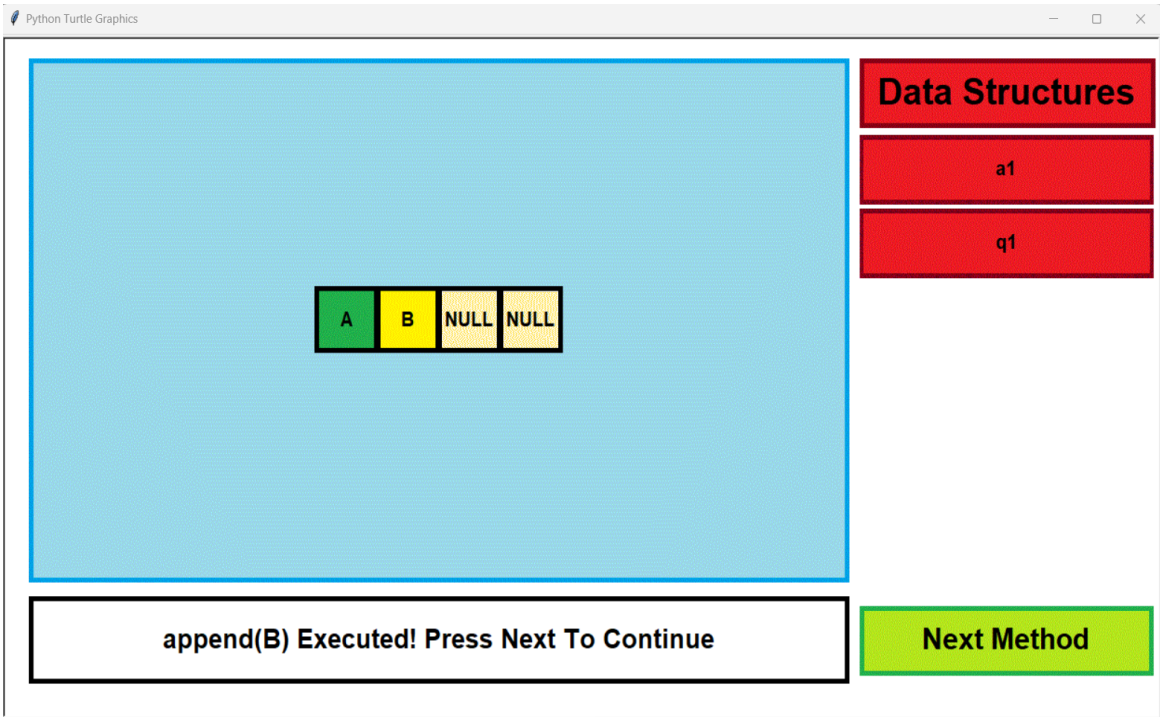

Fig 7. Array Visualization

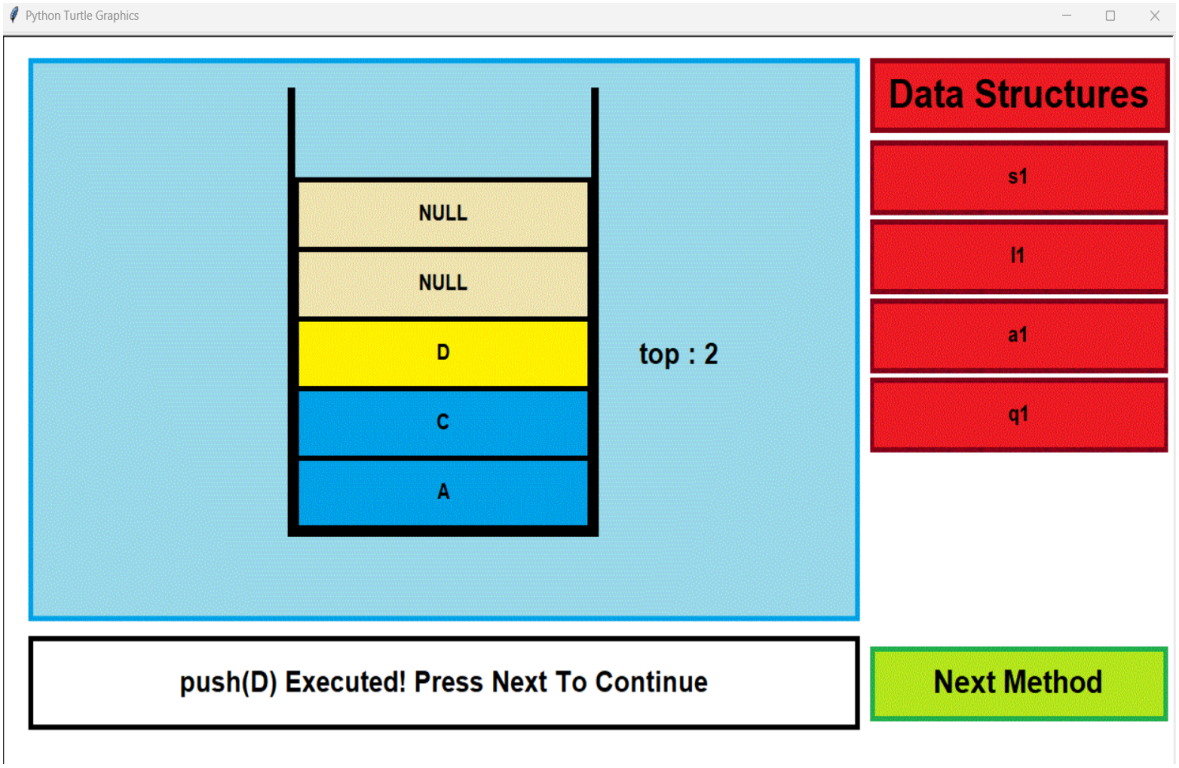

Fig 8. Stack Visualization

The above images represent every Data Structure that has been implemented in the Visualization Tool, with the representation of certain attributes of the Structure, such as “top” in the case of a Stack. A list of declared Data Structure objects is depicted on the right along with a “Next” button to continue the Visualization. There is a Text Box at the bottom which informs the user about the operation being performed and the various sub-operations required to perform it.

6. TESTING

The tool has been equipped with different exception handling mechanisms to mitigate any unexpected outcomes such as an inserting into a full Array, or popping an empty Stack, and many other exceptions.

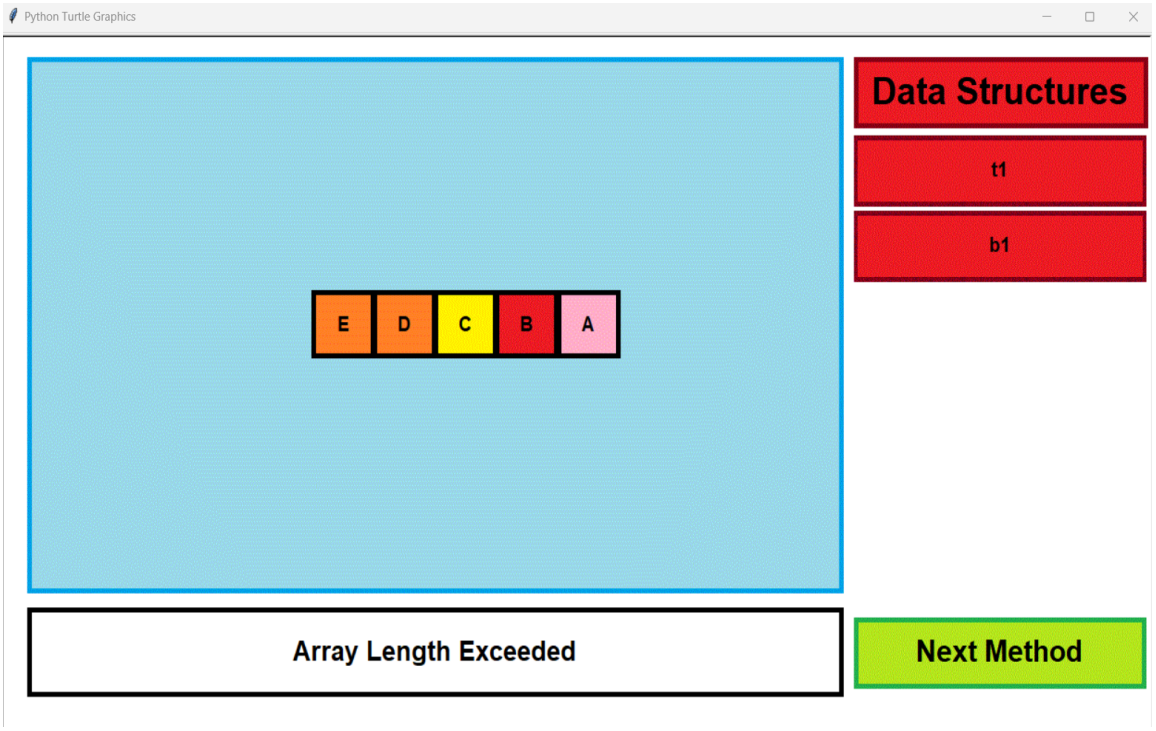

Fig 9. Inserting Into a Full Array

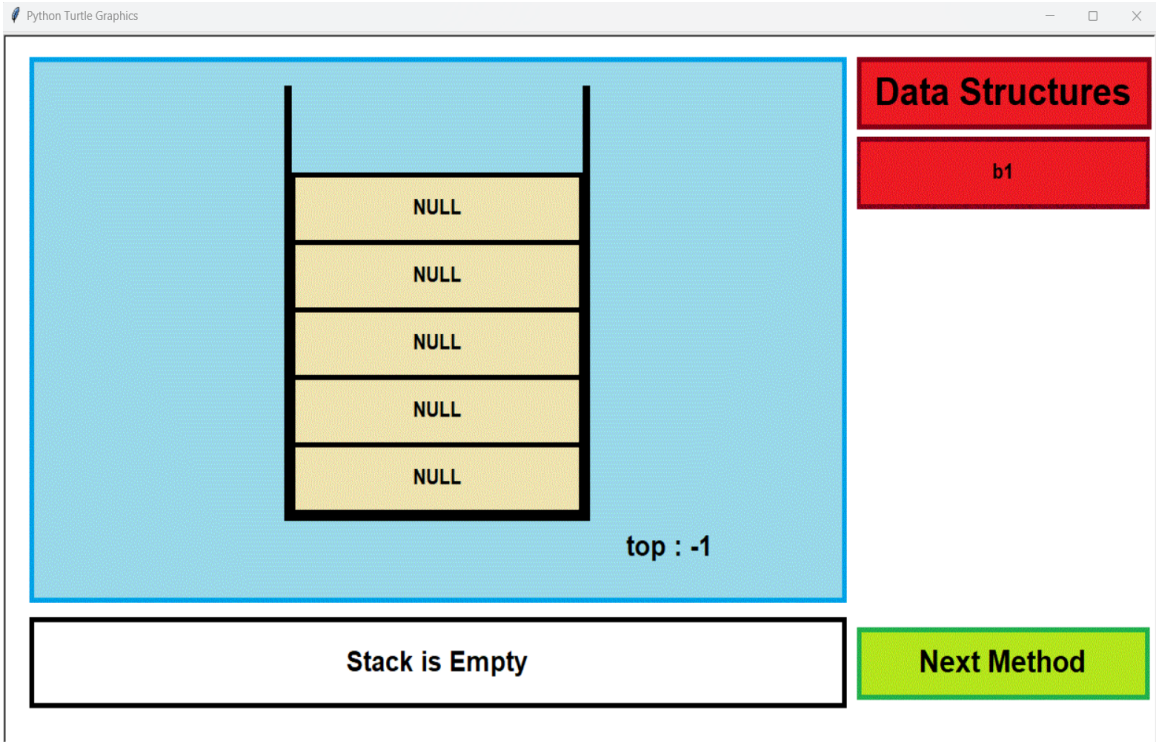

Fig 10. Popping an Empty Stack

7. CONCLUSION AND FUTURE WORK

The tool we created accurately represents the mentioned Data Structures along with their operations in a simple graphical format. All the methods work as expected and handle exceptions as well. The Structures appear in the sequence in which they were declared and the user can move through every Structure and its methods in a sequential manner. Any operation being performed is described in a text box along with sub-operations such as comparisons and checks in order to make it user-friendly. Every template of data has been assigned a random color whenever it is given a value which makes the tool more attractive. The Programmable Interface can compile all the methods which have been implemented and makes the tool more interactive so that the user can code their own programs.

The project's purpose was to develop a Visualization tool for simple visualizations. It currently only works on String data types and does comparisons based on ASCII values. Other primitive data types such as Integer, Float and Boolean can be added to make the tool more diverse. It can be extended to implement other complicated Data Structures such as Red-Black Trees, AVL Trees, and B-Trees. Extensive Exception Handling can be implemented in the Programmable Interface to make the programming experience much more user-friendly.

8. REFERENCES

[1] https://docs.python.org/3/library/turtle.html [Accessed: 26-01-23].

[2] https://docs.python.org/3/library/tkinter.html [Accessed: 26-01-23].

[3] https://www.geeksforgeeks.org/array-data-structure/ [Accessed: 26-01-23].

[4] https://www.geeksforgeeks.org/stack-data-structure/ [Accessed: 26-01-23].

[5] https://www.geeksforgeeks.org/queue-data-structure/ [Accessed: 26-01-23].

[6] https://www.geeksforgeeks.org/data-structures/linked-list

[Accessed: 26-01-23].

[7] https://www.geeksforgeeks.org/binary-tree-data-structure/

[Accessed: 26-01-23].

[8] https://www.geeksforgeeks.org/binary-search-tree-data-structure/

[Accessed: 26-01-23].