Practical No.1
Data Structures

-Abin Pillai S103

AIM: Write a program to implement Abstract Data Types (ADT)

(Code Using Graphical User Interface)

```
import tkinter as tk
from tkinter import messagebox, simpledialog
class Stack:
    def __init__(self):
        self.items = []
    def is_empty(self):
        return len(self.items) == 0
    def push(self, item):
        self.items.append(item)
        return f"'{item}' has been pushed onto the stack."
    def pop(self):
        if self.is_empty():
            raise IndexError("Pop from an empty stack")
        item = self.items.pop()
        return f"'{item}' has been popped from the stack."
    def peek(self):
        if self.is_empty():
            raise IndexError("Peek from an empty stack")
        return self.items[-1]
    def size(self):
        return len(self.items)
    def __str__(self):
        return " <- ".join(reversed(self.items)) if self.items else "Stack is empty"
```

Practical No.1
Data Structures

-Abin Pillai S103

```
class StackApp(tk.Tk):
    def __init__(self):
        super().__init__()
        self.stack = Stack()
        self.title("Stack Operations GUI")
        self.geometry("900x700")
        self.configure(bg='#282c34')
        self.create_widgets()
    def create_widgets(self):
        self.title_label = tk.Label(self, text="Stack Operations", font=("Helvetica", 24, "bold"), fg="#61dafb", bg='#282c34')
        self.title_label.pack(pady=20)
        self.stack_label = tk.Label(self, text=str(self.stack), font=("Helvetica", 18), fg="#ffffff", bg='#282c34')
        self.stack_label.pack(pady=20)
        button_frame = tk.Frame(self, bg='#282c34')
        button_frame.pack(pady=10)
        button_font = ("Helvetica", 14)
        button_bg = "#61dafb"
        button_fg = "#282c34"
        self.push_button = tk.Button(button_frame, text="Push", font=button_font, bg=button_bg, fg=button_fg, command=self.push_item, width=12, height=2)
        self.push_button.pack(side=tk.LEFT, padx=10)
        self.pop_button = tk.Button(button_frame, text="Pop", font=button_font, bg=button_bg, fg=button_fg, command=self.pop_item, width=12, height=2)
        self.pop_button.pack(side=tk.LEFT, padx=10)
```

Practical No.1
Data Structures

-Abin Pillai S103

```
self.peek_button = tk.Button(button_frame, text="Peek", font=button_font, bg=button_bg, fg=button_fg, command=self.peek_item, width=12, height=2)

self.peek_button.pack(side=tk.LEFT, padx=10)

self.is_empty_button = tk.Button(button_frame, text="Is Empty", font=button_font, bg=button_bg, fg=button_fg, command=self.check_is_empty, width=12, height=2)

self.is_empty_button.pack(side=tk.LEFT, padx=10)

self.size_button = tk.Button(button_frame, text="Size", font=button_font, bg=button_bg, fg=button_fg, command=self.check_size, width=12, height=2)

self.size_button.pack(side=tk.LEFT, padx=10)

self.quit_button = tk.Button(button_frame, text="Quit", font=button_font, bg=button_bg, fg=button_fg, command=self.quit, width=12, height=2)

self.quit_button.pack(side=tk.LEFT, padx=10)

bottom_frame = tk.Frame(self, bg='#282c34')

bottom_frame.pack(fill="both", expand=True, padx=20, pady=10)

# Description frame

description_frame = tk.LabelFrame(bottom_frame, text="Description", font=("Helvetica", 14, "bold"), fg="#61dafb", bg='#282c34', bd=2, padx=10, pady=10)

description_frame.pack(side=tk.TOP, fill="both", expand=True, padx=10, pady=10)

description_text = ("A Stack is a linear data structure that follows a particular order in which the operations are performed. "

                    "The order may be LIFO (Last In First Out) or FILO (First In Last Out). LIFO implies that the element that is inserted last, "

                    "comes out first and FILO implies that the element that is inserted first, comes out last.")
```

Practical No.1
Data Structures

-Abin Pillai S103

```
    description_label = tk.Label(description_frame, text=description_text, font=("Helvetica", 12), fg="#ffffff", bg='#282c34', wraplength=400, justify="center")

    description_label.pack(anchor="center")

    # Advantages frame

    advantages_frame = tk.LabelFrame(bottom_frame, text="Advantages of Stacks", font=("Helvetica", 14, "bold"), fg="#61dafb", bg='#282c34', bd=2, padx=10, pady=10)

    advantages_frame.pack(side=tk.LEFT, fill="both", expand=True, padx=10, pady=10)

    advantages_text = ("Simplicity: Stacks are a simple and easy-to-understand data structure, making them suitable for a wide range of applications.\n"

                       "Efficiency: Push and pop operations on a stack can be performed in constant time (O(1)), providing efficient access to data.\n"

                       "Last-in, First-out (LIFO): Stacks follow the LIFO principle, ensuring that the last element added to the stack is the first one removed. This behavior is useful in many scenarios, such as function calls and expression evaluation.\n"

                       "Limited memory usage: Stacks only need to store the elements that have been pushed onto them, making them memory-efficient compared to other data structures.")

    advantages_label = tk.Label(advantages_frame, text=advantages_text, font=("Helvetica", 12), fg="#ffffff", bg='#282c34', wraplength=400, justify="center")

    advantages_label.pack(anchor="center")

    # Operations frame

    operations_frame = tk.LabelFrame(bottom_frame, text="Key Operations on Stack Data Structures", font=("Helvetica", 14, "bold"), fg="#61dafb", bg='#282c34', bd=2, padx=10, pady=10)
```

Practical No.1
Data Structures

-Abin Pillai S103

```
    operations_frame.pack(side=tk.RIGHT, fill="both", expand=True, padx=10, pady=10)

    operations_text = ("Push: Adds an element to the top of the stack.\n"
                       "Pop: Removes the top element from the stack.\n"
                       "Peek: Returns the top element without removing it.\n"
                       "IsEmpty: Checks if the stack is empty.\n"
                       "IsFull: Checks if the stack is full (in case of fixed-size arrays).")

    operations_label = tk.Label(operations_frame, text=operations_text, font=("Helvetica", 12), fg="#ffffff", bg='#282c34', justify="center", anchor="center")

    operations_label.pack(anchor="center")

    # Disadvantages frame

    disadvantages_frame = tk.LabelFrame(bottom_frame, text="Disadvantages of Stacks", font=("Helvetica", 14, "bold"), fg="#61dafb", bg='#282c34', bd=2, padx=10, pady=10)

    disadvantages_frame.pack(side=tk.BOTTOM, fill="both", expand=True, padx=10, pady=10)

    disadvantages_text = ("Limited access: Elements in a stack can only be accessed from the top, making it difficult to retrieve or modify elements in the middle of the stack.\n"
                          "Potential for overflow: If more elements are pushed onto a stack than it can hold, an overflow error will occur, resulting in a loss of data.\n"
                          "Not suitable for random access: Stacks do not allow for random access to elements, making them unsuitable for applications where elements need to be accessed in a specific order.\n"
                          "Limited capacity: Stacks have a fixed capacity, which can be a limitation if the number of elements that need to be stored is unknown or highly variable.")
```

```
        disadvantages_label = tk.Label(disadvantages_frame,
text=disadvantages_text, font=("Helvetica", 12), fg="#ffffff", bg='#282c34',
wraplength=400, justify="center")
        disadvantages_label.pack(anchor="center")
    def update_stack_display(self):
        self.stack_label.config(text=str(self.stack))
    def push_item(self):
        item = simpledialog.askstring("Input", "Enter an item to push:",
parent=self)
        if item:
            message = self.stack.push(item)
            messagebox.showinfo("Push", message, parent=self)
            self.animate_push(item)
            self.update_stack_display()
    def pop_item(self):
        try:
            message = self.stack.pop()
            messagebox.showinfo("Pop", message, parent=self)
            self.animate_pop(message)
            self.update_stack_display()
        except IndexError as e:
            messagebox.showerror("Error", str(e), parent=self)
    def peek_item(self):
        try:
            item = self.stack.peek()
            messagebox.showinfo("Peek", f"Top item: {item}", parent=self)
```

```
        except IndexError as e:
            messagebox.showerror("Error", str(e), parent=self)
    def check_is_empty(self):
        is_empty = self.stack.is_empty()
        messagebox.showinfo("Is Empty", f"Is the stack empty? {'Yes' if is_empty else 'No'}", parent=self)
    def check_size(self):
        size = self.stack.size()
        messagebox.showinfo("Size", f"Size of the stack: {size}", parent=self)
    def animate_push(self, item):
        original_color = self.stack_label.cget("fg")
        for _ in range(3):
            self.stack_label.config(fg="yellow")
            self.update()
            self.after(100)
            self.stack_label.config(fg=original_color)
            self.update()
            self.after(100)
    def animate_pop(self, item):
        original_color = self.stack_label.cget("fg")
        for _ in range(3):
            self.stack_label.config(fg="magenta")
            self.update()
            self.after(100)
            self.stack_label.config(fg=original_color)
```

```
        self.update()
        self.after(100)
if __name__ == "__main__":
    app = StackApp()
    app.mainloop()
```

Output: