Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

A Project Report on "Stack Visualization"

[Code No.: COMP 202]

(For mini-project)

Submitted by

Kiran Ranabhat (037996-24)

Submitted to

Sagar Acharya

Department of Computer Science and Engineering

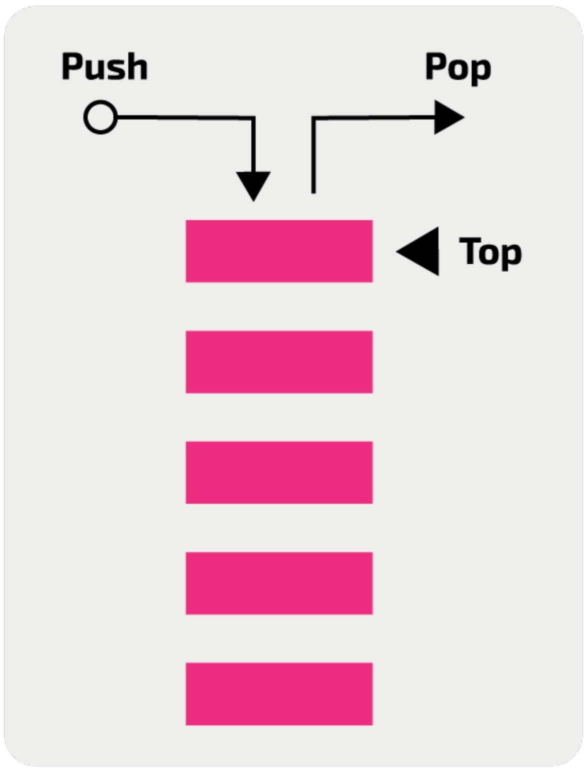
February 25, 2026

Acknowledgements

First of all, I am really thankful to my course instructor for his motivating and guiding me at in creating this mini project. This mini project gave me an idea how data structures particularly stack operations can be implemented in real-life through graphical visualization. My sincere thanks also go to the institution for providing the necessary facilities which helped me in giving my best in the project.

Abstract

It is a graphical representation of the Stack data structure originally coded in C language along with a graphics library named Raylib. A stack is one of the very basic linear data structures that strictly follow the Last In First Out (LIFO) principle. In other words, the element that was last put in the stack is the first one to be taken out. Just theoretically, stack operations could be very confusing tasks for beginners in Data Structures and Algorithms. This program is basically a very interactive and cool visualization that users can actually perform push and pop operations on a Stack in real time. And not only numbers as elements of the Stack, they are depicted visually as rectangles, so users can actually see the stack growing or shrinking.



The project with a simple graphical interface does not only make the conceptual understanding solid, but also the learning process becomes far more attractive than the traditional console-based programs. With this kind of visualization, the students are helped to understand the stack operations at the micro-level and thus, deepen their knowledge of basic data structures.

Acronyms / Abbreviations

The list of abbreviations used in this report are as follows:

- DSA — Data Structures and Algorithms
- UI — User Interface
- LIFO — Last In First Out
- GUI — Graphical User Interface
- API — Application Programming Interface

Table of Contents

Introduction

Background

Data Structures are indeed a very fundamental concept in computer science; basically, one cannot program efficiently if they do not have data structures. A stack is a very commonly used data structure and it is a linear data type.

The stack follows the Last In First Out (LIFO) principle, which means the last element to be pushed will be the first element to be popped out of the stack. Hence, stacks find their applications in arithmetic expression evaluation, undo features, recursive procedures, and computer memory management. Unbelievably, students sometimes get confused with the idea of a stack internally since, at first, stack operations are only explained theoretically or through text-based programs. At the end of the day, the acquisition of knowledge is best consolidated when it is visualized. Thus, picturing data structures is really a great method to grasp their functionalities and the internal working mechanisms.

This software is a graphical Stack visualization tool developed by using C language and Raylib graphics library. The system allows the user to perform push and pop operations through an intuitive interface. At every stage, through graphical blocks, one can see the screen and hence, the picture is very clear that the stack level is going up at one time while it is going down at another time.

The main objective of this tool is to provide a simple and fun educational visualizer, which somebody can use to get a first-hand experience of stack and its operations and thus, going beyond the textbook.

Objectives

The main objectives of this project are:

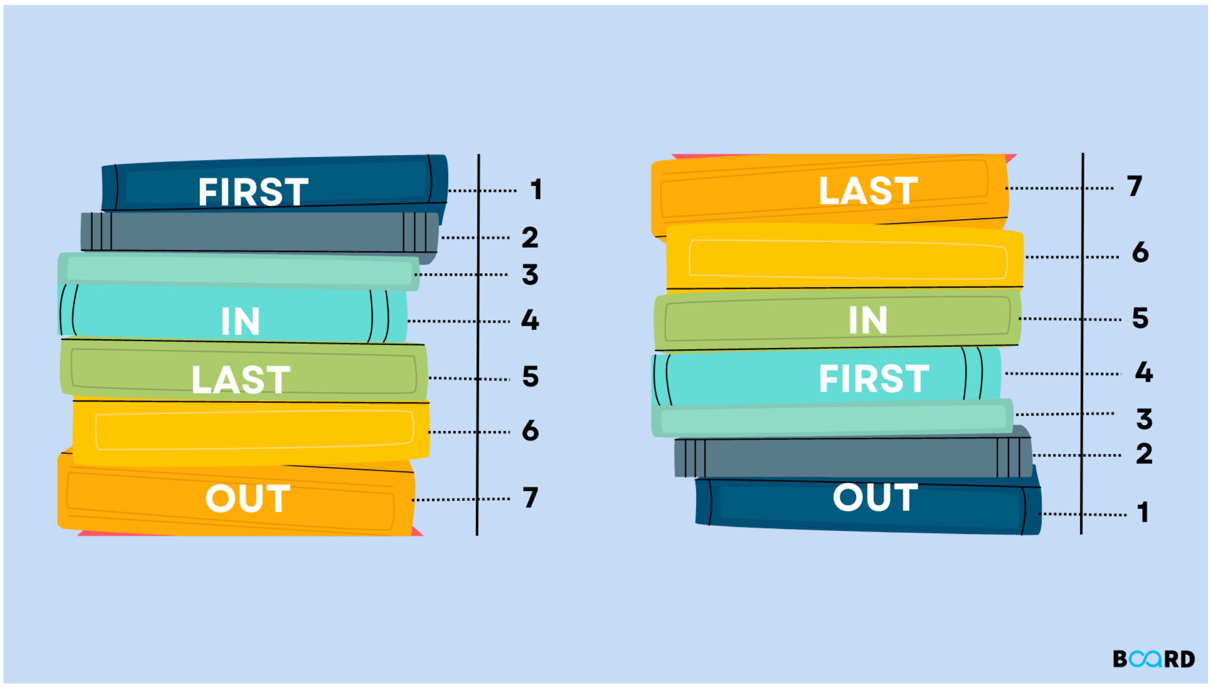
- To implement stack data structure using C programming
- To visualize push and pop operations graphically
- To display the top element pointer clearly
- To show stack size dynamically
- To demonstrate stack overflow and underflow conditions
- To create a simple and user-friendly graphical interface
- To enhance understanding of DSA concepts through visualization

Motivation and Significance

Students may find studying data structures from books alone very difficult and tedious. For instance, a lot of students fail to visualize how stack operations work inside the operation just by looking at the operations. So, a graphical stack visualizer that can provide real-time interaction and, hence, better conceptual clarity, was the idea. The most important thing about this project is that it is educational. It helps students easily understand the concept of LIFO, memory structure, stack operations, etc. Using a graphical representation makes learning more enjoyable and easier to understand as compared to the traditional methods. In addition, the project is a means of combining theoretical DSA with graphics programming which will not only be beneficial for academic learning but also for making presentations.

Literature Review

Computer science education has embraced visualization tools to explain data structures and algorithms which are considered complicated to the students. The conventional ways of teaching mostly rely on static diagrams and console outputs that don't always represent the dynamic behaviour of data structures such as stacks and queues. Graphically depicting data structures through various educational software and visualization tools has been one of the ways to help students understand. These tools reveal to students the internal workings of the data structures one step at a time. The authors of interactive visualization systems usually rely on graphical libraries like Raylib, SDL, and OpenGL for support because these are easy and efficient.



Raylib is a super lightweight, very easy-to-use graphics library that is implemented in C and therefore is perfect for an educational project or a small-scale application. It mainly serves basic necessities such as creating shapes, capturing inputs, and displaying text, which are all the essentials for making visualization tools. This project chooses simplicity and minimalism to demonstrate stack operations only, while sophisticated visualization frameworks come up with a host of features. More emphasis has been placed on making the presentation clear, the tool user-friendly, and the educational value high rather than on flashy animations or complicated interfaces.

Methodology

The Stack Visualization project is a modular and nicely structured application developed in C language and Raylib graphics library. To have neat and orderly code, the program is divided into different components.

First, the stack data structure is realized through a fixed-size array. In addition to the main operations pushing and popping, the stack module also prevents overflow and underflow through the conditions. Consequently, the stack's main operation is separated from the graphical user interface. Subsequently, Raylib is utilized for the development of the GUI.

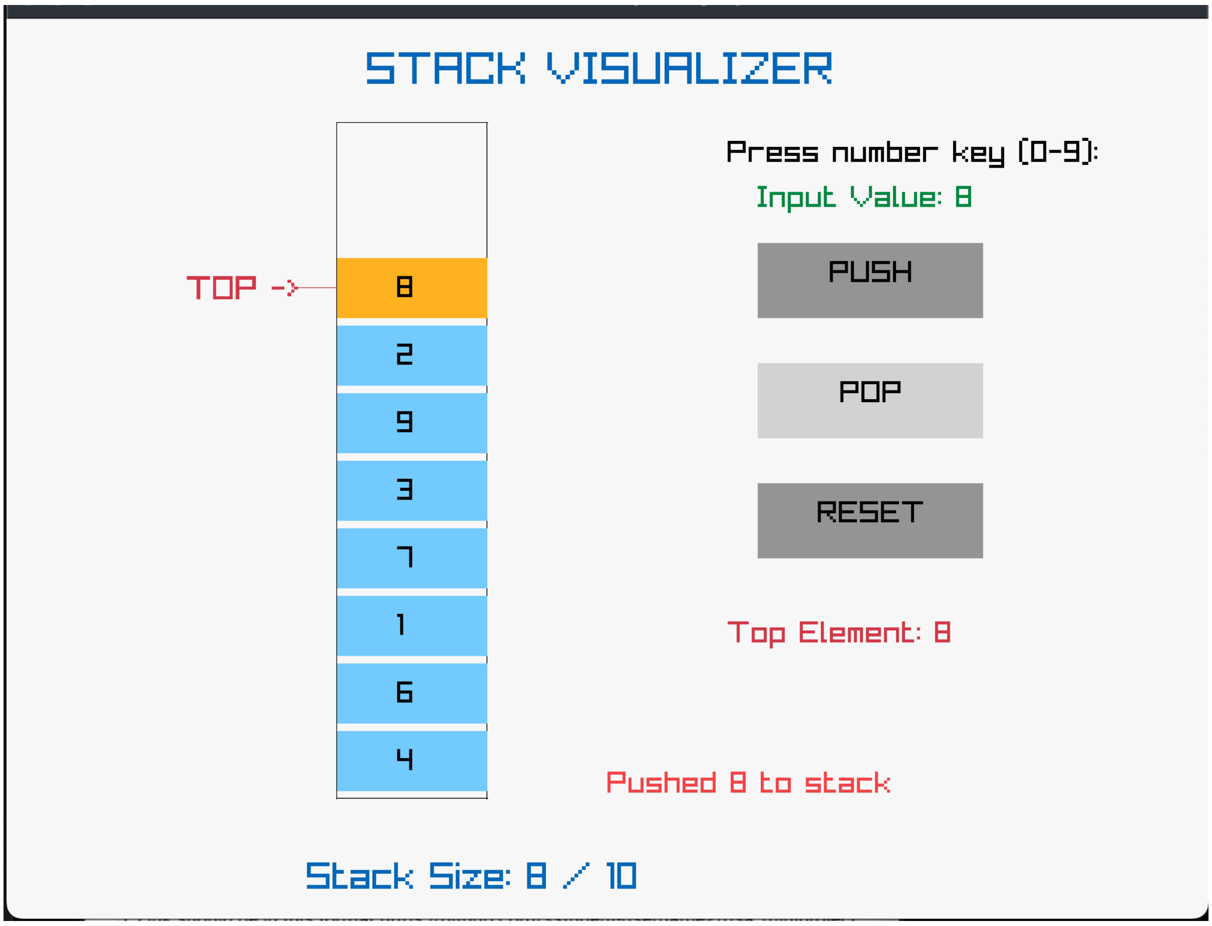
Once the window is ready, stack blocks, buttons, and text labels are shown as visual elements on the screen. A stack item is represented by a rectangular box which is stacked vertically so that the LIFO principle is clearer. Input is taken through the mouse device. Clicking the push button, for instance, results in adding the new element to the stack not only logically but also visually. Similarly, the pop button is pressed to get rid of the top element of the data stack as well as of the visualization.

Thus, the app remains simple, fun, and intuitive, and at the same time, the stack operation is shown in a very accurate manner.

Implementation

The whole project has been done in C and for visualization, the Raylib graphics library has been utilized. To make the code more understandable and easier to manage, the program has been split into different source files such as main. c, stack. c, stack. h, ui. c, and ui. h. For the stack, a pointer top is used to denote the position of the stack with the implementation being array-based. The push operation puts a new item at the incremented top pointer, but the pop operation removes the top item and decrements the pointer.

Out of the many functions of Raylib to make a GUI, the main functions used are InitWindow(), BeginDrawing(), DrawRectangle(), and DrawText(). As far as graphics are concerned, elements in the stack are visually represented as colored rectangular bricks, and operation commands are provided via buttons.



Press number key (0-9):

Input Value: 8

Top Element: 8

Pushed 8 to stack

Stack Size: 8 / 10

The entire program is inside a loop and the screen is refreshed according to user input at real time. Real-time rendering of this sort turns the visualization very smooth and interactive which leads to the overall user experience being enhanced.

Result and Discussion

The freshly introduced Stack Visualization application was tested on a macOS machine with the help of the Raylib library. Apart from the program correctly executing all the basic stack operations, such as push and pop, it also shows the changed stack visually with the help of a diagram on the display.

The GUI changes the majority of operations into something that is, visually, very easy to understand at the first glance; that is, it is a lot clearer how elements are added and removed from the stack if one follows the LIFO principle. Therefore, this visualization means a lot to getting a better and, maybe, even more fun learning experience as compared with a conventional console-based stack program. Also, the program is very much capable of dealing with situations of both stack overflow and underflow without experiencing any issues. Hence, the software's performance and trustworthiness are not impinged. Here interaction with such interface elements as buttons is so pleasant that visually the changes are seen immediately without any performance issues.

Overall, the results show that through the use of graphical visualization the concept of stack operations can be comprehended to a very great extent, which makes the learning of Data Structures not only a more enjoyable experience but also a more effective one for students.

Conclusion and Recommendation

Conclusion

This project displayed a picture of the stack data structure being implemented in the C programming language and the graphics library Raylib. Through the program screen, the users are allowed to perform stack operations visually via the interface provided by the program.

Showing the stack elements as blocks is a wonderful method for students to understand the LIFO (Last In First Out) concept more deeply. Moreover, learning through a GUI is more natural than just having a simple text-based one. Besides, from a logical point of view, the different components of the app are separated which makes the source code more straightforward to maintain and continue development. Essentially, the Stack Visualization project is a very good teaching aid that promotes the basic understanding of DSA concepts.

Adding more features such as animating effects, inputting any values, and illustrating other data structures like queues and linked lists would also be a brilliant idea to enhance the system.

Limitations

- Limited stack size (fixed array implementation)
- Only integer input supported
- No advanced animations
- Desktop-only application
- Basic UI design

Future Enhancement

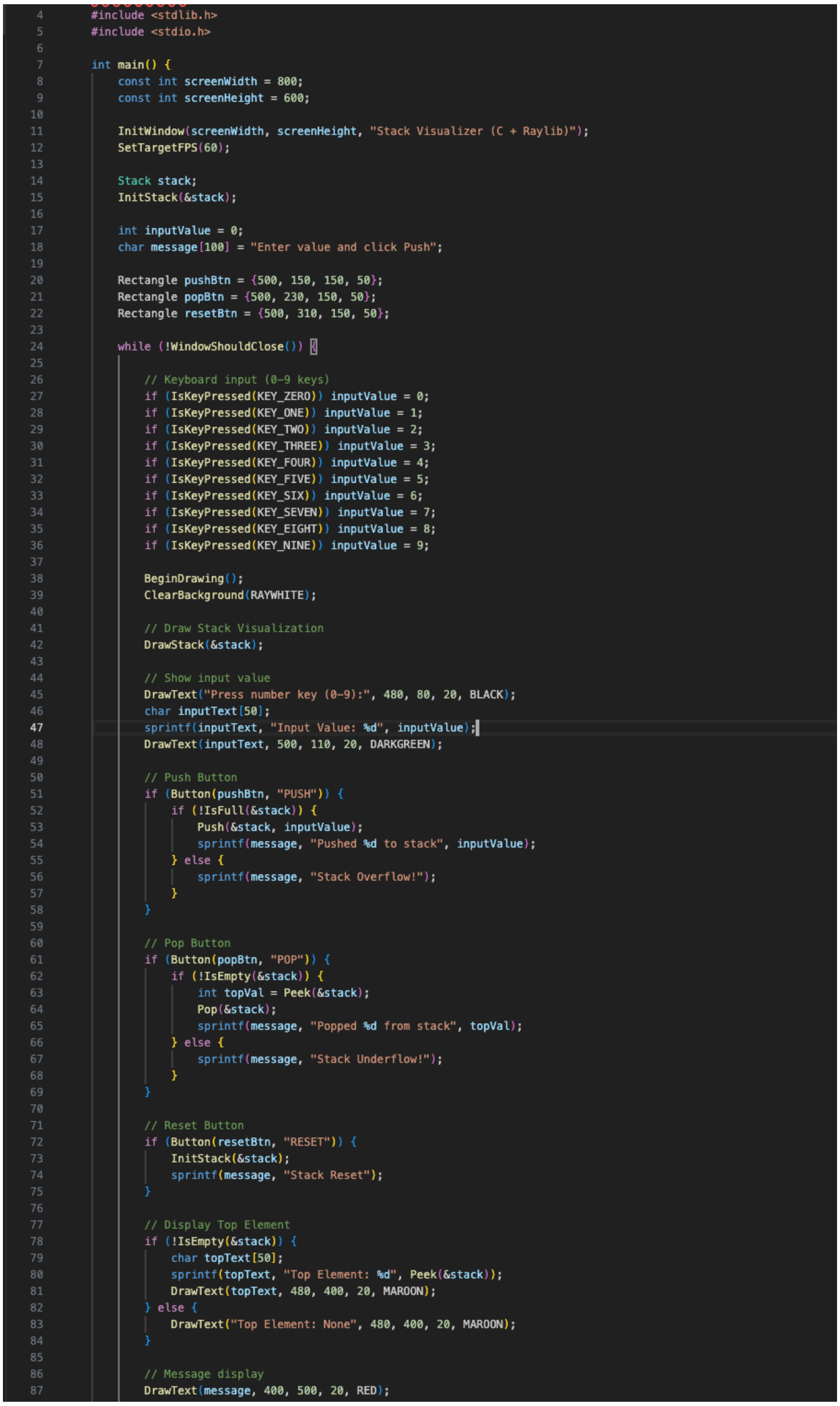
- Add smooth animation for push and pop
- Support dynamic stack size
- Add multiple data structure visualizations (Queue, Linked List)
- Improve UI design and themes
- Add sound effects and advanced interaction
- Convert into educational DSA learning tool

References

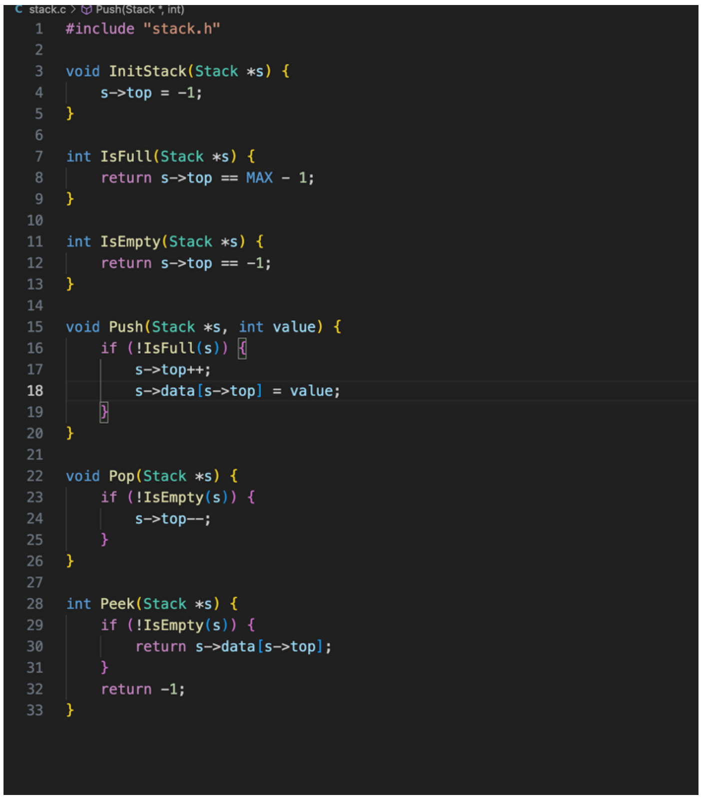
1. Introduction to Algorithms – Thomas H. Cormen
2. Raylib Official Documentation
3. Data Structures and Algorithms Course Notes

Appendix

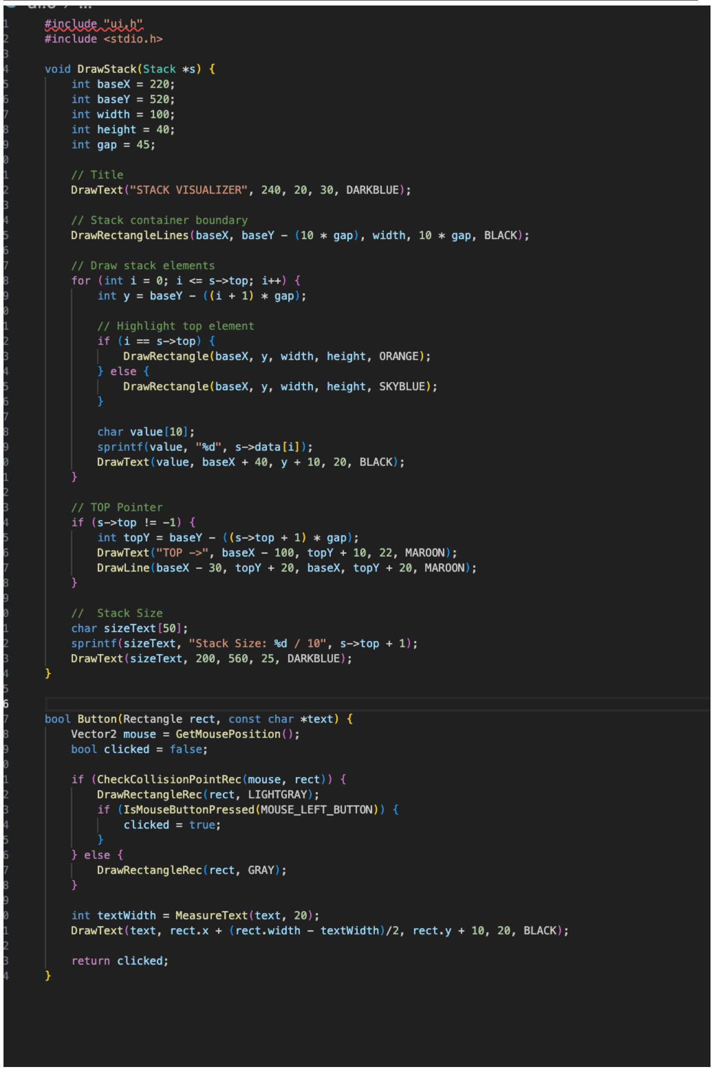
Appendix A



```c
#include <stdlib.h>
#include <stdio.h>

int main() {
    const int screenWidth = 800;
    const int screenHeight = 600;

    InitWindow(screenWidth, screenHeight, "Stack Visualizer (C + Raylib)");
    SetTargetFPS(60);

    Stack stack;
    InitStack(&stack);

    int inputValue = 0;
    char message[100] = "Enter value and click Push";

    Rectangle pushBtn = {500, 150, 150, 50};
    Rectangle popBtn = {500, 230, 150, 50};
    Rectangle resetBtn = {500, 310, 150, 50};

    while (!WindowShouldClose()) {

        // Keyboard input (0-9 keys)
        if (IsKeyPressed(KEY_ZERO)) inputValue = 0;
        if (IsKeyPressed(KEY_ONE)) inputValue = 1;
        if (IsKeyPressed(KEY_TWO)) inputValue = 2;
        if (IsKeyPressed(KEY_THREE)) inputValue = 3;
        if (IsKeyPressed(KEY_FOUR)) inputValue = 4;
        if (IsKeyPressed(KEY_FIVE)) inputValue = 5;
        if (IsKeyPressed(KEY_SIX)) inputValue = 6;
        if (IsKeyPressed(KEY_SEVEN)) inputValue = 7;
        if (IsKeyPressed(KEY_EIGHT)) inputValue = 8;
        if (IsKeyPressed(KEY_NINE)) inputValue = 9;

        BeginDrawing();
        ClearBackground(RAYWHITE);

        // Draw Stack Visualization
        DrawStack(&stack);

        // Show input value
        DrawText("Press number key (0-9):", 480, 80, 20, BLACK);
        char inputText[50];
        sprintf(inputText, "Input Value: %d", inputValue);
        DrawText(inputText, 500, 110, 20, DARKGREEN);

        // Push Button
        if (Button(pushBtn, "PUSH")) {
            if (!IsFull(&stack)) {
                Push(&stack, inputValue);
                sprintf(message, "Pushed %d to stack", inputValue);
            } else {
                sprintf(message, "Stack Overflow!");
            }
        }

        // Pop Button
        if (Button(popBtn, "POP")) {
            if (!IsEmpty(&stack)) {
                int topVal = Peek(&stack);
                Pop(&stack);
                sprintf(message, "Popped %d from stack", topVal);
            } else {
                sprintf(message, "Stack Underflow!");
            }
        }

        // Reset Button
        if (Button(resetBtn, "RESET")) {
            InitStack(&stack);
            sprintf(message, "Stack Reset");
        }

        // Display Top Element
        if (!IsEmpty(&stack)) {
            char topText[50];
            sprintf(topText, "Top Element: %d", Peek(&stack));
            DrawText(topText, 480, 400, 20, MAROON);
        } else {
            DrawText("Top Element: None", 480, 400, 20, MAROON);
        }

        // Message display
        DrawText(message, 400, 500, 20, RED);
```

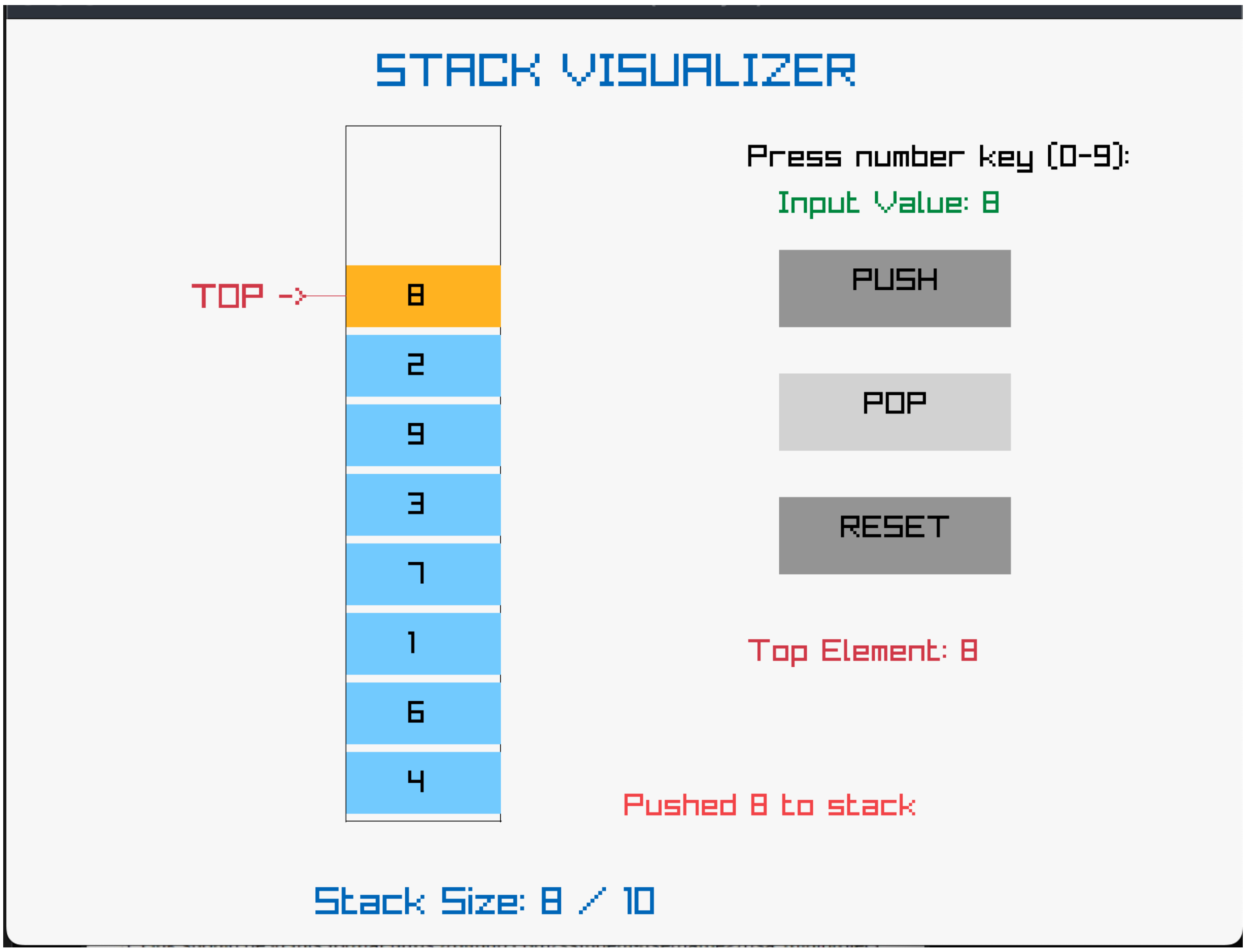

stack.c > Push(Stack *, int)

```c
#include "stack.h"

void InitStack(Stack *s) {
    s->top = -1;
}

int IsFull(Stack *s) {
    return s->top == MAX - 1;
}

int IsEmpty(Stack *s) {
    return s->top == -1;
}

void Push(Stack *s, int value) {
    if (!IsFull(s)) {
        s->top++;
        s->data[s->top] = value;
    }
}

void Pop(Stack *s) {
    if (!IsEmpty(s)) {
        s->top--;
    }
}

int Peek(Stack *s) {
    if (!IsEmpty(s)) {
        return s->data[s->top];
    }
    return -1;
}
```



```c
#include "ui.h"
#include <stdio.h>

void DrawStack(Stack *s) {
    int baseX = 220;
    int baseY = 520;
    int width = 100;
    int height = 40;
    int gap = 45;

    // Title
    DrawText("STACK VISUALIZER", 240, 20, 30, DARKBLUE);

    // Stack container boundary
    DrawRectangleLines(baseX, baseY - (10 * gap), width, 10 * gap, BLACK);

    // Draw stack elements
    for (int i = 0; i <= s->top; i++) {
        int y = baseY - ((i + 1) * gap);

        // Highlight top element
        if (i == s->top) {
            DrawRectangle(baseX, y, width, height, ORANGE);
        } else {
            DrawRectangle(baseX, y, width, height, SKYBLUE);
        }

        char value[10];
        sprintf(value, "%d", s->data[i]);
        DrawText(value, baseX + 40, y + 10, 20, BLACK);
    }

    // TOP Pointer
    if (s->top != -1) {
        int topY = baseY - ((s->top + 1) * gap);
        DrawText("TOP ->", baseX - 100, topY + 10, 22, MAROON);
        DrawLine(baseX - 30, topY + 20, baseX, topY + 20, MAROON);
    }

    //  Stack Size
    char sizeText[50];
    sprintf(sizeText, "Stack Size: %d / 10", s->top + 1);
    DrawText(sizeText, 200, 560, 25, DARKBLUE);
}


bool Button(Rectangle rect, const char *text) {
    Vector2 mouse = GetMousePosition();
    bool clicked = false;

    if (CheckCollisionPointRec(mouse, rect)) {
        DrawRectangleRec(rect, LIGHTGRAY);
        if (IsMouseButtonPressed(MOUSE_LEFT_BUTTON)) {
            clicked = true;
        }
    } else {
        DrawRectangleRec(rect, GRAY);
    }

    int textWidth = MeasureText(text, 20);
    DrawText(text, rect.x + (rect.width - textWidth)/2, rect.y + 10, 20, BLACK);

    return clicked;
}
```

Appendix B



Press number key (0-9):

Input Value: 8

Top Element: 8

Pushed 8 to stack

Stack Size: 8 / 10

Appendix C

https://github.com/KiranRanabhat/dsa_miniproject