计算机科学与技术系

《数据结构课程设计》评分表

学生/学号：　　田天成 2021011615　　专业/班级：　计算机 21-3

设计题目	12	成绩	
课程设计主要内容	一、内容 用一个一维数组作为双端共享栈，设计可视化算法实现进栈、出栈和判栈空和栈满操作。 二、任务和要求 对于双端共享栈可视化项目，主要任务包括实现双端共享栈的数据结构、开发一个直观的用户界面来展示栈操作，以及编写处理用户交互的逻辑。关键要求是确保准确性和用户友好性，保持程序的稳定性和健壮性，同时保持代码的可读性和可维护性。这个项目旨在结合数据结构理论与实际的软件开发实践。		

序号	评　价　项　目	评　分	
		满分	得分
1	实验报告内容充实，调理清晰，算法设计阐述的清楚。	20	
2	实验报告图表齐全、格式规范，质量高。	20	
3	算法设计与算法应用能力。	20	
4	程序测试用例设计全面，对算法的各种不同情况进行测试。	20	
5	程序代码质量高，代码可读性好。	20	
累计得分			

注：（1）成绩评定　采用百分制。

中国石油大学

计算机科学与技术系

《数据结构》课程设计报告

2023～2024 学年第二学期

课程设计题目	12
学生姓名	田天成
学号	2021011615
专业班级	计算机 21-3

2023 年 12 月

1.问题描述与分析

双端共享栈通常是指一个数组，其中两个栈共享同一个数组空间，但一个栈从数组的一端进出元素，另一个栈从数组的另一端进出元素。这种方式可以有效地利用数组空间。

基本操作

1. **进栈（Push）**：
   - 对于栈1，元素被添加到数组的开始处。
   - 对于栈2，元素被添加到数组的末尾。
2. **出栈（Pop）**：
   - 对于栈1，元素从数组的开始处被移除。
   - 对于栈2，元素从数组的末尾被移除。
3. **判栈空（IsEmpty）**：
   - 栈1空当它的头指针小于或等于数组的起始索引。
   - 栈2空当它的头指针大于或等于数组的最大索引。
4. **判栈满（IsFull）**：
   - 当两个栈的头指针相邻时，表示栈已满。

可视化设计

1. **数组表示**: 使用一维数组可视化，其中每个元素都是一个矩形块，可以标记为栈1或栈2的元素。
2. **指针显示**: 使用不同颜色，分别表示栈1和栈2的顶部。
3. **操作步骤**:
   - 进栈操作: 可视化元素在数组的相应端点添加。
   - 出栈操作: 可视化元素从数组的相应端点移除，并更新栈顶指针。
   - 判栈空: 显示一个消息，指出哪个栈是空的。
   - 判栈满: 当栈满时，显示一个消息。
4. **动态变化**: 当操作发生时，数组的状态实时更新，显示元素

2. 具体设计过程

2.1设计思路

1. **定义全局数据结构和变量**
   - 创建一个数组来表示栈。
   - 定义两个指针变量 `top1` 和 `top2` 来分别跟踪两个栈的顶部位置。
2. **实现栈的可视化**
   - 使用 HTML 创建一个显示栈的容器，以及用于用户交互的按钮和输入框。
   - 使用 CSS 设置栈元素的样式，包括栈顶指针的特殊样式。
3. **编写 JavaScript 逻辑**
   - 编写 `updateStackDisplay` 函数来动态显示栈的当前状态。
   - 实现进栈和出栈操作的函数（`pushStack1`，`pushStack2`，`popStack1`，`popStack2`）。
   - 为进栈和出栈按钮添加事件监听器，以便在用户点击时执行相应的操作。
   - 实现栈的状态更新和错误处理逻辑。

4. **用户交互**

   - 用户通过输入框输入数据，并通过按钮进行进栈或出栈操作。
   - 每次操作后，栈的视觉表示会根据最新的数据动态更新。

2.2 流程图

以下是对该程序的一个简化流程图：

```
+----------------------+
| 初始化全局变量和数组 |
+----------------------+
          |
          v
+----------------------+
| 创建HTML界面元素     |
+----------------------+
          |
          v
+----------------------+
| 应用CSS样式          |
+----------------------+
          |
          v
+----------------------+
| 添加JS事件监听器     |
+----------------------+
          |
          v
+----------------------+
| 用户输入和操作       |
+----------------------+
          |
          v
+----------------------+
| 执行进栈/出栈操作    |
+----------------------+
          |
          v
+----------------------+
| 更新栈的可视化表示   |
+----------------------+
          |
          v
+----------------------+
| 检查并处理错误       |
+----------------------+
          |
          v
+----------------------+
| 循环到新的用户操作   |
+----------------------+
```

这个流程图表示了从程序初始化到用户交互，以及栈状态更新的整个过程。每一步骤都是互相连接的，确保了程序可以根据用户的操作动态响应和更新状态。

2.3 函数实现说明

1. `updateStackDisplay`

功能

- `updateStackDisplay` 函数负责更新网页上的栈可视化显示。它遍历整个栈数组，为每个元素创建一个 HTML `div` 元素，并根据栈顶的位置调整这些 `div` 的样式。

参数

- 无参数。

输入/输出

- **输入**：无直接输入。该函数依赖于全局变量 `stack`、`top1` 和 `top2` 的当前状态。
- **输出**：无返回值。此函数直接修改 DOM，更新栈的视觉表示。

2. `pushStack1` **和** `pushStack2`

这两个函数功能相似，分别用于处理栈1和栈2的进栈操作，因此我会合并它们的说明。

功能

- 这些函数用于将用户输入的值添加到栈中。它们首先检查输入值是否为空，然后检查是否有足够的空间在栈中进行进栈操作。如果条件满足，函数将用户输入的值添加到栈的相应端。

参数

- 无参数。

输入/输出

- **输入**：
  - 对于 `pushStack1`：来自 `id="inputStack1"` 的 HTML 输入元素的值。
  - 对于 `pushStack2`：来自 `id="inputStack2"` 的 HTML 输入元素的值。
- **输出**：无返回值。此函数会更改全局 `stack` 数组，并调用 `updateStackDisplay` 来更新 UI。

3. `popStack1` **和** `popStack2`

与 `pushStack1` 和 `pushStack2` 类似，`popStack1` 和 `popStack2` 也有相似的功能，但用于出栈操作。

功能

- 这些函数用于从栈中移除顶部元素。它们首先检查栈是否为空，如果不为空，则从栈中移除顶部元素。

参数

- 无参数。

输入/输出

- **输入**：无直接输入。这些函数依赖于全局变量 `stack`、`top1` 和 `top2` 的当前状态。
- **输出**：无返回值。此函数会更改全局 `stack` 数组，并调用 `updateStackDisplay` 来更新 UI。

注意事项

- 这些函数都是为特定应用场景编写的，它们依赖于特定的全局变量和 DOM 元素。在不同的上下文中使用时可能需要适当的修改。

- 函数 `pushStack1` 和 `pushStack2` 在处理输入时没有进行类型检查或转换。在更复杂的实现中，可能需要考虑输入验证和错误处理。

3. 程序运行说明

输入数据格式与内容

1. **栈元素输入**
   - **类型**：文本
   - **格式**：用户通过文本框输入栈的元素。输入通常是字符串或数字。
   - **范围**：通常是任何字符串或数字，但取决于实际应用的需要，可能会有限制。
   - **示例**：用户在栈1的输入框中输入 "A"，在栈2的输入框中输入 "5"。
2. **操作选择**
   - **类型**：用户操作
   - **格式**：用户通过点击按钮选择操作，如进栈（push）或出栈（pop）。
   - **范围**：四种操作 - 栈1进栈、栈1出栈、栈2进栈、栈2出栈。
   - **示例**：用户点击“栈1进栈”按钮来将输入的元素添加到栈1的顶部。

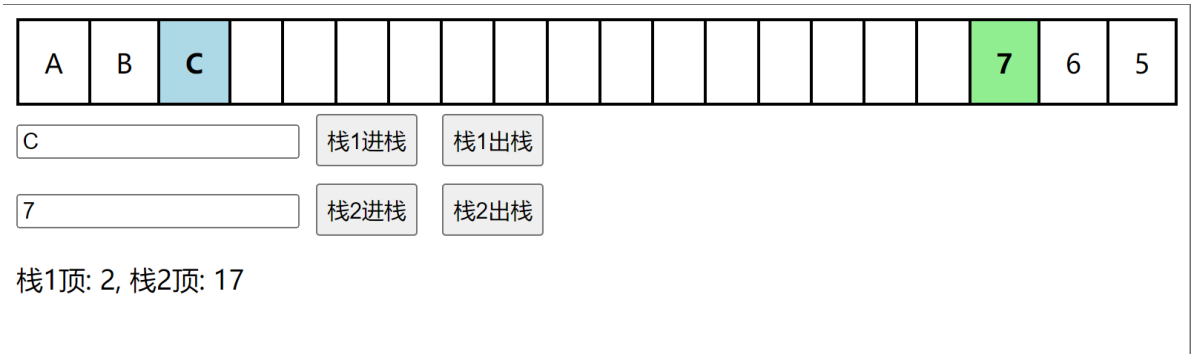
输出格式

1. **栈的可视化表示**
   - **类型**：图形界面
   - **格式**：栈的每个元素都在一个水平排列的格子中显示。栈顶指针用特殊颜色或标记显示。
   - **更新**：每次操作后，栈的可视化表示会根据栈的最新状态更新。
   - **示例**：如果栈1的顶部有元素 "A"，那么对应的格子会显示 "A"，并且该格子会有特殊的样式表示它是栈顶。
2. **状态信息**
   - **类型**：文本
   - **格式**：显示栈的状态信息，如栈顶位置、是否栈满或栈空。
   - **更新**：每次操作后，状态信息会更新以反映栈的最新状态。
   - **示例**：如果栈1刚刚执行了进栈操作，状态信息可能会显示“栈1顶: 2”。
3. **错误/警告消息**
   - **类型**：弹窗或文本
   - **格式**：当用户尝试执行非法操作时（如在空栈上执行出栈操作），显示错误或警告消息。
   - **更新**：在出现错误操作时立即显示。
   - **示例**：如果用户尝试从空的栈1中出栈，可能会出现一个弹窗消息“栈1为空！”

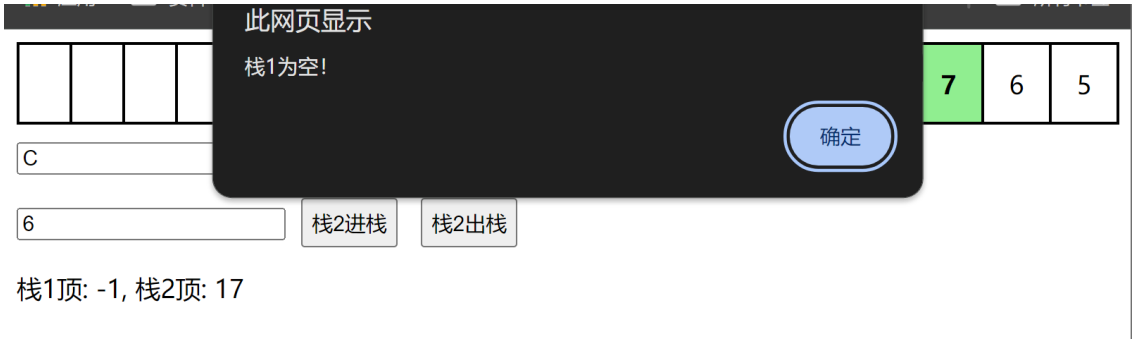
   这些输入和输出格式保证了用户与程序的交互是直观和用户友好的，同时确保了程序可以准确地显示栈的当前状态和操作结果。

4. 程序运行结果

- **同时进行栈1和栈2的操作，确保它们不会相互干扰。**



- **在栈为空时尝试出栈操作，应收到栈空的提示。**

1. 

- **在栈已满时尝试进栈操作，应收到栈满的提示。**

5. 心得与结论

完成双端共享栈可视化程序的设计和开发过程后，有几个关键的心得和结论可以总结：

1. **明确需求和设计目标**：在开始编码之前，清楚地理解需求和设计目标是非常重要的。这包括了解用户将如何与程序交互以及程序需要达到的功能。
2. **模块化和结构化编程**：将程序划分为独立的模块（如HTML结构、CSS样式和JavaScript逻辑），有助于提高代码的可读性和可维护性。同时，每个函数应该有一个明确的目的，这有助于调试和未来的功能扩展。
3. **用户界面和体验**：用户界面的设计对于确保用户能够直观和有效地与程序交互至关重要。良好的用户体验包括直观的操作、清晰的可视化以及及时的反馈（如错误提示）。
4. **测试的重要性**：彻底的测试是确保软件质量的关键。这不仅包括功能测试，还包括界面测试、边缘情况测试和性能测试。测试有助于发现并修复可能在开发过程中遗漏的问题。
5. **响应性和兼容性**：在不同设备和浏览器上测试应用程序的重要性。一个良好的程序应当能够在多种环境下稳定运行。
6. **代码优化和维护**：随着时间的推移和需求的变化，程序可能需要优化和维护。保持代码的简洁和高效，以及文档的更新，是持续维护的关键。
7. **学习和成长**：每个项目都是学习和成长的机会。无论项目的大小，都有机会学习新技术、解决新问题并提高作为开发者的技能。

总之，这个项目不仅是关于编码技能的应用，还涉及到软件开发的各个方面，包括需求分析、设计、测试和用户体验。每个阶段都有其独特的挑战和教训，这些经验对于任何软件开发者都是宝贵的。

```html
<!DOCTYPE html>
<html lang="zh-CN">

<head>
  <meta charset="UTF-8">
  <title>双端共享栈可视化</title>
  <link rel="stylesheet" href="style.css">
```

```
</head>

<body>
  <div id="stackContainer"></div>
  <div>
    <input type="text" id="inputStack1" placeholder="输入栈1值">
    <button id="pushStack1">栈1进栈</button>
    <button id="popStack1">栈1出栈</button>
  </div>
  <div>
    <input type="text" id="inputStack2" placeholder="输入栈2值">
    <button id="pushStack2">栈2进栈</button>
    <button id="popStack2">栈2出栈</button>
  </div>
  <div id="stackStatus"></div>
  <script src="script.js"></script>
</body>

</html>
```

```
// script.js

// 创建一个包含20个元素并初始化为null的数组，模拟两个栈
let stack = new Array(20).fill(null);
let top1 = -1; // 栈1的顶部指针
let top2 = stack.length; // 栈2的顶部指针

// 更新栈的显示
function updateStackDisplay() {
  const container = document.getElementById('stackContainer');
  container.innerHTML = '';
  for (let i = 0; i < stack.length; i++) {
    const element = document.createElement('div');
    element.classList.add('stackElement');
    element.textContent = stack[i] !== null ? stack[i] : '';
    if (i === top1) {
      element.classList.add('stackPointer1');
    }
    if (i === top2) {
      element.classList.add('stackPointer2');
    }
    container.appendChild(element);
  }
  document.getElementById('stackStatus').textContent =
    `栈1项: ${top1}, 栈2项: ${top2}`;
}

// 出栈操作，栈1
function popStack1() {
  if (top1 >= 0) {
    stack[top1--] = null;
    updateStackDisplay();
  } else {
    alert('栈1为空！');
  }
```

```
}

// 入栈操作，栈1
function pushStack1() {
  const value = document.getElementById('inputStack1').value;
  if (value === '') {
    alert('请输入栈1的值！');
    return;
  }
  if (top1 < top2 - 1) {
    stack[++top1] = value;
    updateStackDisplay();
  } else {
    alert('栈已满！');
  }
}

// 入栈操作，栈2
function pushStack2() {
  const value = document.getElementById('inputStack2').value;
  if (value === '') {
    alert('请输入栈2的值！');
    return;
  }
  if (top2 - 1 > top1) {
    stack[--top2] = value;
    updateStackDisplay();
  } else {
    alert('栈已满！');
  }
}

// 出栈操作，栈2
function popStack2() {
  if (top2 < stack.length) {
    stack[top2++] = null;
    updateStackDisplay();
  } else {
    alert('栈2为空！');
  }
}

// 为按钮添加事件监听器
document.getElementById('pushStack1').addEventListener('click', pushStack1);
document.getElementById('popStack1').addEventListener('click', popStack1);
document.getElementById('pushStack2').addEventListener('click', pushStack2);
document.getElementById('popStack2').addEventListener('click', popStack2);

// 初始化时更新栈的显示
updateStackDisplay();

```

```
/* style.css */
#stackContainer {
  display: flex;
```

```
  border: 1px solid black;
  height: 50px;
  overflow: hidden;
}

.stackElement {
  border: 1px solid black;
  flex-grow: 1;
  text-align: center;
  line-height: 50px;
}

.stackPointer1,
.stackPointer2 {
  font-weight: bold;
}

.stackPointer1 {
  background-color: lightblue;
}

.stackPointer2 {
  background-color: lightgreen;
}

button {
  margin: 5px;
  padding: 5px;
}

#stackStatus {
  margin-top: 10px;
}
```