# Assignment: Implementing `strstr` with String Search Algorithms and Regular Expressions

## Objective

- **Implement** a custom `strstr` function using one of the classic string search algorithms: **Knuth-Morris-Pratt (KMP)**, **Rabin-Karp**, or **Boyer-Moore**.
- **Write** regular expressions to find specific patterns within provided text samples.

## Instructions

1. **Algorithm Implementation**

    - **Choose** one of the following algorithms for your custom `strstr` function:
        - **Knuth-Morris-Pratt (KMP) Algorithm**
        - **Rabin-Karp Algorithm**
        - **Boyer-Moore Algorithm**
    - **Implement** the chosen algorithm in a function that accepts two strings:
        - **Text**: The string in which to search.
        - **Pattern**: The substring pattern to find.
    - **Return** the index of the first occurrence of the pattern in the text, or `-1` if not found.
    - **Test** your function with various inputs, including:
        - Patterns that occur multiple times.
        - Patterns that overlap.
        - Patterns that do not exist in the text.

2. **Regular Expressions Practice**

    - **Use** the provided text samples below.
    - **Write** a regular expression for each specified pattern.
    - **Test** your regex to ensure it matches all correct instances and excludes incorrect ones.
    - **Explain** each regex pattern with comments or a brief description.

3. **Reflection**

    - **Compare** the process of implementing the algorithm-based `strstr` function with crafting regex patterns.
    - **Discuss** the following:
        - The challenges faced in each part.
        - Scenarios where one method is more suitable than the other.
        - The importance of understanding both approaches in computer science.

**REGEX:**

---

**Sample Text 1:**

`(See email.txt for text to search).`

- **Pattern to Match**: **Email addresses**
    - **Task**: Write a regex to find all email addresses in the text.

**Sample Text 2:**

(See dates.txt for text to search).

- **Pattern to Match**: **Dates**
- **Task**: Write a regex to find all dates in the formats:
    - `MM/DD/YYYY`
    - `YYYY-MM-DD`
    - `Month DD, YYYY` (e.g., March 30, 2024)

**Sample Text 3:**

`(See phones.txt for text to search).`

- **Pattern to Match**: **Phone numbers**
    - **Task**: Write a regex to find all phone numbers in various common formats.

**Sample Text 4:**

(See urls.txt for text to search).

- **Pattern to Match**: **URLS**
    - **Task**: Write a regex to find all URLs starting with `http`, `https`, or `ftp`.

**Sample Text 5:**
(See color_codes.txt for text to search).

- **Pattern to Match**: **Hexadecimal color codes**
    - **Task**: Write a regex to find all hex color codes in the formats `#RRGGBB` and `#RGB`.

---

**Instructions for Each Pattern:**

- **Write** the regex pattern in the syntax of the programming language you're using (e.g., Python's `re` module).

  - **Test** the regex against the sample text to ensure accuracy.
  - **Include** a brief explanation of your regex pattern:
    - Describe the components (e.g., character classes, quantifiers).
    - Explain how the pattern matches the required text.

**Extra Credit**

- **Second Algorithm Implementation**
  - **Implement** a second `strstr` function using a different algorithm from the one you initially chose.
  - **Perform** a performance comparison between the two implementations:
    - Time complexity analysis.
    - Practical runtime measurements with large texts.
  - **Discuss** the advantages and disadvantages of each algorithm.

---

**Submission Guidelines**

- **Code Files**: Submit your source code files for the `strstr` function(s) and regex tests.
- **Explanation Document**: Include explanations and reflections in a separate document or as comments in your code.
- **Testing Evidence**: Provide sample inputs and outputs that demonstrate your code and regex patterns work correctly.

**Tips**

- **Testing**: Use diverse and edge-case inputs to thoroughly test your implementations.
- **Resources**: Refer to course materials and reputable online resources to understand the algorithms and regex syntax.
- **Documentation**: Write clean, well-commented code to make your logic clear.