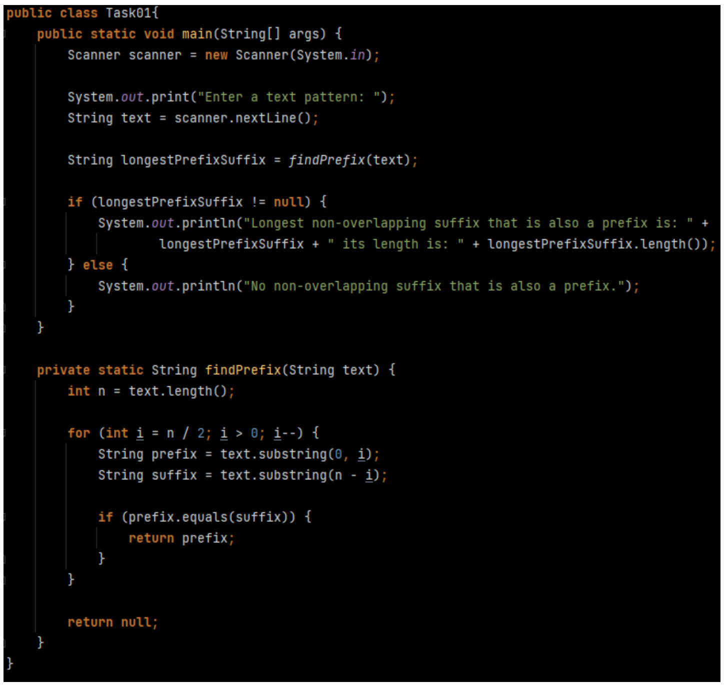
ICS-202 Lab-11 Report

Mohammed Busaleh - 202158210

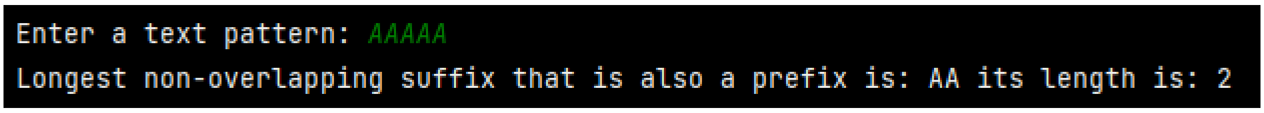
Task 1:

Code:

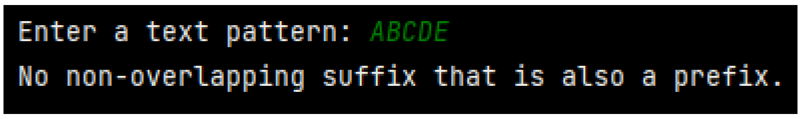


```java
public class Task01{
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a text pattern: ");
        String text = scanner.nextLine();

        String longestPrefixSuffix = findPrefix(text);

        if (longestPrefixSuffix != null) {
            System.out.println("Longest non-overlapping suffix that is also a prefix is: " +
                    longestPrefixSuffix + " its length is: " + longestPrefixSuffix.length());
        } else {
            System.out.println("No non-overlapping suffix that is also a prefix.");
        }
    }

    private static String findPrefix(String text) {
        int n = text.length();

        for (int i = n / 2; i > 0; i--) {
            String prefix = text.substring(0, i);
            String suffix = text.substring(n - i);

            if (prefix.equals(suffix)) {
                return prefix;
            }
        }

        return null;
    }
}
```

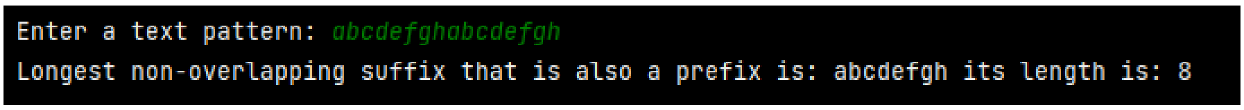
Output:



```
Enter a text pattern: AAAAA
Longest non-overlapping suffix that is also a prefix is: AA its length is: 2
```



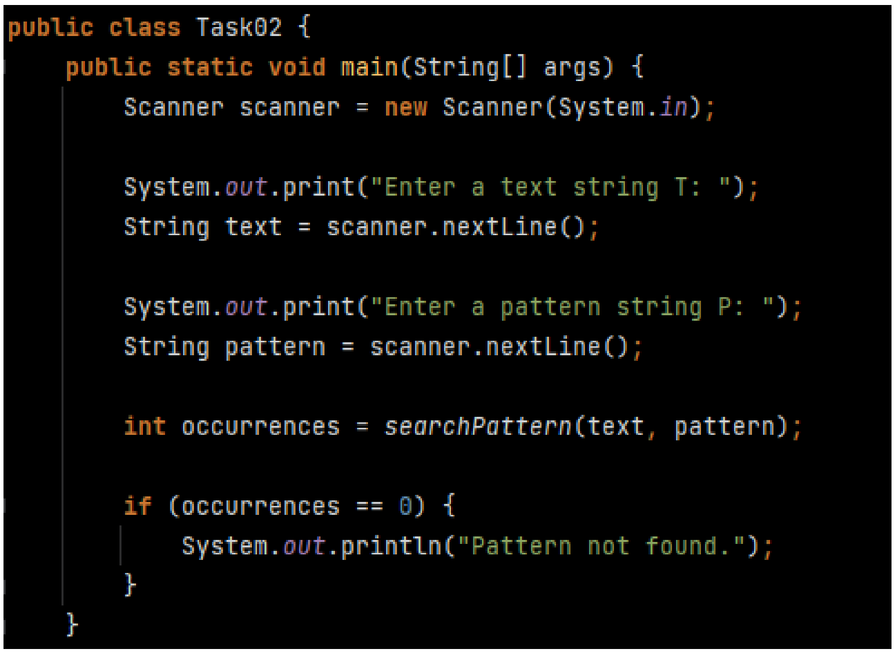
```
Enter a text pattern: ABCDE
No non-overlapping suffix that is also a prefix.
```



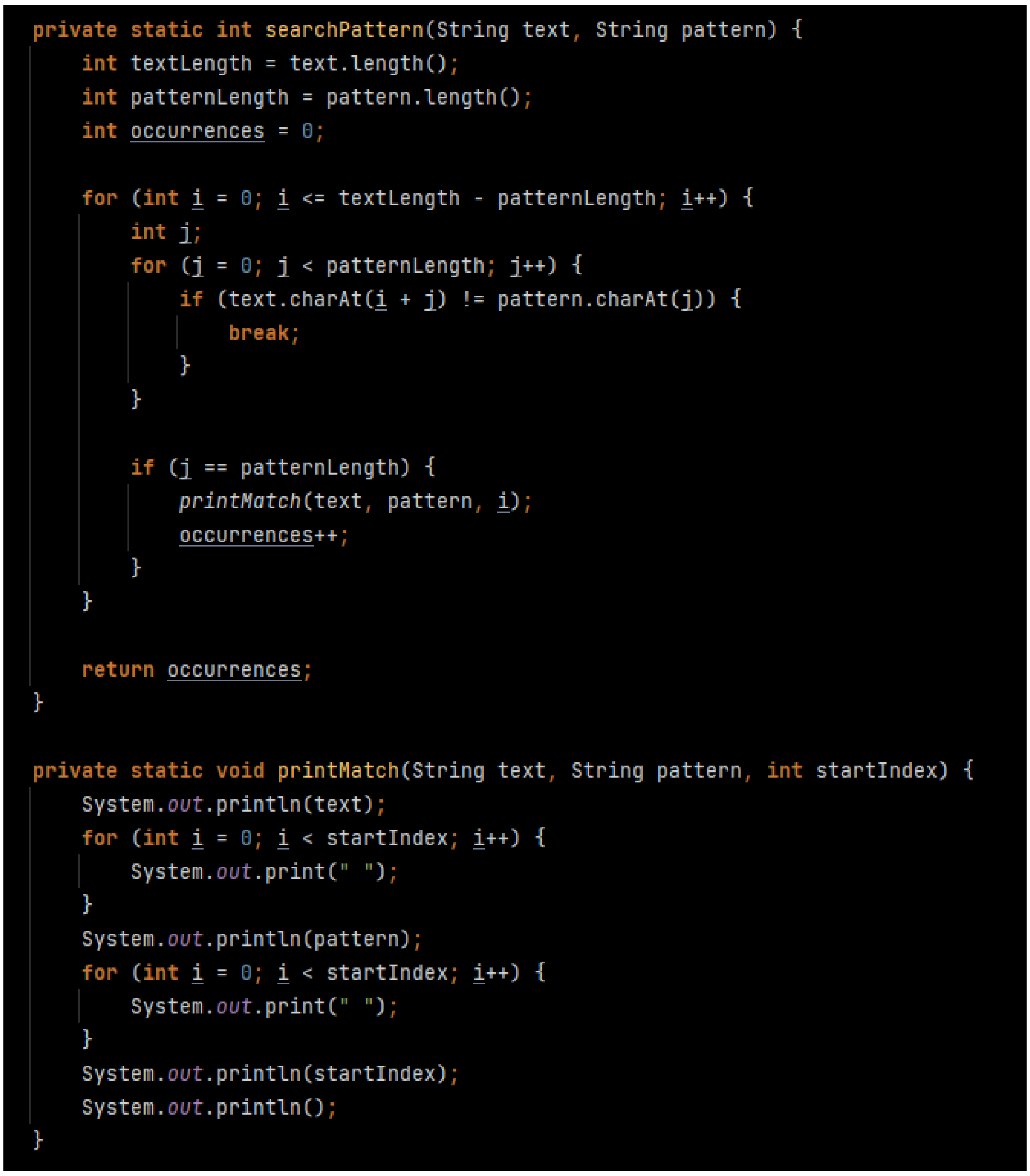
```
Enter a text pattern: abcdefghabcdefgh
Longest non-overlapping suffix that is also a prefix is: abcdefgh its length is: 8
```

Task 2:

Code:

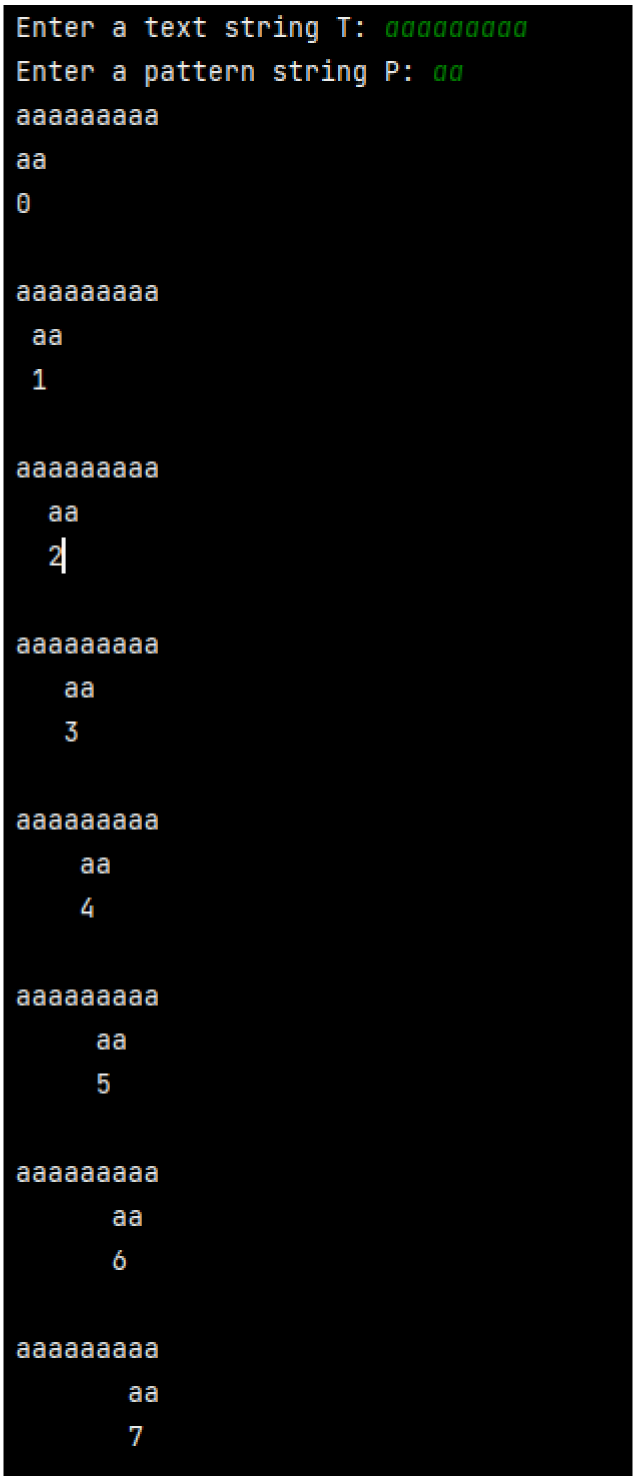


```java
public class Task02 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a text string T: ");
        String text = scanner.nextLine();

        System.out.print("Enter a pattern string P: ");
        String pattern = scanner.nextLine();

        int occurrences = searchPattern(text, pattern);

        if (occurrences == 0) {
            System.out.println("Pattern not found.");
        }
    }
```



```java
private static int searchPattern(String text, String pattern) {
    int textLength = text.length();
    int patternLength = pattern.length();
    int occurrences = 0;

    for (int i = 0; i <= textLength - patternLength; i++) {
        int j;
        for (j = 0; j < patternLength; j++) {
            if (text.charAt(i + j) != pattern.charAt(j)) {
                break;
            }
        }

        if (j == patternLength) {
            printMatch(text, pattern, i);
            occurrences++;
        }
    }

    return occurrences;
}

private static void printMatch(String text, String pattern, int startIndex) {
    System.out.println(text);
    for (int i = 0; i < startIndex; i++) {
        System.out.print(" ");
    }
    System.out.println(pattern);
    for (int i = 0; i < startIndex; i++) {
        System.out.print(" ");
    }
    System.out.println(startIndex);
    System.out.println();
}
```

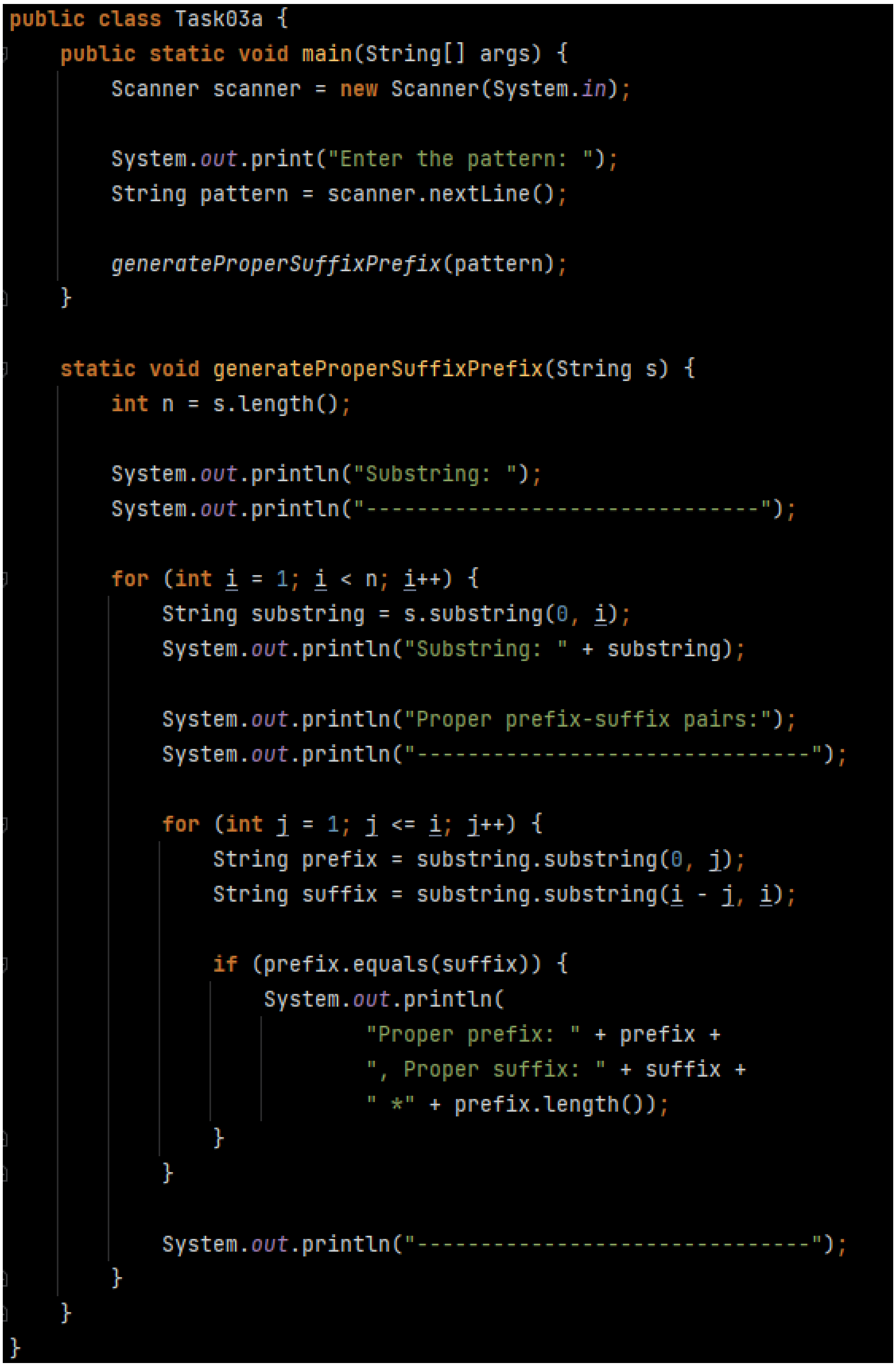
Output:



```
Enter a text string T: aaaaaaaaa
Enter a pattern string P: aa
aaaaaaaaa
aa
0

aaaaaaaaa
 aa
 1

aaaaaaaaa
  aa
  2

aaaaaaaaa
   aa
   3

aaaaaaaaa
    aa
    4

aaaaaaaaa
     aa
     5

aaaaaaaaa
      aa
      6

aaaaaaaaa
       aa
       7
```

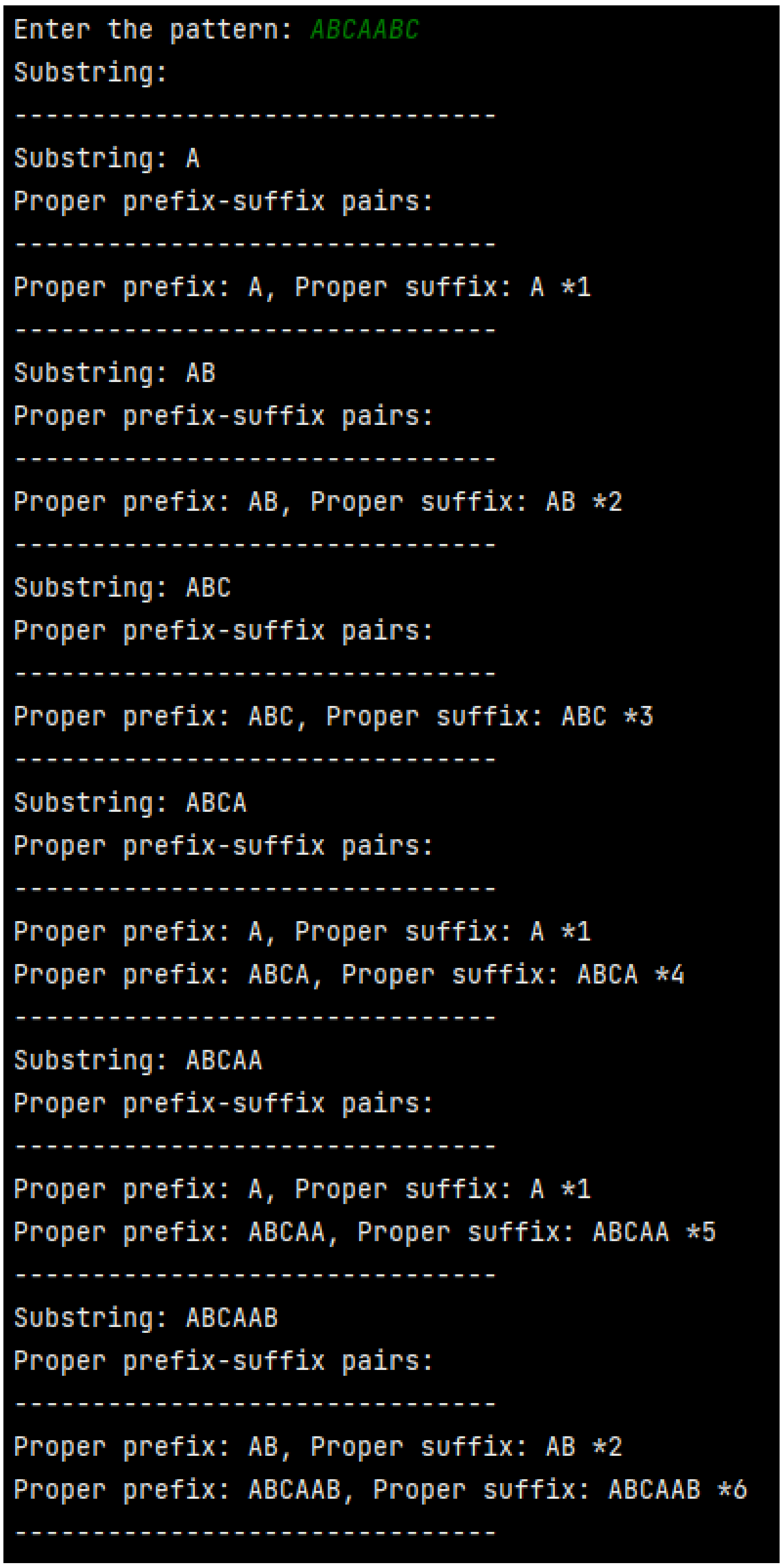
Task 3A:

Code:



```java
public class Task03a {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the pattern: ");
        String pattern = scanner.nextLine();

        generateProperSuffixPrefix(pattern);
    }

    static void generateProperSuffixPrefix(String s) {
        int n = s.length();

        System.out.println("Substring: ");
        System.out.println("-------------------------------");

        for (int i = 1; i < n; i++) {
            String substring = s.substring(0, i);
            System.out.println("Substring: " + substring);

            System.out.println("Proper prefix-suffix pairs:");
            System.out.println("-------------------------------");

            for (int j = 1; j <= i; j++) {
                String prefix = substring.substring(0, j);
                String suffix = substring.substring(i - j, i);

                if (prefix.equals(suffix)) {
                    System.out.println(
                            "Proper prefix: " + prefix +
                            ", Proper suffix: " + suffix +
                            " *" + prefix.length());
                }
            }

            System.out.println("-------------------------------");
        }
    }
}
```

Output:



```
Enter the pattern: ABCAABC
Substring:
-------------------------------
Substring: A
Proper prefix-suffix pairs:
-------------------------------
Proper prefix: A, Proper suffix: A *1
-------------------------------
Substring: AB
Proper prefix-suffix pairs:
-------------------------------
Proper prefix: AB, Proper suffix: AB *2
-------------------------------
Substring: ABC
Proper prefix-suffix pairs:
-------------------------------
Proper prefix: ABC, Proper suffix: ABC *3
-------------------------------
Substring: ABCA
Proper prefix-suffix pairs:
-------------------------------
Proper prefix: A, Proper suffix: A *1
Proper prefix: ABCA, Proper suffix: ABCA *4
-------------------------------
Substring: ABCAA
Proper prefix-suffix pairs:
-------------------------------
Proper prefix: A, Proper suffix: A *1
Proper prefix: ABCAA, Proper suffix: ABCAA *5
-------------------------------
Substring: ABCAAB
Proper prefix-suffix pairs:
-------------------------------
Proper prefix: AB, Proper suffix: AB *2
Proper prefix: ABCAAB, Proper suffix: ABCAAB *6
-------------------------------
```

Task 3B:

Solution:

ABCDE

J	Pattern [0...j-1]	Prop. pre	Prop. suf	next [j]
0	-	λ	λ	-1
1	A	λ	λ	0
2	AB	A	A	0
3	ABC	AB	AB	0
4	ABCD	ABC	ABC	0
5	ABCDE	ABCD	ABCD	0

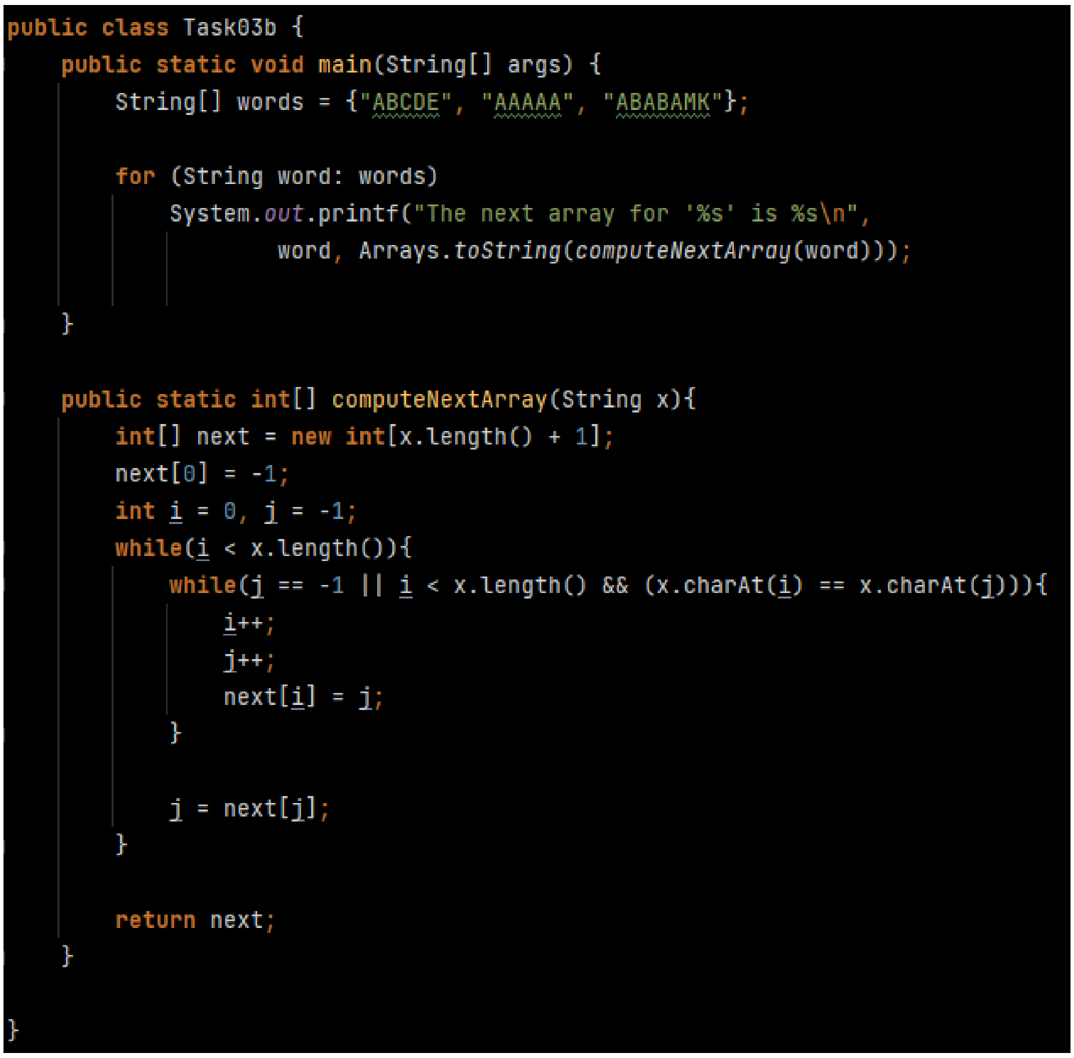
AAAAA

J	Pattern [0...j-1]	Prop. pre	Prop. suf	next [j]
0	-	λ	λ	-1
1	A	λ	λ	0
2	AA	A	A	1
3	AAA	AA	AA	2
4	AAAA	AAA	AAA	3
5	AAAAA	AAAA	AAAA	4

ABABAMK

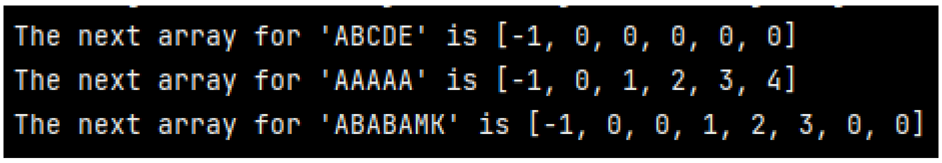
J	Pattern [0...j-1]	Prop. pre	Prop. suf	next [j]
0	-	λ	λ	-1
1	A	λ	λ	0
2	AB	λ	λ	0
3	ABA	A	A	1
4	ABAB	AB	AB	2
5	ABABA	ABA	ABA	3
6	ABABAM	λ	λ	0
7	ABABAMK	λ	λ	0

Code:



```
public class Task03b {
    public static void main(String[] args) {
        String[] words = {"ABCDE", "AAAAA", "ABABAMK"};

        for (String word: words)
            System.out.printf("The next array for '%s' is %s\n",
                    word, Arrays.toString(computeNextArray(word)));

    }

    public static int[] computeNextArray(String x){
        int[] next = new int[x.length() + 1];
        next[0] = -1;
        int i = 0, j = -1;
        while(i < x.length()){
            while(j == -1 || i < x.length() && (x.charAt(i) == x.charAt(j))){
                i++;
                j++;
                next[i] = j;
            }

            j = next[j];
        }

        return next;
    }

}
```

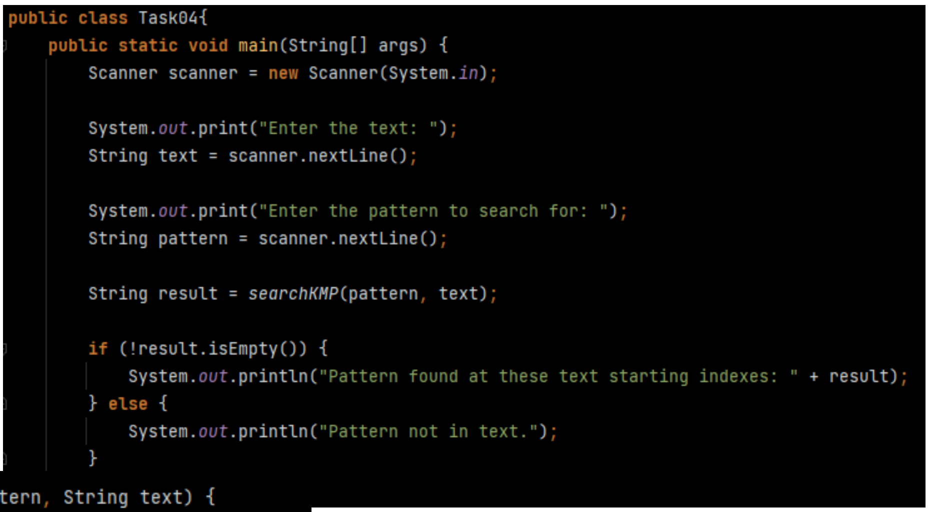
Output:



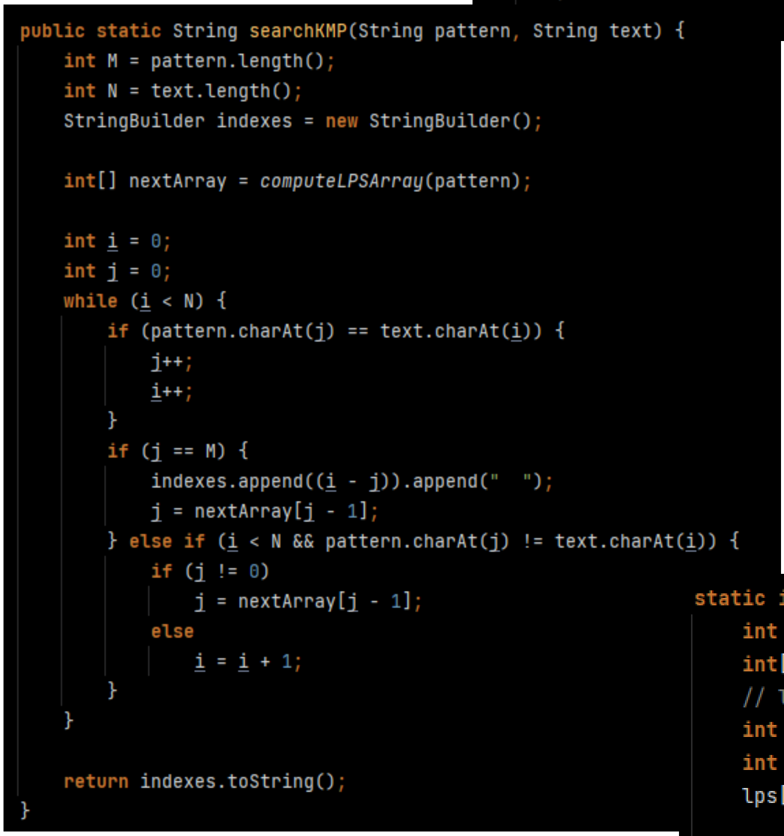
```
The next array for 'ABCDE' is [-1, 0, 0, 0, 0, 0]
The next array for 'AAAAA' is [-1, 0, 1, 2, 3, 4]
The next array for 'ABABAMK' is [-1, 0, 0, 1, 2, 3, 0, 0]
```

Task 4:

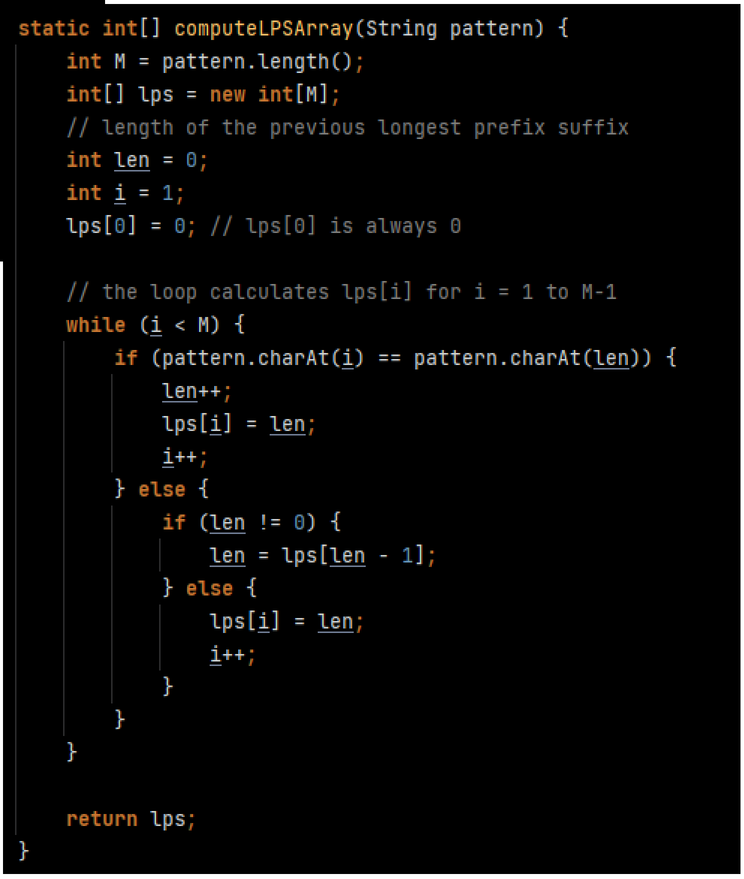
Code:



```java
public class Task04{
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the text: ");
        String text = scanner.nextLine();

        System.out.print("Enter the pattern to search for: ");
        String pattern = scanner.nextLine();

        String result = searchKMP(pattern, text);

        if (!result.isEmpty()) {
            System.out.println("Pattern found at these text starting indexes: " + result);
        } else {
            System.out.println("Pattern not in text.");
        }
```



```java
public static String searchKMP(String pattern, String text) {
    int M = pattern.length();
    int N = text.length();
    StringBuilder indexes = new StringBuilder();

    int[] nextArray = computeLPSArray(pattern);

    int i = 0;
    int j = 0;
    while (i < N) {
        if (pattern.charAt(j) == text.charAt(i)) {
            j++;
            i++;
        }
        if (j == M) {
            indexes.append((i - j)).append("  ");
            j = nextArray[j - 1];
        } else if (i < N && pattern.charAt(j) != text.charAt(i)) {
            if (j != 0)
                j = nextArray[j - 1];
            else
                i = i + 1;
        }
    }

    return indexes.toString();
}
```



```java
static int[] computeLPSArray(String pattern) {
    int M = pattern.length();
    int[] lps = new int[M];
    // length of the previous longest prefix suffix
    int len = 0;
    int i = 1;
    lps[0] = 0; // lps[0] is always 0

    // the loop calculates lps[i] for i = 1 to M-1
    while (i < M) {
        if (pattern.charAt(i) == pattern.charAt(len)) {
            len++;
            lps[i] = len;
            i++;
        } else {
            if (len != 0) {
                len = lps[len - 1];
            } else {
                lps[i] = len;
                i++;
            }
        }
    }

    return lps;
}
```