Magpie Activity 3 Worksheet

Section 1

Directions: Look at the code for the method **findKeyword** and answer the following questions.

Look at the method header.

```
private int findKeyword(String statement, String goal, int startPos)
```

1. What does the keyword **private** at the beginning of the header mean? If you need help refer to PreAP Unit 19 Notes 19.3.

2. What is the purpose of the **int** declaration after the keyword private? If you need help refer to PreAP Unit 13 Notes 13.1.

3. The three variable declarations inside the () are referred to as _____. If you need help refer to PreAP Unit 13 Notes 13.2.

4. What does the parameter **statement** represent?

5. What does the parameter **goal** represent?

6. What does the parameter **startPos** represents?

Look at the next line of code.

```
String phrase = statement.trim();
```

7. What does this line of code do?

Look at the next line of code.

```
int pos = phrase.toLowerCase().indexOf(goal.toLowerCase(), startPos);
```

8. What does this line of code do?

9. When does the while loop in the next code segment stop executing?

Look at the first if statement.

```
if (pos > 0)
{
   before = phrase.substring (pos - 1, pos).toLowerCase();
}
```

10. What is a **local variable**? If you need help refer to PreAP Unit 10 Notes 10.2.

11. What does the local variable **before** store after the code has completed?

Look at the next if statement.

```
if (pos + goal.length() < phrase.length())
{
   after = phrase.substring(pos + goal.length(),
           pos + goal.length() + 1).toLowerCase();
}
```

12. What does the local variable after store after the code has completed?

Look at the next if statement.

```
cif (((before.compareTo ("a") < 0 ) || (before.compareTo("z") > 0)) &&
   ((after.compareTo ("a") < 0 ) || (after.compareTo("z") > 0)))
{
    return pos;
}
```

This is our first introduction to the String class's **compareTo** method. The purpose of the compareTo method is to compare two strings lexicographically which means by alphabetical order. Since strings are objects we cannot use the < less than or > greater than operators to compare their ordering so we use the compareTo method instead.

Recall that the ASCII chart was created to digitize all of the characters in the English language so that they could be represented in a digital computer. Look at the chart again by going to the following website: http://www.asciichart.com/. Notice that the capital letter "A" is assigned a value of 65, the capital letter "B" is assigned a value of 66, and so forth. Also notice that the lowercase letter "a" is assigned a value of 97, the lowercase letter "b" is assigned a value of 98, and so forth.

All of the characters on the chart are ordered. Therefore when comparing "A" to "B" the "A" is less than "B" because the ASCII value for "A" is 65 and the ASCII value of "B" is 66. When comparing "A" to "a" which one has a smaller ASCII value? Yes, "A", therefore "A" is less than "a" because "A" has an ASCII value of 65 and "a" has an ASCII value of 97.

Now back to the compareTo method. Let's look at a simple example of the compareTo method in action.

```
String s1 = "arm";
String s2 = "axe";
int ans = s1.compareTo(s2);
```

Here is a key point. The compareTo method returns an integer value with three possible values. If the string s1 is less than the string s2 then the compareTo method returns a negative number, if the string s1 is greater than s2 then the compareTo method returns a positive number, and if the two strings are equal the compareTo method returns a value of 0. You might be asking yourself what actual numbers does the method return? The answer is "It does not matter". To use the compareTo method effectively all you need to know is that the value returned by the method is either negative, positive, or zero. Now if you still want to know how the String class's compareTo method calculates the return value you can look it up in String class's API documentation.

In our example above the value returned to the variable ans is a negative number. The compareTo method is designed to check each letter of the two strings one at a time until it determines the order. Since the first letter of both strings is an "a" the compareTo method looks at the second letter of each string. It determines that "r" is less than "x" therefore the string "arm" is less than the string "axe" and the method returns a negative number.

Look at the last if statement from the findKeyWord method again.

```
if (((before.compareTo ("a") < 0 ) || (before.compareTo("z") > 0)) &&
   ((after.compareTo ("a") < 0 ) || (after.compareTo("z") > 0)))
{
    return pos;
}
```

13. When is this if statement true?

```
pos = phrase.indexOf(goal.toLowerCase(), pos + 1);
```

Look at the next line of code.
This code is executed only if the previous if statement was false. It uses indexOf to search for another occurrence of the goal string within the phrase.

14. What is the purpose of the **return -1** statement at the end of the method?

Section 2

Directions: Trace the following method calls and write the value of each of the variables **pos**, **before**, and **after** each time the program control reaches the point in the method indicated by the comment:

```
/* determine the values of pos, before, and after at this point in the method. */
```

Example

```
findKeyword("yesterday is today's day before.", "day", 0);
```

Iteration	*pos*	*before*	*after*
1	6	"r"	" "
2	15	"o"	"'"
3	21	" "	" "

1. Look at the last line of the trace table. What value does the method **findKeyword** return and what does this value mean?

```
findKeyword("She's my sister", "sister", 0);
```

Iteration	*pos*	*before*	*after*

2. Look at the last line of the trace table. What value does the method **findKeyword** return and what does this value mean?

```
findKeyword("Brother Tom is helpful", "brother", 0);
```

Iteration	*pos*	*before*	*after*

3. Look at the last line of the trace table. What value does the method **findKeyword** return and what does this value mean?

```
findKeyword("I can't catch wild cats.", "cat", 0);
```

Iteration	*pos*	*before*	*after*

4. Look at the last line of the trace table. What value does the method **findKeyword** return and what does this value mean?

```
findKeyword("I know nothing about snow plows.", "no", 0);
```

Iteration	*pos*	*before*	*after*

5. Look at the last line of the trace table. What value does the method **findKeyword** return and what does this value mean?