SOFTWARE DEVELOPMENT: THE ANAGRAM PROBLEM

Strings, Arrays, And ASCII Conversions

Delroy A. Brinkerhoff

SIMPLE ANAGRAM

- "An **anagram** is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once."
    - https://en.wikipedia.org/wiki/Anagram
- Simple example:
    - See the quick red fox jump over the lazy brown dog.
    - abcddeeeeeefghhijklmnoooopqrrrsttuuvwxyz

CLEVER ANAGRAM

- Letter case (upper vs. lower), spaces, and punctuation are not considered
- Short
  - Dormitory
  - Dirty Room
- From someone with way too much spare time:
  - To be or not to be: that is the question, whether its nobler in the mind to suffer the slings and arrows of outrageous fortune.
  - In one of the Bard's best-thought-of tragedies, our insistent hero, Hamlet, queries on two fronts about how life turns rotten.

THE BASIC ANAGRAM PROBLEM

- Prompt the user to enter two strings. The second string is potentially an anagram of the first
- Input the two strings into two variables input1 and input 2
- Test the two strings to see if they form an anagram
- Print a simple message stating that the string are or are not an anagram

SOLVING THE ANAGRAM PROBLEM

- Developing a solution for a program is like solving a series of story problems
- Sub-problems:
  - Data input
  - Convert each string to a standard form:
    - No spaces or punctuation, and all one case (upper or lower – it doesn't matter)
  - Count all occurrences of each letter
  - Compare all of the counts
  - The inputs are an anagram if all corresponding counts are equal

CREATING A STANDARD FORM (CLEANING UP THE INPUT)

```
define the variable phrase and initialize it to empty

for each character, c, in input
{
    if c is an alphabetic letter
    {
        make c lower case
        append c to phase
    }
}
```

THE COUNTING *IDEA*

```
define and initialize 26 counters:
a_count = 0, b_count = 0, ..., z_count = 0

for each letter, c, in phrase
{
       if (c == 'a')
              a_count1++;
       else if (c == 'b')
              b_count1++;
       . . . .
       else
              z_count1++;
}
```

TESTING FOR AN ANAGRAM: COMPARING LETTER COUNTS

```
if (a_count1 == a_count2 && b_count1 == b_count2 &&
             . . . && z_count1 == z_count2)
      cout << "The phrases form an anagram\n";
else
      cout << "The phrases DO NOT form an anagram\n";
```

REPLACING DISCRETE COUNTERS WITH AN ARRAY

```
for each letter, c, in a standardized phrase
{
        if (c == 'a')
                count[0]++;
        else if (c == 'b')
                count[1]++;
        . . . .
        else
                count[25]++;
}
```

ASCII CONVERSIONS: FROM NUMBER TO CHARACTER

- ASCII ‘0’ is 48; ASCII ‘9’ is 57
- Number to convert is in n
  - n % 10 is the lowest order (one’s) digit
  - n % 10 + ‘0’ is the ASCII code for the numeric value of the one’s digit
  - (char) (n % 10 + ‘0’) is the ASCII letter corresponding to the one’s digit

ASCII CONVERSIONS: FROM CHARACTER TO NUMBER

- ASCII ‘a’ is 97; ASCII ‘z’ is 122
- ASCII ‘A’ is 65; ASCII ‘Z’ is 90
- c - ‘a’ is an integer in 0 - 25

```
count[26]{};
for i = 0 to the end of phrase
   count[phrase[i] - 'a']++;
```

COMPARING THE COUNTS

```
if (a_count1 == a_count2 &&
    b_count1 == b_count2 &&
        . . .
    z_count1 == z_count2)
        cout << "An anagram\n";
else
        cout << "NOT an anagram\n";
```

```
for (int i = 0; i < 26; i++)
        if (count1[i] != count2[i])
                return false;
return true;
```