YEAH Assignment 1

Getting your C++ Legs

Welcome to YEAH

- Your Early Assignment Help!
  - Conceived many moons ago to give students a boost when starting assignments early
  - Have also proved to be helpful for students starting later!
- We'll review the assignment, and I'll give helpful hints / tips – please ask questions!
- If you can (and are not a minor) please come to the **live** session!

Welcome to YEAH

- What I **can** do:
  - Clarify funkiness (or try my best to do so) on the handout
  - Give important insight / highlight common pitfalls on the assignment
  - Memes
- What I **can't** do:
  - Answer questions like "How exactly do I implement this?"
  - Fly

Today’s Agenda

1. Assignment logistics
2. Part I: Perfect Numbers
3. Part II: Soundex

Assignment 1 Logistics

- Due Tuesday, June 30th, 11:59PM (in your local time zone!)
  - This is in 6 days from now! Better get cracking!
  - The grace period for this assignment is 24 hours – late submissions will be accepted until June 31st, 11:59PM.
- You must complete this assignment **individually**. Please read more about what kinds of collaboration are and are **not** permitted on the course website.

Any Questions About Logistics?

Part 1: Perfect Numbers

- A warmup program to get your cpp bearings. You'll write an efficient algorithm that finds perfect numbers!
  - A perfect number is a number whose factors add to the number!
    - 6 = 1 + 2 + 3;   28 = 1 + 2 + 4 + 7 + 14
- 2 fundamental pieces, **coding** and **short answer**.
  - Designed to complement each other and help your understanding!

$$
\begin{aligned}
6 = 2^1(2^2-1) &= 1+2+3, \\
28 = 2^2(2^3-1) &= 1+2+3+4+5+6+7 = 1^3+3^3, \\
496 = 2^4(2^5-1) &= 1+2+3+\cdots+29+30+31 \\
&= 1^3+3^3+5^3+7^3, \\
8128 = 2^6(2^7-1) &= 1+2+3+\cdots+125+126+127 \\
&= 1^3+3^3+5^3+7^3+9^3+11^3+13^3+15^3, \\
33550336 = 2^{12}(2^{13}-1) &= 1+2+3+\cdots+8189+8190+8191 \\
&= 1^3+3^3+5^3+\cdots+123^3+125^3+127^3.
\end{aligned}
$$

Euclid discovered a pretty cool relationship about perfect numbers… we'll come back to this.

Part 1: Perfect Numbers

- The first thing that you'll do in this assignment is examine a pre-written algorithm that finds perfect numbers.

```
/* This function performs an exhaustive search for perfect numbers.
 * It takes as input a number called `stop` and searches for perfect
 * numbers between 1 and `stop`. Any perfect numbers that are found will
 * be printed out to the console.
 */
void findPerfects(long stop)
{
    for (long num = 1; num < stop; num++) {
        if (isPerfect(num)) {
            cout << "Found perfect number: " << num << endl;
        }
        if (num % 10000 == 0) cout << "." << flush;
    }
    cout << "Done searching up to " << stop << endl;
}
```

```
long divisorSum(long n) {
    long total = 0;
    for (long divisor = 1; divisor < n; divisor++) {
        if (n % divisor == 0) {
            total += divisor;
        }
    }
    return total;
}

/* This function is provided a number n and returns a boolean
 * (true/false) value representing whether or not the number is
 * perfect. A perfect number is a non-zero positive number whose
 * sum of its proper divisors is equal to itself.
 */
bool isPerfect (long n) {
    return (n != 0) && (n == divisorSum(n));
}
```

Part 1: Perfect Numbers

- isPerfect() calls a routine called divisorSum(). This approach is called an **Exhaustive Algorithm**. These algorithms attempt to find solutions by doing **all** computations without optimizations.

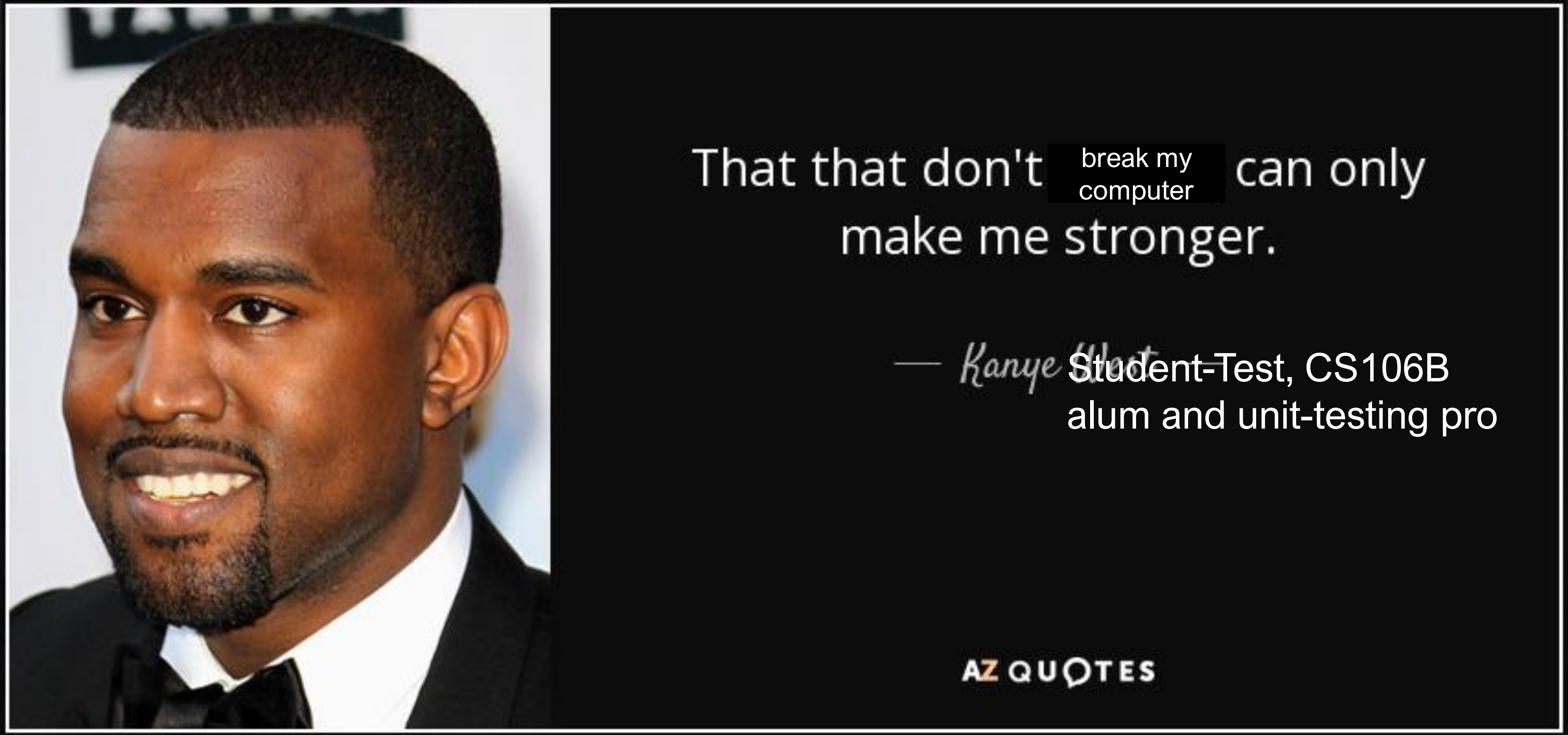
```
// Return sum of all divisors of n, excluding self
long divisorSum(long n)
{
    long total = 0;

    for (long divisor = 1; divisor < n; divisor++) {
        if (n % divisor == 0) {
            total += divisor;
        }
    }
    return total;
}
```

Part 1: Perfect Numbers

- Exhaustive algorithms get the job done, but they’re not the fastest. We can do better!
- But first, it’s time for…

Kanye StudentTest's Testing Overview!



Running Tests in CS106B

- An important part of CS106B is **testing**, the ability to write small pieces of functionality that you can test.
- There are 4 functions you'll be frequently using this quarter, TIME_OPERATION, EXPECT, EXPECT_EQUAL, and EXPECT_ERROR.
  - You will create STUDENT_TESTs and use TIME_OPERATION, EXPECT, EXPECT_EQUAL, and EXPET_ERROR to verify the correctness of your functions!
- TIME_OPERATION (inputsize, operation) function call times how long it takes to perform function OPERATION on INPUTSIZE elements, and reports these numbers to the console.
- Check out the use of the EXPECT functions use on the next slide!

Running Tests in CS106B

```
int returnFive (int num) {
    if (num == -1) error ("Error, cannot process -1!");
    return 5;
}
// EXPECT tests the provided predicate.
STUDENT_TEST ("Verifies that returnFive returns five with EXPECT") {
    EXPECT (returnFive (0) == 5);
}
// EXPECT_EQUAL compares two values.
STUDENT_TEST ("Verifies that returnFive returns five with EXPECT_EQUAL") {
    EXPECT_EQUAL (returnFive (0), 5);
}
// EXPECT_ERROR passes if and only if the code it runs throws an error.
STUDENT_TEST ("Verifies that returnFive throws an error on bad input with EXPECT_ERROR") {
    EXPECT_ERROR (returnFive (-1));
}
```

Questions about testing?

```
int returnFive (int num) {
    if (num == -1) error ("Error, cannot process -1!");
    return 5;
}
// EXPECT tests the provided predicate.
STUDENT_TEST ("Verifies that returnFive returns five with EXPECT") {
    EXPECT (returnFive (0) == 5);
}
// EXPECT_EQUAL compares two values.
STUDENT_TEST ("Verifies that returnFive returns five with EXPECT_EQUAL") {
    EXPECT_EQUAL (returnFive (0), 5);
}
// EXPECT_ERROR passes if and only if the code it runs throws an error.
STUDENT_TEST ("Verifies that returnFive throws an error on bad input with EXPECT_ERROR") {
    EXPECT_ERROR (returnFive (-1));
}
```

Back to our algorithm!

Can we do better?

- After you write some tests for the existing code, your next step will be to **improve** the perfect number-finding algorithm!
  - In its current state, the helper function that computes the sum of **all** divisors of a number loops through all values between 1 and n - 1.

```
// Return sum of all divisors of n, excluding self
long divisorSum(long n)
{
    long total = 0;

    for (long divisor = 1; divisor < n; divisor++) {
        if (n % divisor == 0) {
            total += divisor;
        }
    }
    return total;
}
```

Can we do better?

- It turns out, we only have to loop through numbers 1 -> sqrt(n) to get all divisors of n!
  - Ex. For the number 6, who has perfect factors 1, 2, and 3, we only need to consider 1 and 2 (or 1 -> sqrt(6)); **we can get the complementary factor via (n / divisor)** -> 3 = 6 / 2.
  - Ex. 28 -> [1], [2 AND 28 / 2], [4 AND 28 / 4]
- Suddenly, our work has gone from (n) computations to (sqrt(n)) computations. Nice job!

Speed Racer giving *you* a thumbs-up!

SmarterSum

- You're going to write this into a program called smarterSum() that has the same functionality as divisorSum, but only loops through sqrt(n) numbers!
- Some tips/tricks
  - There are a number of edge cases to consider now that you're not examining all numbers. Think about how you might handle negative values on n, 0, 1, or square roots! **This is the kind of thinking that you should <u>always</u> have when testing your code!**

Questions about smarterSum?

Mersenne Primes

- A Mersenne Prime is a special prime number that is **one less than a power of two.**
  - 31 = 2^5 - 1.
- Euclid discovered a cool property of these numbers:
  - If 2^k – 1 is prime, then 2^(k - 1) * (2^k - 1) is a **perfect number!**
- Can we make our perfect number algorithm even better?

findNthPerfectEuclid

- You’re going to use Euclid’s discovery to write a routine that finds the **nth** perfect number. More specifically, you’ll be implementing the function

long findNthPerfectEuclid(long n)

that returns a **long** signifying the perfect number of order **n.**

findNthPerfectEuclid

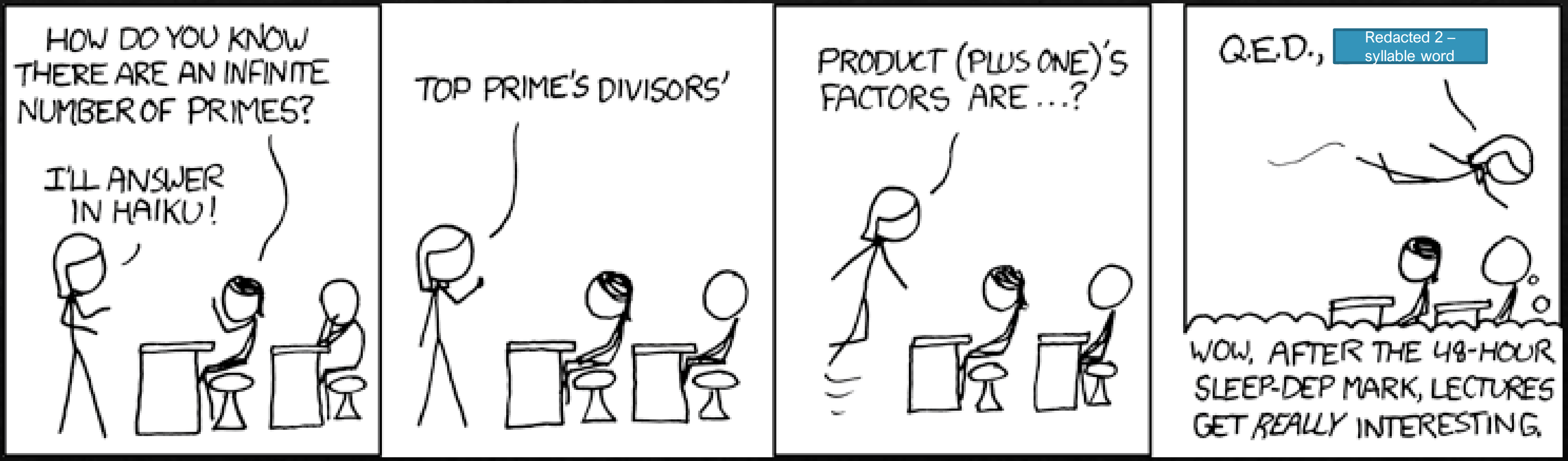
- Here’s how we’d like you to approach this problem:
  1. Start by setting a variable k = 1
  2. Calculate **m** = 2^k – 1 (use cpp library pow() function!)
  3. Determine whether **m** is prime or composite (write an isPrime function that loops through possible divisors!)
  4. If **m** is prime, (it’s a Mersenne Prime!!) calculate 2^(k - 1) * (2^k - 1). This is the associated **perfect number!**
  5. Increment **k** and repeat until you’ve found the **nth** perfect number!

(I’ll wait here for a second ☺)

findNthPerfectEuclid

- A few things you should know:
  - We’re using **long** instead of **int** here because these numbers can get really big!
  - If you’re on a windows machine, you won’t be able to find perfect numbers greater than ~6, but on mac you can find a few more!
    - Want to know why? Take CS107, or look up 32bit vs 64bit architectures!

Questions about findNthPerfectEuclid?



source, xkcd

That’s Part I!

- Congrats! You made it past part I. Now it’s time to do the big one!

source, google images

Part II, Soundex Search

- In this final part, you'll be writing an algorithm that takes a last name and turns it into a **soundex code**, which is a 4-digit pseudo-phonetic representation of a last name.
  - I say pseudo-phonetic because the soundex algorithm we're going to use is pretty crappy, and it doesn't account for most pronounciations.
  - That being said, the US census uses soundex codes! Wonder what that says about our language tolerance ☹

Soundex()

- Here are some examples of soundex conversions:
  - Zelenski -> Z452
  - Lee -> L000 (see what I mean?)

Soundex()

- The good news is, the **string soundex (string s)** routine is actually pretty straightforward. Here are the steps!

1. Discard all non-letters from the name. The isalpha() function will help with this!
2. Save the first letter of the name, and convert to uppercase if necessary.
3. Encode all letters using the depicted table.
4. Coalesce adjacent duplicate numbers (222025 would become 2025)
5. Replace the first number of the encoding with the **saved** first letter of the name.
6. Remove ALL zeroes from the code.
7. Format to length 4 by either truncating the code or padding zeroes at the end.

Digit	represents the letters
0	A E I O U H W Y
1	B F P V
2	C G J K Q S X Z
3	D T
4	L
5	M N
6	R

Soundex()

- Let’s see an example with Zelenski!

Soundex()

- The good news is, the soundex routine is actually pretty straightforward. Here are the steps!

1. Discard all non-letters from the name. The isalpha() function will help with this!
2. Save the first letter of the name, and convert to uppercase if necessary.
3. Encode all letters using the depicted table.
4. Coalesce adjacent duplicate numbers (222025 would become 2025)
5. Replace the first number of the encoding with the **saved** first letter of the name.
6. Remove ALL zeroes from the code.
7. Format to length 4 by either truncating the code or padding zeroes at the end.

Zelenski

Soundex()

- The good news is, the soundex routine is actually pretty straightforward. Here are the steps!

1. Discard all non-letters from the name. The isalpha() function will help with this!
2. Save the first letter of the name, and convert to uppercase if necessary.
3. Encode all letters using the depicted table.
4. Coalesce adjacent duplicate numbers (222025 would become 2025)
5. Replace the first number of the encoding with the **saved** first letter of the name.
6. Remove ALL zeroes from the code.
7. Format to length 4 by either truncating the code or padding zeroes at the end.

Zelenski

Zelenski Z

Soundex()

- The good news is, the soundex routine is actually pretty straightforward. Here are the steps!

1. Discard all non-letters from the name. The isalpha() function will help with this!
2. Save the first letter of the name, and convert to uppercase if necessary.
3. Encode all letters using the depicted table.
4. Coalesce adjacent duplicate numbers (222025 would become 2025)
5. Replace the first number of the encoding with the **saved** first letter of the name.
6. Remove ALL zeroes from the code.
7. Format to length 4 by either truncating the code or padding zeroes at the end.

Zelenski

Zelenski Z

20405220

Soundex()

- The good news is, the soundex routine is actually pretty straightforward. Here are the steps!

1. Discard all non-letters from the name. The isalpha() function will help with this!
2. Save the first letter of the name, and convert to uppercase if necessary.
3. Encode all letters using the depicted table.
4. Coalesce adjacent duplicate numbers (222025 would become 2025)
5. Replace the first number of the encoding with the **saved** first letter of the name.
6. Remove ALL zeroes from the code.
7. Format to length 4 by either truncating the code or padding zeroes at the end.

Zelenski

Zelenski Z

20405220

2040520

Soundex()

- The good news is, the soundex routine is actually pretty straightforward. Here are the steps!

1. Discard all non-letters from the name. The isalpha() function will help with this!
2. Save the first letter of the name, and convert to uppercase if necessary.
3. Encode all letters using the depicted table.
4. Coalesce adjacent duplicate numbers (222025 would become 2025)
5. Replace the first number of the encoding with the **saved** first letter of the name.
6. Remove ALL zeroes from the code.
7. Format to length 4 by either truncating the code or padding zeroes at the end.

Zelenski

Zelenski Z

20405220

2040520

Z040520

Soundex()

- The good news is, the soundex routine is actually pretty straightforward. Here are the steps!

1. Discard all non-letters from the name. The isalpha() function will help with this!
2. Save the first letter of the name, and convert to uppercase if necessary.
3. Encode all letters using the depicted table.
4. Coalesce adjacent duplicate numbers (222025 would become 2025)
5. Replace the first number of the encoding with the **saved** first letter of the name.
6. Remove ALL zeroes from the code.
7. Format to length 4 by either truncating the code or padding zeroes at the end.

Zelenski

Zelenski

20405220

2040520

Z040520

Z452

Soundex()

- The good news is, the soundex routine is actually pretty straightforward. Here are the steps!

1. Discard all non-letters from the name. The isalpha() function will help with this!
2. Save the first letter of the name, and convert to uppercase if necessary.
3. Encode all letters using the depicted table.
4. Coalesce adjacent duplicate numbers (222025 would become 2025)
5. Replace the first number of the encoding with the **saved** first letter of the name.
6. Remove ALL zeroes from the code.
7. Format to length 4 by either truncating the code or padding zeroes at the end.

Zelenski

Zelenski

20405220

2040520

Z040520

Z452

Z452

Done!

Soundex()

- You'll implement this routine in the **string soundex(string s)** routine. You'll take a surname as a string and return the associated soundex code!
  - To do this, **please** decompose! The aforementioned steps can each be their own functions, and that way you'll be able to **write tests for each helper function you write!**
  - If you are able to test your helper functions, it will be much easier to pinpoint potential bugs in your program!

Soundex()

- Here are some tips for Soundex()
  - You'll be doing lots of work with **strings** and **chars** here. Know how to **index into** a string (like str[i]), but be wary that **strings** and **chars** are different types!
    - Strings use "double quotes" and chars use 's'ingle quotes.
    - In case you need to convert between the two, strlib.h includes conversion functions!
  - When converting chars to soundex numbers, ensure that your code is **case insensitive!**
  - If you ever need to convert a char to upper case, the **int toupper(int c)** function in <cctype> will **return** the uppercase version of the char you pass in.

  - wait… why does **int toupper(int c)** deal with integers then….

It’s time for…

Charole Baskin's brief foray into char representation via the ASCII set!

-**Char**ole Baskin, 106B alum
and mariticide suspect

How do we represent characters?

- Let's face it, there are a lot of unique chars out there. When you couple that with the existence of fonts*, you get a data representation nightmare – **how do you represent chars?**
- The computing world decided to get together to create a **standard number representation for popular chars** (128 of them!). **Each char would correspond to an integer in a table called the ASCII set.**
- For example, 'A' -> 65, and 'a' -> 97.

*leave a comment if you know what font this is!

What does this code print?

```
string s = "apple";
cout << toupper(s[0]) << endl;
```

How do we represent characters?

- You need to be careful that you're not working directly with integers when you work with characters!
  - If a function returns an int, be sure **you're storing the data as a character** so that it can be read properly!

```
string s = "apple";
char firstLetter = toupper(s[0]);
cout << firstLetter << endl;
```

Soundex()

- Questions about soundex?
- **Please** remember to decompose the steps on this one ☺

source, xkcd, depicting me in a few years – I'm terrible with names

Soundex Search

- Now it's time to put your soundex function to the test! You will write the function **void soundexSearch(string filePath),** that allows the **user** to find the soundex code for a given name, along with **other** names in the database represented by the filename **filePath** with the same soundex code.
- Your interaction with the user should match this exactly:

```
Read file res/surnames.txt, 26409 names found.

Enter a surname (RETURN to quit): Zelenski
Soundex code is Z452
Matches from database: {"Zelenski", "Zelnick", "Zelnik", "Zelnis", "Zielonka"}

Enter a surname (RETURN to quit): troccoli
Soundex code is T624
Matches from database: {"Therkelsen", "Torkelson", "Trakul", "Traxler", "Trisal", "Troc
coli", "Trockel", "Troxel", "Troxell", "Trujillo", "Turkel"}

Enter a surname (RETURN to quit):
All done!
```

This line is done for you ☺

Soundex Search

- You've already been provided with the code that reads the provided file into a **vector**. If you're not familiar, a **vector** is like a Java **ArrayList** or a python **List**: we use it to store things!
  - Here, the vector **lines** stores all names from the file!

```
void soundexSearch(string filepath)
{
    // The proivded code opens the file with the given name
    // and then reads the lines of that file into a vector.
    ifstream in;
    Vector<string> lines;

    if (openFile(in, filepath)) {
        readEntireFile(in, lines);
    }
    cout << "Read file " << filepath << ", " << lines.size() << " names found." << endl;
```

Soundex Search

- You'll need to **repeatedly** prompt the user for a name (think while loop!)
  - If the user enters empty string, (return) break out of the loop!
- Once you get a string, compute and print its soundex code!
- Then, loop through the vector of strings, compute the soundex for each line, and store any whose soundex match yours into a new vector, which you will print after you've found all matches!

```
Read file res/surnames.txt, 26409 names found.

Enter a surname (RETURN to quit): Zelenski
Soundex code is Z452
Matches from database: {"Zelenski", "Zelnick", "Zelnik", "Zelnis", "Zielonka"}

Enter a surname (RETURN to quit): troccoli
Soundex code is T624
Matches from database: {"Therkelsen", "Torkelson", "Trakul", "Traxler", "Trisal", "Troc
coli", "Trockel", "Troxel", "Troxell", "Trujillo", "Turkel"}

Enter a surname (RETURN to quit):
All done!
```

Vector Semantics

```
Vector<string> names;

// Append to a vector
names += "Trip";
string str = "Kylie";
names.add(str);

// index-based for loop!
for (int i = 0; i < names.size(); i++) {
    string name = names [i];
}

// for-each loop!
for (string name : names) {

}
```

Soundex Search

- A few tips / tricks:
  - Use the getline() function from "simpio.h" to get user input!
  - You need to sort the names in the vector alphabetically. No need to fuss; before you print the vector, simply call **vector_name.sort()** (sorts in place)
  - You can cout vectors just like strings, and they'll present themselves like you see in the example!
  - Give "Stanford string cpp" a google and take a foray through the Stanford cpp library for strings. There are some really helpful functions out there!

Questions About Soundex Search?