FUNCTIONS LAB 1: USING LIBRARY FUNCTIONS

1. Simple Functions

"So, do you want to take a leap of faith? Or become an old man, filled with regret, waiting to die alone!"

–Saito, from Inception[1]

1.1. **Lab objectives.**

In this lab you will learn to use library functions, and then write your own functions. By handling different errors from functions' return values, you will familiarize yourself with return value checking for various functions, an important secure coding concept.

1.2. **Introduction.**

In the beginning of your school year, you enroll in a programming seminar. You and your friend Sheldon decide to make your own calculator as the term project. Being a super geek, Sheldon insists on building the whole thing from scratch. Somehow it doesn't sound right to you, so you scratch your head and say, "There must be a better way…"

You are correct. In order to write their programs, programmers build on pre-written code from libraries. A library is a collection of prewritten tools that perform a set of useful operations [8]. To quote a programming textbook, "Today, it is not at all unusual for 90 percent or more of a program to consist of library code" [8]. The libraries contain functions. For now, we can think of the functions as black boxes where we put in some values and get back some values. We don't know what is actually going on inside the boxes, but by a leap of faith we trust them to do the job we expect them to do, just as we expect vending machines to return a can of soda if we input some coins.

Extending the vending machine metaphor a bit further, we cannot expect to get soda out of thin air – we first need to install the vending machine. In the same manner, we need to "install," or in C++ jargon, "include" our libraries into our programs to make use of their functions. The example below shows how we can include these libraries. Including `<iostream>` lets us use functions for input and output, and including `<math.h>` lets us use functions that compute common mathematical operations[1]:

```
#include <iostream>
#include <math.h>
```

[1] For more information on different types of libraries, visit www.cplusplus.com/reference

Going back to your project, you wish to incorporate the cosine function into your calculator. You find the appropriate function in the `math.h` library: the `cos` function. Therefore you write the following code and show it to Sheldon[2]:

```
#include <iostream>
#include <math.h>
using namespace std; // [3]

int main() {
   double rad_value;
   cout << "Please enter a value in radians: ";
   cin >> rad_value;
   double cos_value = cos(rad_value); // This line of code
                                      // uses the cos function.
   cout << "The cosine of " << rad_value << " is "
        << cos_value << endl;
   return 0;
}
```

Sheldon says doubtfully, "Very cool. What does it do?" So you compile the program and get the following result:

```
> g++ -o test test cos.cpp
>./test
Please enter a value in radians:
```

You enter 3.14, then get the cosine value:

```
The cosine of 3.14 is -0.999999
```

Congratulations! You've just successfully completed a part of your project.

1.3. **Exercises.**

1) Love has limits: You work for a new mobile phone company Horizon. You find a bug in the system: people can put as many characters as they want in a single text message! Now is the time to intervene and put a limit of 200 characters per text. Write the missing code to warn the user when they go over 200 characters. (They will perhaps be charged extra. You may wish to use the `strlen` function from the `string.h` library)

---

[2] If you are unfamiliar with `cout` and `cin`, see www.cplusplus.com/doc/tutorial/basic_io/

[3] This line of code is included for similar reasons as including the libraries. For more information, see www.cplusplus.com/doc/tutorial/namespaces/

```
#include <iostream>
#include <string.h>

using namespace std;
#define buf_length 1000
#define text_limit 200

int main() {
   char str[buf_length];
   cin >> str;









   return 0;
}
```

2) Graphing calculator: You decide to add graphing utility to your calculator. Although the C++ standard library doesn't have a convenient package for graphics, you found online a magical library for graphics called `magicgraph.h`. The plot function in `magicgraph.h` library draws a point on the graph given x and y coordinates. For example, `plot(10, 20)` will draw a point at position 10 pixels away horizontally and 20 pixels away vertically from the origin. Write a program to plot the sine function from -2π radians to $+2\pi$ radians, using at least 20 data points. Since you don't *actually* have access to the graphics library, print the values of the x and y coordinates.

2. **More Complex Functions**

"Never trust anything that can think for itself if you can't see where it keeps its brain."
– Mr. Weasley, Harry Potter and the Chamber of Secrets [2]

2.1. **Checking return values.** Working as the security administrator for MI6, you are in charge of establishing a system for Mr. Bond to send secret messages. You choose the RSA encryption algorithm[4] for this purpose. The RSA encryption algorithm encrypts the information using public keys and decrypts it using private keys. Two different prime numbers are required to generate the public keys. The prime numbers are to be chosen randomly for security reasons [3]. One of the public keys is the result of multiplying the prime numbers together. Let's assume that the random number generator gave you two prime numbers, 17 and $2^{6972593}-1$.[5] Since one of these primes is rather large, you express it using the `pow` function in the `math.h` library:

```
#include <math.h>
#include <iostream>

using namespace std;

int main() {
   double public_key, first_prime, second_prime;
   first_prime = pow(2, 6972593)-1;
   second_prime = 17;
   public_key = first_prime * second_prime;
   cout << "Public key: " << public_key << endl;
   // The rest goes here
   return 0;
}
```

You send out the public keys and wait for Mr. Bond's secret message. Instead, he marches into your office the next day in ragged clothes. "Your keys were useless! Did you even bother to run your program?" Shocked, you run the program and get the following result:

```
>g++ -o test RSA.cpp
>./test
Public key: inf
```

---

[4] For more information on RSA encryption visit the following webpage, along with the cited Wikipedia page on RSA encryption:
http://www.cs.washington.edu/homes/reges/rsa/

[5] 10th largest known Mersenne prime [4]. See http://primes.utm.edu/largest.html

Reading closely the reference page for the `pow` function, you realize your error:

> If the magnitude of the result is so large that it cannot be represented in an object of the return type, a range error occurs, returning `HUGE_VAL` with the appropriate sign and setting the value of the global variable `errno` to the `ERANGE` value [5].

The `pow` function returned `HUGE_VAL` instead of the actual gigantic number, which gets printed out as `inf` in the console window. The rest of your program, which relied on correct prime numbers, failed silently.

Trying not to anger the man with a license to kill, you tell him that you will check for return values next time. To assure him, you show him the following fix:

```
#include <math.h>
#include <iostream>

using namespace std;

int main() {
   double public_key, first_prime, second_prime;
   first_prime = pow(2, 6972593) - 1;
   second_prime = 17;
   while (first_prime == HUGE_VAL || second_prime == HUGE_VAL) {
      cout << "Please enter two primes within range: " << endl;
      cin >> first_prime >> second_prime;
   }
   public_key = first_prime * second_prime;
   cout << "Public key: " << public_key << endl;
   // The rest goes here
   return 0;
}
```

Before he leaves, he remarks, "You might want to add the checks to the rest of the program. God knows how many overflows there might be working with those big numbers."

At the end of your rough day, you have learned an important lesson in programming. Library functions are convenient and necessary. However, we need to be aware that functions can return unexpected values—and surprises are always bad in programming. When a longer program crashes it will be harder to tell where the bug is if you did not check return values each time (Imagine having to track down errors like this in ten thousand lines of code!). Always read the descriptions of functions' behaviors, and add checks to ensure reasonable return values even when odd behaviors aren't listed in the references. It's like wearing a helmet when you're riding a bike.

Note on C-style strings and C++ strings before the exercises[6]: There are two kinds of

---

[6] Content based on CS106L [7] and CS106B course readers [8]

strings in the standard C++ library. The older C-style string and newer C++ string. C-style strings are arrays of characters. Although C++ strings are much easier to use, some legacy functions require the use of C-style strings. Exercises 1, 2, and 3 use C-style strings.

2.2. **Exercises.**

1) Nonexistent obsolete: Your friend, an aspiring fiction writer, wrote a modern-day Pride and Prejudice. One day he panics and comes into your office: he seems to have written one word wrong. He says, "I think I wrote 'She's an obsolete beauty' instead of 'She's an absolute beauty'. Can you find it and change it for me?" He hands you over the text file. Write a program that will fix his *possible* typo. You are to write a function that takes a string as its argument, and modifies the string by replacing 'She's an obsolete beauty' by 'She's an absolute beauty', if it occurs. You may wish to use `strstr` and `strncpy` from `string.h` library. Note that the `strstr` function[7] returns a null pointer if the word you are looking for does not exist in the string.

```
void fixStr(char *str) {









}
```

2) Unhappy clients: As a successful businessperson, you manage clients by sending them letters on special occasions. Since you have over a hundred clients, it seems fit that you use a program to write letters for you. You use the function `autogenerate_greetings` within your program:

```
char *autogenerate_greetings(const char *name) {
   int size = 30;
   char * buffer = (char *) malloc(size);
   if (buffer == NULL) return NULL;
   snprintf(buffer, size, "Dear Mr. %s,", name);
   return buffer;
}
```

A few days later you get an angry phone call from Mr.

---

[7] Interestingly, the example given online for this function does not contain a check for the `NULL` pointer. This is bad practice. To view it, visit http://www.cplusplus.com/reference/clibrary/cstring/strstr/

Wolfeschlegelsteinhausenbergerdorff[8] and others, and the number of your clients decreases significantly. What have you done? And how can you fix it? (Hint: `snprintf` is a *safer* function than `sprintf`, because it *only* copies a specified number of characters from the string to a buffer. It also returns the number of characters that it has written to the buffer.[9] Read more at http://libslack.org/manpages/snprintf.3.html)

3) Student records: As a TA for CS1, you are given the table of students enrolled in the class. You need to use their student ID numbers to read in datasets from the school student records. The table is given as strings, and you need ID numbers as `int`s. A table with three sample inputs is given below:

```
 // first value: student name,
// second value: student ID,
// third value: major
char Student_1[] = "Mickey Mouse,123456789,Computer Science";
char Student_2[] = "Mini Mouse,3141592653,Physics";
char Student_3[] = "Goofy T. Dog,271828,History";
```

Another TA has already used the `strtok`[10] function to isolate the ID numbers for you:

```
char ID_1[] = "123456789";
char ID_2[] = "3141592653";
char ID_3[] = "271828";
```

Write a program to convert the strings to ints. Functions you may consider using are:

- `atoi`: converts a string to an int. (`stdlib.h`)
- `snprintf`: can be used to convert an int into a string by using `"%d"` as the format string. (`stdlib.h`)

---

[8] This is the shortened name of an actual person with the longest personal name ever used [6].

[9] Description is as follows: "On success, returns the number of characters that would have been written had `size` been sufficiently large, not counting the terminating null character." [9]

[10] A function from the `string.h` library.

- `strcmp`: compares two strings, returning 0 if they are identical, nonzero value otherwise. (`string.h`)
- `strlen`: returns the length of a string. (`string.h`)

Fill out the following function:

```
// Convert id string to int.  Return -1 if there was a problem.
int convertId(char *id) {









}
```

4) A case study: Y2K problem.[11] Because most computer programs used two digits to represent the year, they faced a horrifying problem as the year 2000 approached. When 99 turns to 00, anything could happen! Perhaps banking programs will think we went back in time to 1900, and give strange values for interests. Write a fix to the following code (one line fix!):

```
#include <iostream>
#include <math.h>
#include <stdio.h>

using namespace std;

#define interest_rate 0.01

double calculate_current_value(double initial_value, int years_in_bank);

int main() {
   double initial_value, current_value;
   int current_year, year_deposit, years_in_bank;

   cout << "Please enter the current year "
        << "(drop the first two digits, enter the last two): " << endl;
   cin >> current_year;

   cout << "Please enter the year of deposit "
        << "(enter the last two digits): " << endl;
   cin >> year_deposit;

   cout << "Please enter the initial value of the deposit: " << endl;
   cin >> initial_value;

   years_in_bank = current_year - year_deposit;
   current_value = calculate_current_value(initial_value, years_in_bank);
   cout << "The customer's current deposit is: " << current_value << endl;

   return 0;
}

double calculate_current_value(double initial_value, int years_in_bank) {
   return initial_value * pow(1 + interest_rate, years_in_bank);
}
```

[11] Did you know that a problem similar to Y2K is waiting for us? The year 2038 problem is a similar sort of time-calculation wrap-around bug. All the more reason to check return values when using functions! For more information visit:
http://www.rogermwilcox.com/Y2038.html

REFERENCES

[1] *Inception*. dir. Christopher Nolan. Warner Bros, 2010, Film.

[2] Rowling, J.K. Harry Potter and the Chamber of Secrets. New York: Scholastic, 1999. Print.

[3] "RSA." Wikipedia, The Free Encyclopedia. Wikimedia Foundation, Inc. 25 July 2011. Web. 8 Aug. 2011.

[4] Caldwell, Chris. The Largest Known Primes. The University of Tennessee at Martin, 8 Aug. 2011. Web. 8 Aug. 2011.

[5] "pow - C++ Reference." cplusplus.com. cplusplus.com, 2000-2011., n.d. Web. 8 Aug. 2011.

[6] "Wolfe+585, Senior." Wikipedia: The Free Encyclopedia. Wikimedia foundation, Inc. 14 July 2011. 8 Aug 2011.

[7] Schwarz, Keith. "CS106L Standard C++ Programming Laboratory Course Reader." www.keithschwarz.com. Stanford University, 2009. Web. 12 Sep 2011. http://www.keithschwarz.com/coursereader.pdf.

[8] Roberts, Eric, and Zelenski Julie. "Programming Abstractions in C++." Stanford University, Sept-12-2011. Web. 12 Sep 2011. http://www.stanford.edu/class/cs106b/materials/CS106BX-Reader.pdf.

[9] "snprintf." libslack.org. raf, n.d. Web. 8 Aug. 2011.