Problem Set 1

Due by 11:55 pm on Monday, September 12, 2016.

1 Assignment Goals

1. Learn how to prepare and build C++ code on the Zoo.
2. Learn how to customize and use `tools.cpp` and `tools.hpp` in your own code.
3. Learn good practices for labeling every code file with appropriate date, authorship, and acknowledgments of code fragments taken from elsewhere.
4. Learn conventions used in this course regarding the fqorm of the main program, use of `banner()` and `bye()` from the tools library, compiler switches, include guards, and so forth.
5. Learn simple uses of C++ strings.
6. Learn how to do simple C++ I/O with validity check.
7. Learn how to use C library time functions from within C++.
8. Learn how to report a fatal error by throwing a `Fatal()` exception.
9. Learn how to test and submit your code on the Zoo.

2 Problem

You are to write a program called `aboutme` that, when run, prompts the user to enter their first name, last name, and year of birth. The program then prints out the first name, last name, and age, which we take to be the difference between the current year and the year of birth. Here's what a sample run for me might look like:

```
------------------------------------------------------------
Michael J. Fischer
CPSC 427/527
Wed Jan 27 2016 15:56:09
------------------------------------------------------------
Please enter your first name: Michael
Please enter your last name: Fischer
Please enter the year of your birth: 1942
Michael Fischer becomes 74 years old in 2016.

------------------------------------------------------------
Normal termination.
```

The lines before the first prompt are printed by `banner()`. The lines after the age line are printed by `bye()`.

3 Programming Notes

1. Copy the `tools` files from `/c/cs427/code/tools/` into your working directory. Customize them by editing in your own name in place of the generic "Ima Goetting Closeau". Be sure to submit *your* tools files along with your code and other required files. Read Chapter 1 of the textbook for more information about tools, but be warned that my version of tools differs somewhat from what is in the textbook.

2. Your main function should be exactly as follows:

```
int main() {
    banner();
    try {
        run();
    }
    catch (Fatal& e) {
        cerr << "Catching Fatal exception\n" << e.what() << endl;
    }
    catch (...) {
        cerr << "Uncaught exception" << endl;
    }
    bye();
}
```

   Your own code will go into the file `main.cpp` before and/or after function `main()`. You will need to define the function `run()`.

   We will study later how the `try` and `catch` blocks work. This code as written will catch any exceptions thrown by `run()` or by any function that `run()` calls. A statement inside `run()` of the form `throw Fatal(format, data1, data2, ...)` will be caught by the first `catch` block and will cause the same string to be printed as `printf(format, data1, data2, ...)` would print in C . For example,

```
throw Fatal("Can't open file %s", "data.in");
```

   would cause the two lines to be printed:

```
Catching Fatal exception
Can't open file data.in
```

3. You should use the standard C++ iostreams facility for your I/O. To use it, one must `#include` the header file `iostream`. `tools.hpp` already does this for you, so all you need in `main.cpp` is the statement `#include "tools.hpp"`. The standard input stream is `cin`; the standard output stream is `cout`. The input operator is `>>`; the output operator is `<<`. Further information and examples are in the textbook.

4. There are two ways to store a string in C++ . First is as a `char` array as one does in C . The other is to use the standard `string` library. To use it, you must `#include` the header file `string` in your code. (`tools.hpp` already does this for you.) This makes `string` available as a new type, which you can then use to declare variables and parameters.

A variable of type `string` acts just like one would expect. You can store a string literal of any length into it. You can copy one string to another using the assignment operator. You can print it using the `<<` operator and read a whitespace-delimited string into it using the `>>` operator. A `string` manages its own storage so that you do not have to worry about allocating storage. You should use `string` variables to store the first and last names to be read in.

5. Compute the current year by using the standard C library functions `time()` and `localtime()`. `time()` reads the clock and returns a value of type `time_t` that is the number of seconds since the beginning of year 1970. To find the year from a `time_t` value, use `localtime()` to create a `struct tm` broken-down data structure. This contains an `int` field called `tm_year` that sounds like it ought to be the year (like 2016), but it's actually the year minus 1900. To use these functions in C++, you would `#include` the header file `ctime`. (This is the C++ version of the C header file `time.h` and is included by `tools.hpp`.)

6. Use the sample `Makefile` from lecture 2, modified if necessary.

7. Submit this assignment following the submission instructions from lecture 2.

4 Grading Rubric

Your assignment will be graded according to the scale given in Figure 1 (see below).

#	Pts.	Item
1.	1	All source code is contained in the single file `main.cpp`.
2.	1	The source code is an ASCII text file (not a `.doc` file).
3.	1	`main.cpp` contains required authorship information.
4.	1	`main.cpp` `#includes` the `tools` header file.
5.	1	`main()` calls `banner()` and `bye()`.
6.	1	`main()` contains the required `try` and `catch` blocks.
7.	1	The only code inside the `try` block is a call on `run()` (which you must write).
8.	1	Each function is preceded by a comment that describes briefly what it does.
9.	1	Program uses C++ I/O operators `<<` and `>>` with standand streams `cout` and `cin`.
10.	1	Program uses `good()` to check for errors after reading input and throws a `Fatal` exception in case of error.
11.	2	Program correctly computes the current year using `time()` and `localtime()` functions.
12.	1	A well-formed `Makefile` or `makefile` is submitted that specifies compiler options `-O1 -g -Wall -std=c++14`.
13.	1	Running `make` results in an executable file `aboutme`.
14.	1	There are no compiler warnings when compiled with `-Wall`.
15.	1	There is a block comment at the bottom of `main.cpp` that contains the input and actual output from at least one test run of the program.
16.	2	The compiled program, when run on the input data described in the comment at the bottom of `main.cpp`, yields the *same* output as in the comment, and that output is *correct*. (1 point each.)
17.	2	All required files are submitted on classes∗v2 as described in lecture 2.
	20	Total points.

Figure 1: Grading rubric.