CS106B
J Zelenski

Handout #9
Jan 14, 2008

Assignment #1: Getting started in C++

Due: Wed, Jan 23 2:15pm

Lecture has been priming you on how to take your previous programming skills and re-deploy them in C++, and now it's time to get to work on that! Your first assignment consists of a warm-up task (setting up your compiler) and then writing a few simple programs to develop your C++ legs. It's highly recommended you do the warmup as soon as possible. Hey, why not today? That way, if you run into problems, you'll have plenty of time to resolve those issues before tackling the actual assignment.

Warmup: Get your C++ compiler working

Your first task is to set up your C++ compiler. Start with a copy of the appropriate compiler handout ("Using Xcode " for Mac or "Using Visual Studio " for Windows). These handouts walk through the installation steps and have instructions on how to create a new project, edit code, and compile and run a program. Decide what is the most likely computer(s) that you will use and make sure that the compiler and CS106 libraries are properly installed. The public computer clusters on campus should have our software pre-installed. If you plan to work on your own computer, you will need to install the appropriate compiler and CS106 libraries on it.

Once you have the compiler ready, follow the steps in the compiler handout to create a new CS106 C++ project. Add the source file **`warmup.cpp`** that we provide (this file is available from the "Assignments" area of the class web site). Your goal is to get this program running. The source file we give you is a complete C++ program, so complete, in fact, that it comes complete with two bugs. The errors are not difficult to track down (in fact, we'll tell you that one is incorrect arguments to a function call and the other is a needed #include statement is missing). This task is designed to give you a little experience with the way errors are reported by the compiler and what it takes to fix them.

Once you fix the errors, compile and run the program. When the program executes, it will ask for your name. Enter your name and it will print out a "hash code" (a number) generated for that name. We'll talk later in the class about hash codes and what they are used for, but for now just run the program, enter your name, and record the hash code. You'll email us this number. A sample run of the program is shown below (with user input in *italics*):

```
Please enter your name: Julie Zelenski↵
The hash code for your name is 5310.
```

While you're getting familiar, pull out the debugging handout for your compiler and spend a little time learning how the debugger works. Now you are ready for action!

Send us e-mail to announce your compiler success

Being the geeks that we are, your course staff relies heavily on e-mail for our communication and we want to be sure that you are in the e-loop, too. Be sure that your Axess registration is current (that gets you on the class mailing list) and take care to check your mail regularly so you don't miss any important messages from us.

Once you've got your hash code, we want you to email it to your section leader and introduce yourself. You don't yet know your section assignment, but will receive it via email after signups

close, so hold on to your email until then. I'd also appreciate if you would cc me on the e-mail ( ) so I can meet you as well.

Here's the information to include in your e-mail:

1) Your name and the hash code that was generated for it by the program
2) Your year and major
3) When you took 106A (or equivalent course) and how you feel it went for you
4) What you are most looking forward to about 106B
5) What you are least looking forward to about 106B
6) Any information that might be helpful to us about how to best help you learn and master the course material

With the formalities out of the way, include some random and amusing content to make it interesting. Tell us a good joke or what your new quarter resolutions are or what your roommates say when they talk in their sleep or… Share with us what intrigues you about CS106! Tell us something unusual about yourself… Do you play the erhu? Do you own a NeXT machine? Have you been arrested? Do you speak Estonian? Are you double-jointed?

Problem solving in C++

Many of the assignments in this course are single programs of a substantial size. However, this first assignment is a set of small problems to solve in isolation. These problems were adapted from CS106A midterm/final problems and involve only simple A-level concepts (loops, parameters, strings, I/O, arrays, etc.), so you should find the problem-solving very straight-forward. The idea is to become acquainted with expressing code in C++ and working with the language and library features that are almost but not quite the same as those in Java. For each problem, you are to write a short program to solve it. To reinforce good programming habits, you are expected to decompose your solutions, use clean style, and include comments to explain non-obvious parts of the code. Our solution to each program required about a page of code and typically was decomposed into 3-5 functions.

For those of you unsure of whether your programming background is sufficient, this assignment may be helpful as a gauge. You should find the algorithms and problem-solving relatively easy, although you may need to spend time figuring out how to express your intentions correctly in C++. For those of you considering CS106X instead, take a look at their Assignment 1 (available on their course web site) and see how the other half lives. You are welcome to talk to me to figure out what is the right place for you.

1. Variables, arithmetic, and control structures

Greek mathematicians took a special interest in numbers that are equal to the sum of their proper divisors (a proper divisor of *N* is any divisor less than *N* itself). They called such numbers **perfect numbers.** For example, 6 is a perfect number because it is the sum of 1, 2, and 3, which are the integers less than 6 that divide evenly into 6. Similarly, 28 is a perfect number because it is the sum of 1, 2, 4, 7, and 14.

Write a predicate function `IsPerfect` that takes an integer `n` and returns `true` if `n` is perfect, and `false` otherwise. Test your implementation by writing a main program that uses the `IsPerfect` function to check for perfect numbers in the range 1 to 10000 by testing each number in turn. When a perfect number is found, your program should print it. The first two perfect numbers are 6 and 28, there are two others in the range to 10000.

This program will take less than half a page of code. It uses only simple variables, arithmetic expressions and basic control structures.

2. Simulation and use of random/io libraries

A recent New York Times article featured a report on the usability, stability, and effectiveness (or lack thereof) of current electronic voting systems. By one expert's estimate, some voting technologies have a 10% error rate, which means one in ten votes is misrecorded. This sounds alarming, but if the errors are uniformly distributed (i.e. affect each candidate with equal probability), there is some consolation in the fact that it won't alter the final outcome except in very tight races. However, it is interesting to consider what situations could produce invalid results.

Consider a 1000-voter election with a single percentage point spread between two candidates, i.e. 50.5% vote for one candidate, 49.5% for the other. The voting machine makes an error 8% of the time and records a vote for the opposite candidate than intended. Is this error rate high enough to invalidate the results of the election?

With a little knowledge of statistics, it is not hard to calculate the exact probability of an invalid outcome, but it is even easier to simulate this process. Generate a sequence of 505 votes for candidate A and 495 for candidate B where each vote has a 8% chance of being inverted when recorded. Do the vote totals result in B defeating A, despite the original intentions of the voters? This outcome represents one trial in the simulation. If you repeat this trial many times and keep track of the results, the ratio

$$\frac{\text{number of trials in which election result was invalid}}{\text{total number of trials}} \times 100$$

provides an estimate of the percentage chance of an invalid election result.

Write a program that prompts the user to enter the voting simulation parameters, then performs 500 simulation trials and reports the ratio calculated above. A sample run of the program is shown below:

```
Enter number of voters: 10000↵
Enter percentage spread between candidates: .005↵
Enter voting error percentage: .15↵

Chance of an invalid election result after 500 trials = 13.4%
```

Your program should take care to verify the user's chosen simulation parameters are within range (percentages must be 0 to 1.0 and number of voters should be positive) and if necessary, re-prompt for valid input. Note that because of the randomness in the simulation, it is expected that the results will vary from run to run.

This program will give you practice with writing functions, using parameters, and client use of the CS106 simpio and random libraries.

(Given what you know of computer programming, how confidant do you feel about e-voting? Does running this simulation make you feel better or worse?)

References:

Thompson, Clive. "Can You Count On Voting Machines?" *New York Times Magazine*. January 6, 2008.

3. Graphics

For this problem, you will use the graphics library to explore an interesting phenomenon known as the "Chaos Game."

Prompt the user to click three points within the graphics window. We'll refer to these points as A, B, C. Draw a triangle connecting A, B, and C. Now, follow these steps to fill in the triangle with points:

1. Randomly choose one vertex (A, B, or C) as the current point
2. Draw a small filled circle around the current point
3. Randomly choose one vertex (A, B, or C) and move the current point half of the distance toward that vertex
4. Repeat steps 2 & 3 (stop when the user clicks the mouse)

After many iterations, you might expect this random process to produce nothing but an odd mishmash or that the points will eventually fill the entire triangle. Without giving too much away, I'll tell you that the resulting image never varies and will likely surprise you with its beauty and symmetry.

CS106B uses a simple paint-based drawing library that operates on a Cartesian coordinate system where dimensions are expressed in inches. The **`graphics.h`** and **`extgraph.h`** header files give details on the individual functions. For this program, you'll use the functions for handling mouse events (**`WaitForMouseDown`**, **`GetMouseX`**, etc.) and pen-drawing (**`MovePen`**, **`DrawArc`**, **`StartFilledRegion`**, etc). You may also need to call **`UpdateDisplay`** in your loop to flush drawing to the screen without delay.

This program will give you practice with drawing and event-handling via the CS106 graphics library and perhaps even teach you a little bit about fractals!

References:

http://en.wikipedia.org/wiki/Chaos_game Wikipedia page on Chaos Game

4. Strings

One of the more pesky features of the English language is the lack of consistency between phonetics and spelling. Matching surnames is a particularly vexing problem because of the especially wide variation in spelling and the spelling often even changes over time. Classifying names by their phonetic structure is the goal of the *Soundex* system, patented by Margaret O'Dell and Robert C. Russell in 1918. Genealogical researchers, directory assistance, and private investigators are consumers of this neat idea.

The Soundex algorithm produces a code for a surname based on the way it sounds, rather than how it is spelled. The surnames Vaska, Vasque, and Vussky are all assigned the same Soundex code V200 because of their similar pronunciation, despite their differences in spellings. The algorithm is only intended to work with English pronunciation, and there are counter-examples, even in English, where it gets tripped up, but nonetheless this simple algorithm does a surprisingly good job. The US census has extensively used a variation on the Soundex algorithm for almost a century!

All Soundex codes have the same format (an uppercase letter followed by three digits e.g. Z452 or V200). The Soundex algorithm converts a surname to a code using these steps:

1. Keep the first letter of the surname (convert to uppercase if necessary)
2. Convert all other letters in the surname to a digit using the table below (discard any non-letter characters: dashes, spaces, and so on)

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	M N
5	L
6	R

3. Remove any consecutive duplicate digits (e.g. A122235 becomes A1235)
4. Remove any zeros
5. Make resulting code exactly length 4 by truncating or padding with zeros

Write a program that allows the user to enter a name and prints its Soundex code as determined by the above algorithm. It should operate in a loop that allows the user to get codes for as many names as desired before exiting. A sample run of the program is shown below:

```
Enter surname (RETURN to quit): Vaska↵
Soundex code for Vaska is V200
Enter surname (RETURN to quit): Zelenski↵
Soundex code for Zelenski is Z452
Enter surname (RETURN to quit): ↵
```

This program will give you practice working with C++ strings and characters and a little insight into phonetics. How good a job does Soundex do matching your surname to other spellings?

References:

Knuth, Donald. The Art of Computer Programming, Volume 3: Sorting and Searching. Addison-Wesley, 1972, pp. 391-392.

5. File processing and vectors

You are to write a program to produce a histogram from a file of exam scores. You first ask the user for the name of the data file containing the score information. The data file contains a list of scores, one per line. Your function should read the scores from the file, tally them into buckets, and print a histogram. If the data file contained these 12 scores:

```
76
89
95
83
80
71
54
82
59
92
41
72
```

This histogram is printed:

```
0-9:
10-19:
20-29:
30-39:
40-49: X
50-59: XX
60-69:
70-79: XXX
80-89: XXXX
90-99: XX
```

A few specifications:

- Each score is between 0 and 99 and you are dividing the scores into buckets by the first digit (i.e. 0-9, 10-19, 20-29 and so on).
- If the named file can't be opened, you should re-prompt for another name until you succeed. You may assume a file that opens is well-formed, i.e. each line contains a single valid score.
- You should use a Vector object to store the histogram information.

This program will give you practice with using file streams and the CS106B Vector template class.

Getting started

The class web site http://see.stanford.edu/see/materials/icspacs106b/assignments.aspx
For Assign 1, you will need the source file **`warmup.cpp`**, which contains the buggy hash code program.

For the assigned problems, create a new project and add a new file in which to write your solution. Since you have several different programs to write, you might choose to create separate projects, one for each program. Alternatively, you could re-use the same project as you work your way through the problems, but note that a project cannot simultaneously contain two files that both declare the **`main`** function, so you will need to either remove the previous file (ie remove part1.cpp from your project and then add part2.cpp) or temporarily rename the unused **`main`** function(s).

Deliverables

The deliverables for this assignment are your .cpp source files. You should hand in separate programs for each problem, respectively named part1.cpp, part2.cpp, and so on.

You are required to submit both a printed version of your source, as well as an electronic version via ftp. Both are due before the **beginning of lecture** on the due date. See the electronic submission handout for details on how to submit the electronic version. The printed version should be handed in before lecture starts. Be sure the pages are firmly stapled together and that the printout is clearly marked with your name, CS106B and your section leader's name! SCPD students need only submit electronically, no paper copy needed.

You are also responsible for keeping a copy of your assignment to ensure that there is a backup in case your submission is lost or the electronic submission is corrupted. Never turn in your only copy!

"I can't think of a job I'd rather do than computer programming. All day, you create patterns and structure out of the formless void, and you solve dozens of smaller puzzles along the way."
—Peter Van Der Linden, "Expert C Programming: Deep C Secrets"