

Project #01: iClicker Analysis

Complete By: Thursday, September 14th @ 11:59pm

Assignment: C++ program to perform clicker data analysis

Policy: Individual work only, late work **is** accepted (see “Policy” section on last page for more details)

Submission: online via repl.it (exercise P01)

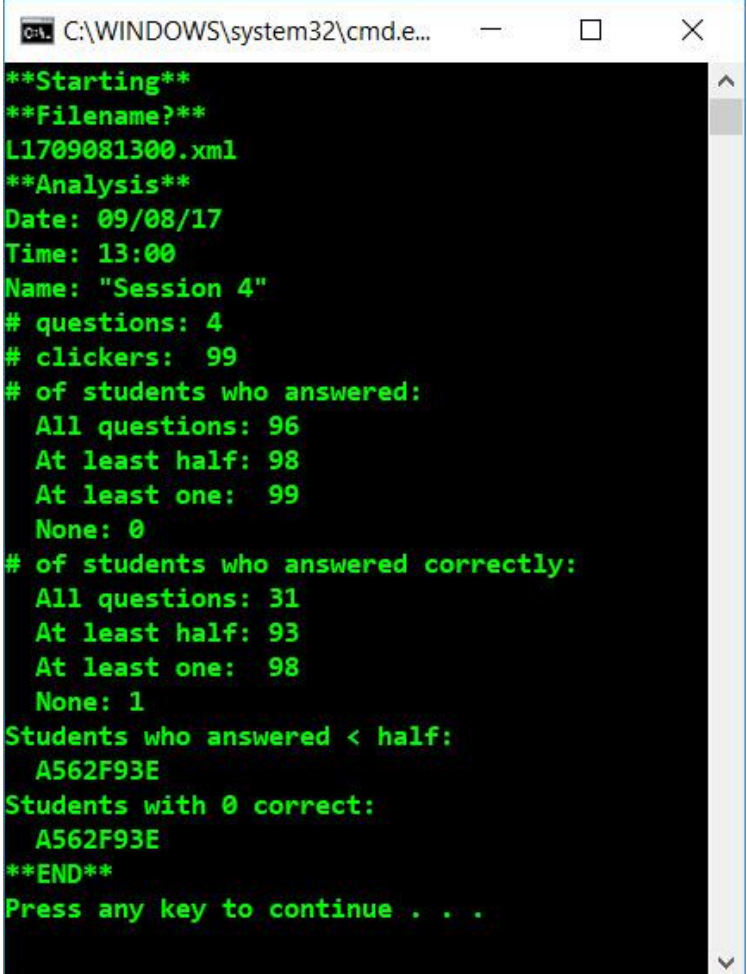
Assignment

The assignment is to input iClicker data for one class session, and then analyze this data to see how many questions were given, how many students answered, how many students answered correctly, and so on. The input file is a text file in XML format.

Here’s a screenshot for the input file “L1709081300.xml”, which can be downloaded from the course web page. The user inputs the filename, and then the program opens the file, reads the data, performs the analysis, and outputs the results. Your program must match this output exactly since it will be auto-graded using the repl.it system.

Notice that the filename is where you will obtain the date and time. For example, the file shown here --- “L1709081300.xml” --- denotes the year 17, the month 09, the day 08, the hour 13, and the minute 00.

Since most of you are relatively new to C++, approach this like a CS 1 assignment: one baby step at a time. Do not try to solve this assignment all at once.



```
C:\WINDOWS\system32\cmd.e...
**Starting**
**Filename?**
L1709081300.xml
**Analysis**
Date: 09/08/17
Time: 13:00
Name: "Session 4"
# questions: 4
# clickers: 99
# of students who answered:
  All questions: 96
  At least half: 98
  At least one: 99
  None: 0
# of students who answered correctly:
  All questions: 31
  At least half: 93
  At least one: 98
  None: 1
Students who answered < half:
  A562F93E
Students with 0 correct:
  A562F93E
**END**
Press any key to continue . . .
```

Input Files

The program takes a single input file. The input file is a text file in XML format. You'll need to open the file using a text editor, or a programming environment (such as Visual Studio) and learn your way around the format. Here's a short overview of the file "L1709081300.xml":

```
<!--***** Naming Convention *****-->
<!--qn : Question Name-->
<!--quid : Question Universally Unique Identifier-->
.
.
.
<ssn cdt="19" perf="-1" ... ssn="Session 4" ...>
  <p resp_c="19|0.00" isDel="N" isap="N" ... cans="B" ...>
    <v att="4" scr="0.00" fans="A" ans="B" .../>
    <v att="1" scr="0.00" fans="B" ans="B" .../>
    <v att="3" scr="0.00" fans="A" ans="A" .../>
    .
    .
    .
  </p>
  .
  .
  .
</ssn>
```

XML is like HTML in that it is a tag-based language. The file denotes a single session, which begins with the tag `<ssn ...>` and ends with the tag `</ssn>`. The `<ssn ...>` tag is where you'll find the session name attribute `ssn="..."`.

A session may contain 0 or more questions. Each question begins with the `<p ...>` tag and ends with `</p>`. The correct answer is denoted by the attribute `cans="..."`. The correct answer can be a single letter such as "B", or multiple letters each separated by |, e.g. "A|B".

For each question, the answer by each student is given by the `<v .../>` tag. The student's final answer is contained in the attribute `ans="..."` --- if this string is empty then the student did not answer the question. The `id` attribute is the clicker ID, and how we identify the student.

At least two sample input files are available on the course web page under "Projects", then "[project01-files](#)". The files are being made available for each platform (Linux, Mac, and PC) so you may work on your platform of choice. **NOTE:** the best way to download the files is **not** to click on them directly in dropbox, but instead use the "download" button in the upper-right corner of the dropbox web page.

Requirements

As is the case with all assignments in CS 341, how you solve the problem is just as important as simply getting the correct answers. You are required to solve the problem the “proper” way, which in this case means using modern C++ techniques: classes, objects, built-in generic algorithms and data structures, lambda expressions, etc. It’s too easy to fall back on existing habits to just “get the assignment done.”

In this assignment, your solution is required to do the following:

- Use **ifstream** to open / close the input file; see [lecture #01](#) for an example of using ifstream, and then **getline(file, line)** to read lines from the file.
- When parsing the input file, use only **standard C++**. There is no XML parser in standard C++, so my recommendation is to use the **find()** and **substr()** functions in the **std::string** class. In other words, use **find** to search for the attribute, and then **substr** to extract the value.
- Use one or more classes to store the input — at a minimum you must have a **Student** class that contains data about each student: their clicker ID, the # of questions answered, and the # of questions answered correctly. The design of that class is up to you, but at the very least (1) all data members must be private, (2) the class contains a constructor to initialize the data members, and (3) getter/setter functions are used to access/update the data members.
- Use **std::vector<T>** to store your objects (yes, there are other containers, but in this assignment you must use **std::vector<T>**).
- When searching the vector, use the built-in linear search function **std::find_if**. No looping and searching yourself --- use **std::find_if**.
- When counting the # of students who answered questions, use **std::count_if**. No explicit looping and counting yourself --- use **std::count_if**.
- In fact, the only loops you should have are (1) to input the data, and (2) to output the clicker ids of students who either didn’t answer at least half, or didn’t answer any correctly. All other iteration should be performed by library functions.
- No explicit pointers of any kind — no **char ***, no **NULL**, no C-style pointers
- No global variables whatsoever; use functions and parameter passing.

Writing code that is not modern C++ will result in a score of 0. Using a container other than **std::vector**? 0. No classes? 0. Using global variables? 0. Using a 3rd-party library such as Boost? 0.

For online documentation, I typically use the site <http://www.cplusplus.com/reference/>. Another way is to google with the prefix **std::**, which refers to the C++ standard library. Example: for information about the C++ string class, google “**std::string**”. For the vector class, google “**std::vector**”.

Assignment Details

A student answer that is blank --- i.e. the empty string "" --- means he/she did not answer the question. If there are N questions, then a student must answer N/2 or more to have answered at least half of them. If N is even, e.g. 4, this means a student must answer 2 or more. If N is odd, e.g. 5, this means a student must

answer 3 or more. Assuming N is an integer, an easy way to handle both cases is to check to see if the # of answers is $\geq ((N+1)/2)$.

We've talked about `std::count_if`, but not `std::find_if`. The two are similar in that they both take a predicate function for the lambda expression, but `std::find_if` returns an *iterator* if it finds what you are looking for. Here's an example. Suppose we want to search a vector of students for the first student named "Venky":

```
std::string name = "Venky";

auto result = std::find_if(students.begin(), students.end(),
    [=](const Student& s)
    {
        if (s.getName() == name) // found what we are looking for, return true!
            return true;
        else // not a match, return false so alg keeps searching:
            return false;
    }
);
```

Note that the lambda expression takes a *single* element of the container — a student in this case — and returns true if this student is a match and false if not. The `std::find_if` function calls the lambda expression with different elements of the container, until either a match is found or all elements have been checked.

How do we know if the search was successful? If a match is found, `std::find_if` returns an *iterator* that "points" to the first object that successfully matched the search. You check by comparing `std::find_if`'s return value to the end of the container:

```
if (result == students.end()) // search did *not* find matching student:
    cout << "Student not found" << endl;
else
    cout << result->getName() << ": avg=" << result->getExamAvg() << endl;
```

Since **result** is an iterator, and iterators are like pointers, the `->` operator must be used to call the underlying functions in the Student object.

Programming Environment

To facilitate testing, we'll be using the repl.it system for submission and grading; see "**P01 iClicker Analysis**". You can work entirely within repl.it, or you can download the input files (and starter C++ files) from the course web page and work in the C++ environment of your choice; Visual Studio is a good choice, but is not required for this assignment. Any environment with a modern (C++11 or newer) compiler will suffice; note that **bert** and **ernie** do not support modern C++, though **bertvm** does.

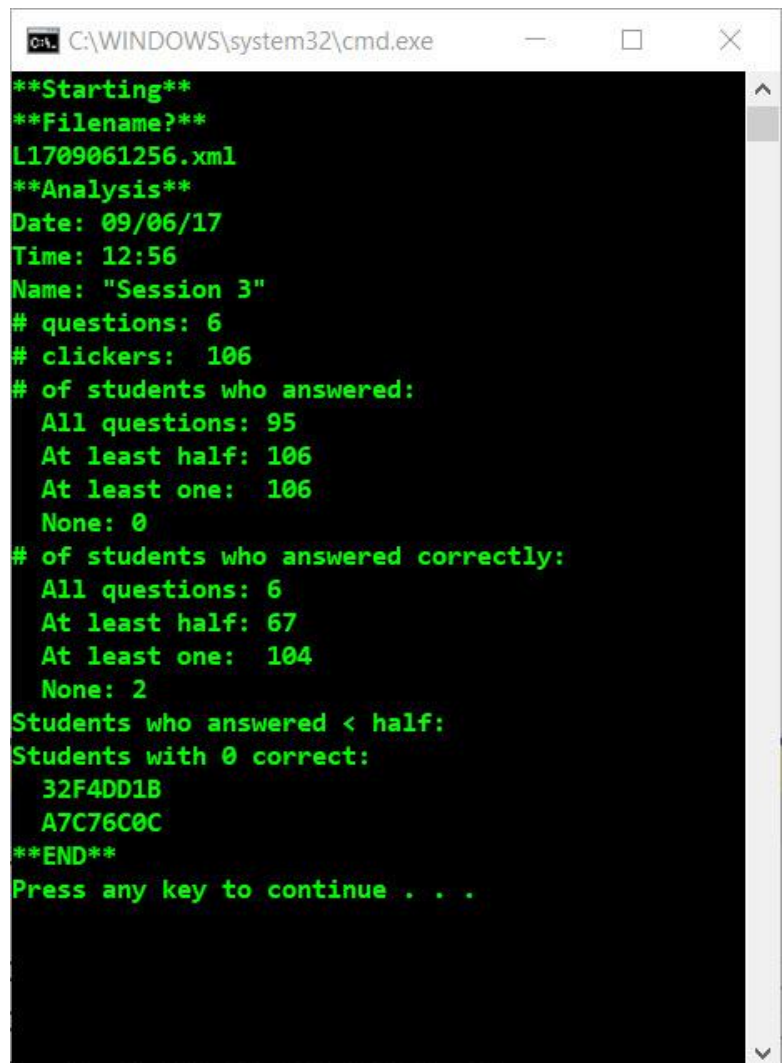
Have a question? Use Piazza, not email

Do not email the staff with questions — use [Piazza](#). Remember the guidelines for using Piazza:

1. Look before you post — the main advantage of Piazza is that common questions are already answered, so search for an existing answer before you post a question. Posts are categorized to help you search, e.g. “Pre-class” or “HW”.
2. Post publicly — only post privately when asked by the staff, or when it’s absolutely necessary (e.g. the question is of a personal nature). Private posts defeat the purpose of piazza, which is answering questions to the benefit of everyone.
3. Ask pointed questions — do not post a big chunk of code and then ask “help, please fix this”. Staff and other students are willing to help, but we aren’t going to type in that chunk of code to find the error. You need to narrow down the problem, and ask a pointed question, e.g. “on the 3rd line I get this error, I don’t understand what that means...”.
4. Post a screenshot — sometimes a picture captures the essence of your question better than text. Piazza allows the posting of images, so don’t hesitate to take a screenshot and post; see <http://www.take-a-screenshot.org/>.
5. Don’t post your entire answer — if you do, you just gave away the answer to the ENTIRE CLASS. Sometimes you will need to post code when asking a question --- in that case post only the fragment that denotes your question, and omit whatever details you can. If you must post the entire code, then do so privately --- there’s an option to create a private post (“visible to staff only”).

Another Sample Input file

As another data point, here’s the output for the provided input file “L1709061256.xml”.



```
C:\WINDOWS\system32\cmd.exe

**Starting**
**Filename?**
L1709061256.xml
**Analysis**
Date: 09/06/17
Time: 12:56
Name: "Session 3"
# questions: 6
# clickers: 106
# of students who answered:
  All questions: 95
  At least half: 106
  At least one: 106
  None: 0
# of students who answered correctly:
  All questions: 6
  At least half: 67
  At least one: 104
  None: 2
Students who answered < half:
Students with 0 correct:
  32F4DD1B
  A7C76C0C
**END**
Press any key to continue . . .
```

Submission

The program is to be submitted via <http://repl.it>: see the exercise “**P01 iClicker Analysis**”. The grading scheme at this point is 90% correctness, 10% style — we expect basic commenting, indentation, and readability. You should also be using functions, parameter passing, and good naming conventions.

Policy

Late work *is* accepted. You may submit as late as 24 hours after the deadline for a penalty of 25%. After 24 hours, no submissions will be accepted.

All work is to be done individually — group work is not allowed. While I encourage you to talk to your peers and learn from them (e.g. via Piazza), this interaction must be superficial with regards to all work submitted for grading. This means you *cannot* work in teams, you cannot work side-by-side, you cannot submit someone else’s work (partial or complete) as your own. The University’s policy is described here:

<http://www.uic.edu/depts/dos/docs/Student%20Disciplinary%20Policy.pdf> .

In particular, note that you are guilty of academic dishonesty if you extend or receive any kind of unauthorized assistance. Absolutely no transfer of program code between students is permitted (paper or electronic), and you may not solicit code from family, friends, or online forums. Other examples of academic dishonesty include emailing your program to another student, copying-pasting code from the internet, working in a group on a homework assignment, and allowing a tutor, TA, or another individual to write an answer for you. Academic dishonesty is unacceptable, and penalties range from failure to expulsion from the university; cases are handled via the official student conduct process described at <http://www.uic.edu/depts/dos/studentconductprocess.shtml> .