Michigan Tech

UN5390: Scientific Computing I

Fall 2016

Assignment #01 (Due 11:59 am on Sunday, 11th Sep 2016; 5%)

With recent feedback from students, it is likely that the Training Camps contain new or revised information – e.g., `${HOME}/.bash_${USER}`, `${HOME}/bin/functions.sh`, functions, scripts, rationale, and index. Review the material, and update your notes, if necessary.

Problem 1

[50 points] The US Constitution requires a census every decade. With less than four million people to tally, it was a manageable task in 1790 – the first year of census. Following the one in 1880 conducted at a cost of approximately five million dollars, the population had increased to upwards of 50 million and publication of reports wasn't completed until later parts of 1888. As if that weren't enough, preliminary/projected estimates indicated that 1890 census would not be completed before the 1900 census had already started [1].

So, the Census Bureau held a competition in 1888 to find a more efficient method to process and tabulate data. Contestants were asked to process 1880 census data from four areas in St. Louis, Missouri. Whoever captured and processed the data fastest (and correctly) would win a contract for the 1890 census.

Three contestants accepted the Census Bureau's challenge. The first two contestants captured the data in 144.5 hours and 100.5 hours respectively. The third contestant completed the same process in just 72.5 hours. Next, the contestants had to prove that their designs could prepare data for tabulation by age category, race, gender, and so on. First two contestants required 44.5 hours and 55.5 hours respectively. The third contestant astounded the agency officials by completing the task in just 5.5 hours. These impressive results earned the third contestant, Herman Hollerith, the contract to process and tabulate 1890 census data.

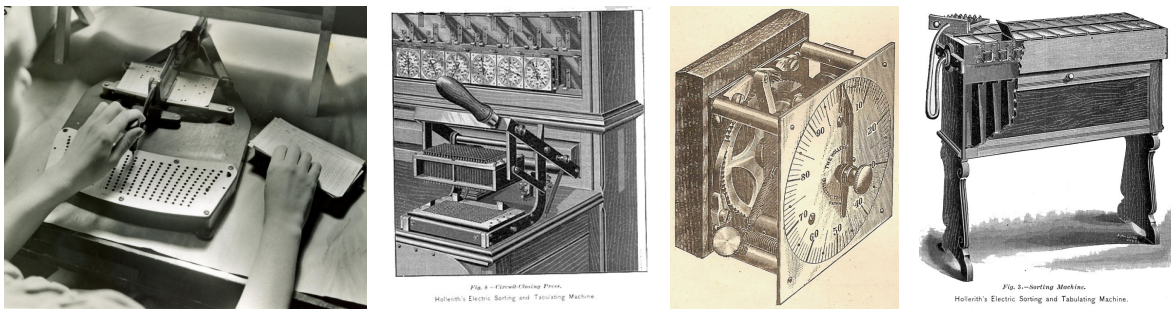
His tabulator consisted of the following electrically-operated components that captured and processed census data by *reading* holes on paper punch cards [2]:

1. Pantograph - Used by the Census Bureau clerks to transfer the census information from schedules to paper punch cards. Each card was *punched* to represent specific data on the census schedule. Clerks could prepare about 500 cards per day.

   The punched cards (and the reading board as well) had one corner cut diagonally to protect against upside down and/or backwards cards that might not otherwise be detected. They measured $3.25 \times 7.375\ in$, the same size as the 1887 US paper currency because Hollerith wished to use Treasure Department containers as card carrying boxes.

2. Card Reader - The manually-operated card reader consisted of two hinged plates operated by a lever (similar to a waffle iron). Clerks opened the reader and positioned a punched card between the plates. Upon closing the plates, spring-loaded metal pins in the upper plate passed through the punched data holes in the cards, through the bottom plate, and into wells of mercury beneath.

   Pins that passed through the punch card completed an electrical circuit when contacting the mercury below. The completed circuit energized the magnetic dials on the tabulator and advanced the counting hands. Upon completion of the electrical circuit (signaled by the ringing of a bell), the clerk transcribed the data indicated by the dial hands.



(a) Pantograph **(b)** Card Reader **(c)** Dials **(d)** Sorting Table

Figure 1: Components of Herman Hollerith Tabulator

3. Tabulator Dials - Tabulators consisted of 40 data-recording dials. Each dial represented a different data item collected during the census. The electrical impulses received as the reader's pins passed through the card into the mercury advanced the hands on the dials corresponding to the data contained on the punch card (i.e., responses to inquiries about race, gender, citizenship, age, etc). When the bell signaled the card had been read, the operator recorded the data on the dials, opened the card reader, removed the punch cards, and reset the dials.

4. Sorting Table - After registering the punch card data on the dials, the sorter specified which drawer the operator should place the card. The clerk opened the reader, placed the punch card in the designated sorter drawer, reset the dials, and positioned a new card to repeat the process. An experienced tabulator clerk could process 80 punch cards per minute.

Using his technique, with 30 questions per person, the census of 1890 for nearly 63 million people was completed in just about one year (and only about six weeks to process and tabulate the data) with a total operational cost of approximately 11 million dollars.

Herman Hollerith was a once short term Mechanical Engineering faculty in MIT, and a former Census Bureau employee in Baltimore, Maryland. He earned a PhD from Columbia University in 1890 and founded *Tabulating Machine Company*. The latter merged with three other corporations to form *Computing Tabulating Recording Company* in 1911 and was renamed as *International Business Machines* (IBM) in 1924.

Modified versions of his technology would continue to be used at the Census Bureau until replaced by computers in the 1950s. His invention of the punched card evaluating machine marks the beginning of the era of automatic data processing systems, and his concept dominated the computing landscape for nearly a century. Herman Hollerith is long since considered the architect of modern machine data processing.

How would you have done what he did? Are there ways to improve his technique? Is there a problem, in our current world, that requires a (radical) new approach to solve? Is there anything else noteworthy about the 1890 census? Use schematics to visually aid your thought process if necessary but keep your answer concise and under one single-sided page.

Problem 2

[50 points] National Oceanic and Atmospheric Administration (NOAA) [3] announced in January 2015 a plan to increase the capacity of each of its two operational supercomputers to 2.5 PFLOPS by October 2015 – nearly a 10-fold increase from the then current capacity. NOAA completed these supercomputer upgrades in January 2016. Search for and read up on these press releases, and write a two-page summary addressing the following question:

What can NOAA do with the increased computing power that they could not do before?

Problem 3

[50 points] The Collaboration of Oak Ridge [4], Argonne [5], and Livermore [6] – CORAL – is a joint procurement activity among three of the Department of Energy's [7] national laboratories launched in 2014 to build state of the art computing technologies.

Search for and read up on press releases and news articles, and write a two-page summary on CORAL. Include information about sites, source and amount of funding, vendors, hardware configuration, power and cooling specifications, performance, and the class of intendend audience as well as potential applications.

Problem 4

[50 points] Understand the programming language of your choice. To that effect, write a brief description – not exceeding four pages – of the language's features, naming conventions, data types and their size limitations, reserved names, quirks, etc.

Problem 5

[50 points] Write down the steps you would follow – including logical simplifications, mathematical substitutions and cautionary checks, if any – when computing the value of π_c using the given approximation, and the error in such an approximation (ϵ).

$$\frac{1}{\pi_c} = \frac{2\sqrt{2}}{9801} \sum_{n=0}^{\infty} \frac{(4n)!\ (1103 + 26390n)}{(n!)^4\ 396^{4n}} \qquad \epsilon = |\pi - \pi_c|$$

The answer needs to include historical and/or computational significance of this approximation, if any. Translate these steps into a computer program, `pi_sr.EXTN`, to compute π_c to 15 decimal places. Estimate the associated error, ϵ, using 3.141592653589793 as the known value of π. Is your program in compliance with language's features and guidelines? Explain why it could be a bad idea to name this program as just `pi.EXTN`.

`EXTN` indicates the language-specific file name extension. For e.g., `pi_sr.c` for C, `pi_sr.cpp` for C++, `pi_sr.f90` for FORTRAN90, `pi_sr.java` for Java, `pi_sr.jl` for Julia, `pi_sr.nb` for Mathematica, `pi_sr.m` for MATLAB, `pi_sr.py` for Python, and so on.

Problem 6

[**50 points**] Write a well-commented BASH script with meaningful error messages and exit codes, `seconds2hms.sh`, that converts the given number of seconds into human readable format, `h:mm:ss`. Its usage should be as follows.

```
seconds2hms.sh SECONDS
```

where `SECONDS` is the number of seconds supplied by the user.

1. Test the script for following values of `SECONDS`: 1, 61, 3661 and 90061. Output should be `0:00:01`, `0:01:01`, `1:01:01` and `25:01:01`
2. Display a help message and quit if the script is not called with exactly one argument
3. Ensure that `SECONDS` is an integer. Display an error message and quit if it is not

Problem 7

[**Optional**] Write a well-commented BASH script with meaningful error messages and exit codes, `julian2ymd.sh`, that converts a given Julian date in a specific year into `YYYY-MM-DD` format. Its usage should be as follows.

```
julian2ymd.sh YEAR JULIAN_DATE
```

where `YEAR` and `JULIAN_DATE` are the year and Julian date supplied by the user.

1. Test the script for following combinations of `YEAR` and `JULIAN_DATE`: (2015, 263), (2016, 60), (2017, 60) Output should be `2015-09-20`, `2016-02-29`, `2017-03-01`
2. Display a help message and quit if the script is not called with exactly two arguments OR if `YEAR` and `JULIAN_DATE` are not integers
3. List any other input validation the script might need. Implement it/them if you can

Problem 8

[Optional] Write a well-commented BASH script with meaningful error messages and exit codes, `git_${USER}.sh`, that uses the a given string as a commit message and commits changes to the remote repository. Its usage needs to be as follows.

```
git_${USER}.sh "WEEK" "COMMIT_MESSAGE" "[PUSH]"
```

`WEEK` is the week number, `COMMIT_MESSAGE` is the commit message to accompany the changes, and `PUSH` is an optional argument. `git_${USER}.sh` needs to perform the following workflow.

1. Ensure the local working directory is in a Git [8] repository. Exit otherwise
2. Pull down the latest changes from the remote repository (i.e., GitHub [9])
3. Change into `${UN5390}/CourseWork/Week_${WEEK}/${USER}_${WEEK}`
4. Make an array of files larger than 2 MB and symbolic links to exclude from the repository. Stage and commit all, excluding the entities in the aforementioned array, tracked entities in `${USER}_${WEEK}` via `git commit -am "${COMMIT_MESSAGE}"`
5. If the `PUSH` argument is present, then push to the remote repository (i.e., GitHub) via `git push origin master`

Problem 9

[Optional] Write a well-commented BASH script with meaningful error messages and exit codes, `git_tag.sh`. It should detect the week number by analyzing the full path to the present working directory, tag the assignment, and push the tags to the repository. It should not accept any arguments.

Problem 10

[Optional] Write a well-commented BASH script with meaningful error messages and exit codes, `git_purge.sh`, that purges a given file (or a folder) completely from the repository's history. Its usage needs to be as follows.

```
git_purge.sh FILENAME
git_purge.sh FOLDER
```

References

[1] US Census Bureau.
`https://www.census.gov/`.

[2] Computer History Museum.
1401 N. Shoreline Blvd., Mountain View, CA 94043
`http://www.computerhistory.org/`.

[3] National Oceanic and Atmospheric Administration.
`http://www.noaa.gov/`.

[4] Oak Ridge National Laboratory.
`http://www.ornl.gov/`.

[5] Argonne National Laboratory.
`http://www.anl.gov/`.

[6] Lawrence Livermore National Laboratory.
`http://www.llnl.gov/`.

[7] US Department of Energy.
`http://www.energy.gov/`.

[8] Git.
`https://git-scm.com/` | `https://git-scm.com/doc/`.

[9] GitHub.
`https://github.com/` | `https://education.github.com/`.

Guidelines

1. Every problem in this assignment has a value beyond its numerical score. It is either a *swan kick* moment for what you already know OR a *wax on, wax off* training for something you will learn soon OR a combination thereof. Searching the literature to learn more about the underlying concepts and treating every problem as a tiny research project is very highly encouraged

2. Do not seek writing, scripting and/or programming help from your classmates or anyone else. This assignment is intended to demonstrate whether or not you have completed the training camp material made available during Summer. It is also intended to showcase your own writing, scripting and programming styles, and help you learn how to get help when working by yourself

3. Read through/Review the course material, and additional references, if necessary. Give yourself enough time to complete the assignment on or ahead of time. Checked the assignment for spelling and grammatical correctness. Include suggested corrections from the instructor, if any, and show continuous integration/improvement. Contact the instructor if you didn't understand the course material and/or the assignment

4. Typeset your solutions/answers in `${USER}_01.tex`, `${USER}_01.pdf`, and other files as indicated throughout this assignment. Include images, etc. to explain your answers, when necessary. There is no need to explicitly include the code in the `${USER}_01.tex` document. There needs to be at least one commit per problem to the GitHub repository to show timely work, demonstrate your work ethic, and a final tagged version of the assignment for submission to earn any credit

5. Undergraduate and non-traditional students can opt between problems #2 and #3

   Problems #7 – #10 represent an opportunity to *do a little more*. Each is worth 0.50%

   Deadline for seeking help from the instructor: 7:59 am on Friday, 9th Sep 2016

Assignment Submission Workflow

1. Replace `#1` with the appropriate problem number. Keep all necessary files and folders under

   `${UN5390}/CourseWork/Week_01/${USER}_01/`

2. Before starting to work on a problem (or part thereof)

   ```
   cd ${UN5390}
   git pull
   ```

   Read through `${UN5390}/CourseWork/Week_01/suggested_corrections_01.txt` for suggestions to correct/improve your answers, if any

3. After each problem (or a part thereof), commit to the repository

   ```
   cd ${UN5390}/LaTeXTemplates/Course/
   git pull
   git add ${USER}.bib
   cd ../../CourseWork/Week_01/
   git add ${USER}_01
   git commit -m "Submitting problem #1 in assignment #01"
   git push origin master
   ```

4. When the assignment in its entirety is ready for final submission

   ```
   cd ${UN5390}/LaTeXTemplates/Course/
   git pull
   git add ${USER}.bib
   cd ../../CourseWork/Week_01/
   git add ${USER}_01
   git tag -a a01 -m "Submitting assignment #01"
   git push origin a01
   ```

5. If you made edits to your submission after tagging but before the final due date

   ```
   cd ${UN5390}/LaTeXTemplates/Course/
   git pull
   git add ${USER}.bib
   cd ../../CourseWork/Week_01/
   git add ${USER}_01
   git tag -a a01.1 -m "Submitting assignment #01"
   git push origin a01.1
   ```

   and so on