Lab 2 - An Apple a Day

Lab 2 due on January 29th, 2023 at 11:59PM.

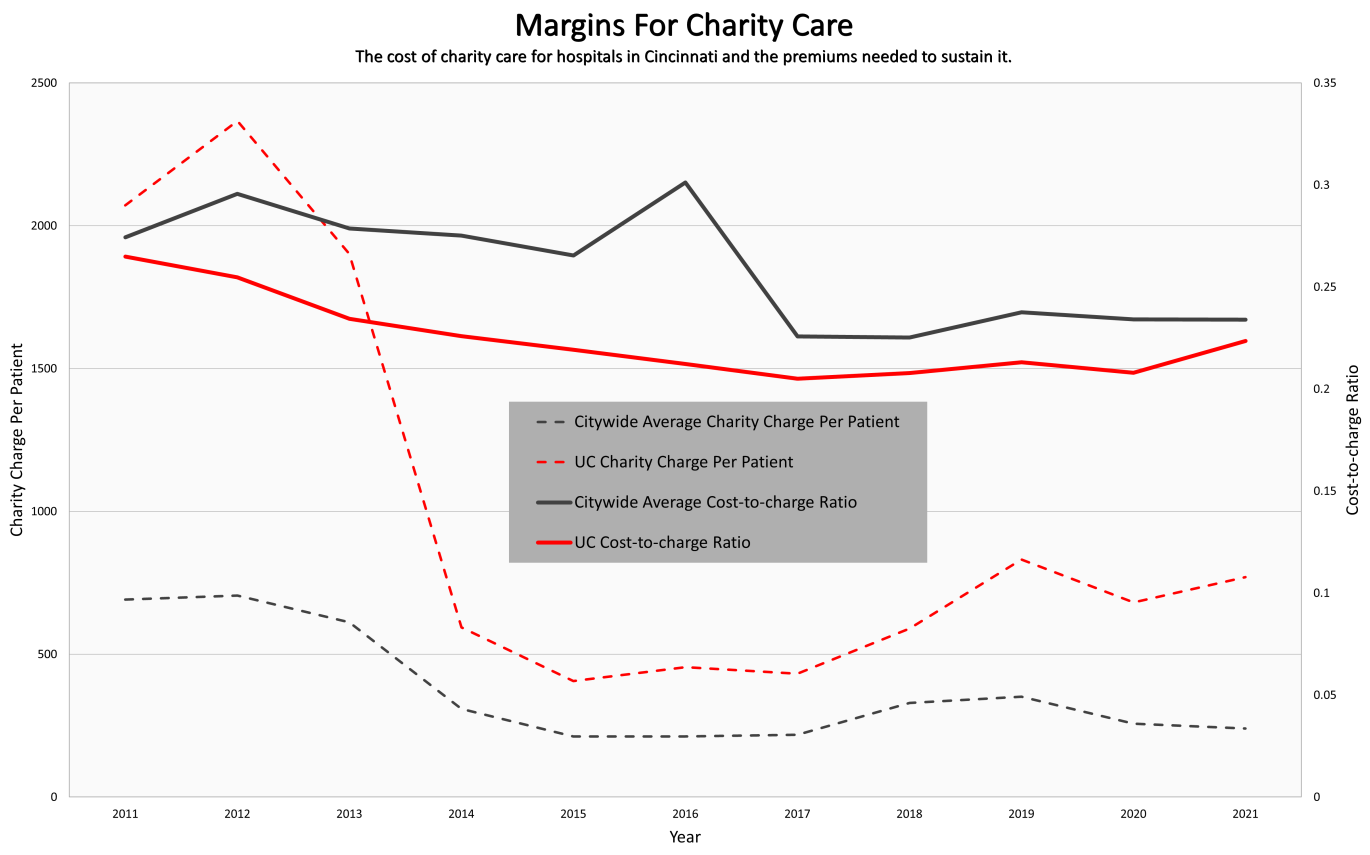
Introduction

Modern medicine is an amazing thing. In the span of approximately 50 years, we have gone from **never having done a single organ transplantation (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8682823/)** to doing over **40,000 *a year* (https://unos.org/news/2021-all-time-records-organ-transplants-deceased-donor-donation/)**! Heck, we didn't even realize that germs were a *thing* until after the Civil War and only in **1928 did we create the first antibiotic (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5403050/)**. Just a few years later, **right here at the University of Cincinnati (https://med.uc.edu/about/history)** we had the development of the first live vaccine for Polio, the first training program for doctors specializing in emergency medicine and the first heart-lung machine.

That success does not come cheap, however. Every year the National Health Expenditure Accounts provide the official "estimates" for spending on health care in the county. **"U.S. health care spending grew 4.1 percent in 2022, reaching $4.5 trillion or $13,493 per person. As a share of the nation's Gross Domestic Product, health spending accounted for 17.3 percent." (https://www.cms.gov/data-research/statistics-trends-and-reports/national-health-expenditure-data/historical)** By 2031, the Centers for Medicare & Medicaid Services **forecasts (https://www.cms.gov/files/document/nhe-projections-forecast-summary.pdf)** that spending on healthcare will account for 19.6% of the nation's Gross Domestic Product. Many of us get lots of care through hospitals (and their associated clinics) and the way that these entities do accounting is, well, interesting. Hospitals do not really know how much each procedure actually costs. Instead they use a special number called the cost-to-charge ratio to "back" calculate the costs based on what they charge. (If you were to ask, "How do they know how much to charge, then?", I would just shrug.) The cost-to-charge ratio is, well, just that: the total costs of the medical provider divided by the total charges that they issue to their patients. It is tempting to think of this ratio as an approximation of a providers' profit margin and I don't think that is necessarily incorrect.

> If you do think about it as something akin to a profit margin, be careful: the ratio is actually upside down. The lower the number the more a provider charges relative to what they spend.

Not all of the people that hospitals treat are able to pay their bills. Fortunately, hospitals have an obligation to treat everyone. Each hospital records the fees that they charge to patients that are ultimately deemed to be charity. Dividing the cost of charity care by the number of patients a hospital system sees each year gives us the costs incurred for charity treatment per patient. It seems reasonable that hospitals that spend more to treat patients who cannot pay (i.e., those that have a higher cost-of-charity-per-patient-seen ratio) would need to have a lower cost-to-charge ratio (i.e., they would need to charge more relative to their costs) to support all the work that they do without remuneration. The graph below shows the statistics for hospitals in Cincinnati with special focus on the University of Cincinnati Medical System.



The data seem to confirm this hypothesis for UC: They spend a higher amount per patient to support their care for people who cannot pay than the regional average and they have a lower cost-to-charge ratio than the regional average.

In this lab you are going to take the Cincinnati-region's average charity-charges-per-patient to do some calculations that will help local hospitals compare how they are doing relative to regional norms.

Ledgeon, your hospital accounting calculator, will take four (4) inputs:

1. a year,
2. the number of patients seen by the hospital in that year,
3. a second year, and
4. the number of patients seen by the hospital in that year

and will

1. **calculate the increase/decrease** (https://www.mathsisfun.com/numbers/percentage-change.html) in patients seen between the first and second years,
2. calculate the total charity charges issued by the hospital each year, and
3. print the results of those calculations on the console (Note: Do ***not*** format your outputs using output manipulators -- the default output format is expected!).

Please be sure to read the *entire* lab document before beginning!

Good luck and have fun!

Program Design Task

As my dad always says, "Measure once and make the laproscopic incision twice." *Before* you start writing code, please write the pseudocode for your implementation of the *Ledgeon* application. In particular, **read about calculating the percent increase/decrease (https://www.mathsisfun.com/numbers/percentage-change.html)** and formulate a plan for how you will implement that calculation in your program.

Program Design Requirements

Your pseudocode or flow chart must describe the entirety of the solution. You may choose to write the flow chart or the pseudocode at any level of detail but, remember, this is a tool for you! Your pseudocode or flow chart *must* include a description of how you plan

1. to calculate the increase/decrease in patients seen in the two years,
2. to calculate the total charity charges per year for each of the two years, and
3. to seek input from the user and present output to the console.

Programming Task

If you are a Windows developer, begin your solution with **this skeleton (https://uc.instructure.com/courses/1657742/files/176197251?wrap=1)** **(https://uc.instructure.com/courses/1657742/files/176197251/download?download_frd=1)** . If you are a macOS developer, begin your solution with **this skeleton (https://uc.instructure.com/courses/1657742/files/176244901?wrap=1)** **(https://uc.instructure.com/courses/1657742/files/176244901/download?download_frd=1)** . If you are a Linux developer, begin your solution with **this skeleton (https://uc.instructure.com/courses/1657742/files/176197319?wrap=1)** **(https://uc.instructure.com/courses/1657742/files/176197319/download?download_frd=1)** ! Your programming task is to write *Ledgeon* to prompt the user for four (4) different inputs

1. a year
2. the number of patients seen that year
3. a year
4. the number of patients seen that year

(in that order), calculate

1. the percent increase/decrease in patients seen between the two years
2. the total charity charges in each of those years

and output the results of the calculations in the following format:

```
In year <first year entered>, our hospital issued <result of calculation> dollars of charity
charges.
In year <second year entered>, our hospital issued <result of calculation> dollars of charity
charges.
```

```
Between <first year entered> and <second year entered>, there was a <result of calculation>%
increase in patients seen at our hospital.
```

Your program should seek user input with the following prompts:

```
Enter the first year:
Enter the number of patients we saw that year:
Enter the second year:
Enter the number of patients we saw that year:
```

You may assume

1. The first year that the user enters will precede the second year they enter.
2. The patients seen by the hospital each year are whole numbers (How would you see part of a patient?).
3. The number of patients seen in the first year will be less than the number of patients seen in the second year.
4. The region's average charity-charges-per-patient ratio is $1008.97.

Example *Ledgeon* input and output (text in bold is the user's input):

```
Enter the first year: 2019
Enter the number of patients we saw that year: 25540
Enter the second year: 2021
Enter the number of patients we saw that year: 26817
In year 2019, our hospital issued 2.57691e+07 dollars of charity charges.
In year 2021, our hospital issued 2.70575e+07 dollars of charity charges.
Between 2019 and 2021, there was a 5% increase in patients seen at our hospital.
```

Note: Your program will be tested against other test cases. Your program must compute properly in all cases in order to receive full points! Make sure that you check the autograder to make sure that your program behaves as expected.

You *must* use properly-typed variables to store the value of intermediate calculations. In other words, you may not simply output the result of the calculation. Going through these (seemingly) unnecessary steps will help you practice using the *assignment operator*.

Although we know that an apple a day *can* keep the doctor away, reading the entire lab before you start is guaranteed to bring the Apple. If you are the first person to read this sentence, come see me for an Apple gift card!

Getting Started

Note: In the *Getting Started* section of labs, I will tell you how *I* would get started. If you have a plan for how to complete the lab, feel free to skip this section and get right to work.

When faced with a task like the one in this lab, my first step would be to make sure I can gather all the input from the user correctly. In other words, I would make sure that I correctly prompt the user for their input and store their responses in variables correctly. Let's walk through how I would do that for the first piece of data that *Ledgeon* will gather from the user.

The first prompt that the user should receive is one that asks them to enter the "first year". We know from the lab that the prompt should be

```
Enter the first year:
```

Step 1, then, would be to write code that will print that prompt to the screen. To print something to the screen, we use `std::cout`, right? Great!

```
std::cout << "Enter the first year: "
```

When I run my application now, I expect it to output

```
Enter the first year:
```

Once I saw that output, I would clearly be very happy! I would have the first part of the application written.

Now, I would write code to capture the user's response. I would need a variable to store what they input, so I'd declare one! Years are *whole numbers*, and I want to store the user's response in a variable whose type is *perfectly* suited to hold whole numbers! Having decided that an int is the perfect type for this kind of data, I would need to decide on the variable's name. I would make sure that I named my variable according to its purpose in the program! Oh, and I'd have to remember to initialize it to a good default value, too. I wouldn't want to run the risk of having an *indeterminate value* in my program!

With my properly-typed variable declared/defined to hold the user's response, I could actually write the code that would read the user's input from the command line and store it in the variable. Remember that we use `std::cin` to do that. Because I named my variable `year1`, I would write the following code:

```
std::cin >> year1;
```

Another smile! I am really making progress.

At this point I could absolutely move on to writing code to prompt the user for their next input. However, I am a cautious person. My next step would be to write code that prints the contents of the variable that I used to store the user's response back out so that I can verify that it holds the correct value. I would write some code that prints a little "label" and the value of the variable. To do that, I would rely, again, on `std::cout`:

```
std::cout << "The user entered: " << year1 << "\n"
```

If I run my program now, I would see

```
Enter the first year: 2019
The user entered: 2019
```

(Remember: values in bold are what the user enters.)

I would try my code on several different input values and make sure that it works the way I expected.

Wohoo! Now I am on my way! Before continuing I would obviously remove the code that helped me verify that I got the user's input correctly. It's "debugging" code, after all, and not part of the way my application should ultimately work.

Critical Thinking Task

Type checking a program written in a high-level programming language can catch lots of bugs before the application is released to users. Your critical thinking task is to answer the following three questions:

1. What is the definition of *type*? (Hint: there are two parts to the definition)
2. What is the difference between a *strongly* and *weakly typed* high-level programming language?
3. Why do applications written in a strongly typed high-level programming language usually have fewer bugs than applications written in weakly typed high-level programming languages?

Critical Thinking Requirement

Your submission must include answers for all three prompts. Your submission cannot be more than 200 words and you may use resources from outside class. All references to external resources must be properly documented and formatted. The choice of formatting for external references is up to you, but you may find it helpful to consult the Purdue OWL for **help (https://owl.purdue.edu/owl/research_and_citation/apa_style/apa_style_introduction.html)** The Purdue OWL also has extensive information on ways to **avoid plagiarism (https://owl.purdue.edu/owl/avoiding_plagiarism/index.html)** .

Question Task:

For this lab you deployed your knowledge of the mathematical operators and the difference between `int`eger and floating-point (`double`) division. You also learned how to accept input from the user using `std::cin`. Along the way we explored some of the subtleties of C++. Was there anything surprising? Something that doesn't make sense? Please feel free to ask anything related to anything that we've learned so far (or will learn)!

Question Requirement:

Submit a file named `question.pdf` which contains your question. Take particular notice of the fact that the question does not have to be subtle, or deep or philosophical for you to receive full points. The goal of the question is for me to be able to get a sense whether there are common questions about the material which would indicate where I failed to teach the material appropriately.

Deliverables

1. The pseudocode describing the algorithm of your *Ledgeon* program in PDF format (named `design.pdf`).
2. The C++ source code for your *Ledgeon* application (named `ledgeon.cpp`).

3. A written response to the three questions in the Critical Thinking Task in PDF format (named `type.pdf`).
4. A written question about the material covered so far in this class in PDF format (name the file `question.pdf`).

Rubric

Points	Task	Criteria
15	Design	Pseudocode uses algorithmic thinking to express a high-level description of how to implement the *Ledgeon* application and meets the specified criteria.
5	Critical Thinking	Definition of *type* is correct.
5	Critical Thinking	Description of the difference between strongly and weakly typed languages is correct.
5	Critical Thinking	Reason why applications written in a strongly typed high-level programming language usually have fewer bugs than applications written in weakly typed high-level programming languages is persuasive.
15	Programming	Program prompts for user input according to the specified format.
5	Programming	Program uses variables to store the result of calculations.
5	Programming	Program uses a `const` variable to store the charity-charges-per-patient ratio.
15	Programming	Program correctly calculates its outputs for all test cases.
15	Programming	Program displays output according to the specified format for all test cases.
5	Programming	Program uses appropriately named variables.

5	Programming	`main` function’s purpose is described by a grammatically correct multi-line comment.
5	Question	Your question shows an engagement with the material.

Related Learning Objectives

1. Students will understand the difference between a *strongly* and *weakly typed* programming language.
2. Students will effectively use named constants to increase code readability.
3. Students will be able to define *type*.
4. Students will be able to use basic arithmetic operators on primitive types.
5. Students will understand when floating-point and decimal multiplication/division occur.
6. Students will be able to declare and use variables of primitive types.
7. Students will be able to use an assignment statement to modify a variable.