


This is CS50x

CS50's Introduction to Computer Science

OpenCourseWare



Donate  (<https://cs50.harvard.edu/donate>)


David J. Malan (<https://cs.harvard.edu/malan/>)

malan@harvard.edu


 (<https://www.clubhouse.com/@davidjmalan>)  (<https://www.facebook.com/dmalan>)

 (<https://github.com/dmalan>)  (<https://www.instagram.com/davidjmalan/>) 

(<https://www.linkedin.com/in/malan/>)  (<https://orcid.org/0000-0001-5338-2522>) 

(<https://www.quora.com/profile/David-J-Malan>)  (<https://www.reddit.com>

/user/davidjmalan)  (<https://www.tiktok.com/@davidjmalan>) 

(<https://davidjmalan.t.me/>)  (<https://twitter.com/davidjmalan>)

Lab 1: Population Growth

You are welcome to collaborate with one or two classmates on this lab, though it is expected that every student in any such group contribute equally to the lab.

Determine how long it takes for a population to reach a particular size.

```
$ ./population
Start size: 100
End size: 200
Years: 9
```

Background

Say we have a population of n llamas. Each year, $n / 3$ new llamas are born, and $n / 4$ llamas pass away.

For example, if we were to start with $n = 1200$ llamas, then in the first year, $1200 / 3 = 400$ new llamas would be born and $1200 / 4 = 300$ llamas would pass away. At the end of that year, we would have $1200 + 400 - 300 = 1300$ llamas.

To try another example, if we were to start with `n = 1000` llamas, at the end of the year, we would have `1000 / 3 = 333.33` new llamas. We can't have a decimal portion of a llama, though, so we'll truncate the decimal to get `333` new llamas born. `1000 / 4 = 250` llamas will pass away, so we'll end up with a total of `1000 + 333 - 250 = 1083` llamas at the end of the year.

Getting Started

Recall that Visual Studio Code (aka VS Code) is a popular “integrated development environment” (IDE) via which you can write code. So that you don't have to download, install, and configure your own copy of VS Code, we'll use a cloud-based version instead that has everything you'll need pre-installed.

1. Log into code.cs50.io (<https://code.cs50.io/>) using your GitHub account and follow the on-screen instructions to set up your very own “codespace” for Visual Studio Code. Once your codespace loads, you should see that, by default, VS Code is divided into three regions. Toward the top of VS Code is your “text editor,” where you'll write all of your programs. Toward the bottom is a “terminal window,” a command-line interface (CLI) that allows you to explore your codespace's files and directories (aka folders), compile code, and run programs. And on the left is your file “explorer,” a graphical user interface (GUI) via which you can also explore your codespace's files and directories.
2. Once your codespace has loaded, close any **Welcome** tabs that might have opened by default
3. Log into submit.cs50.io (<https://submit.cs50.io>) using your GitHub account and click **Authorize cs50** to activate `check50`.
4. Run `update50` in your codespace's terminal window to ensure your codespace is up-to-date and, if prompted, click **Rebuild now**.

Once complete, start by clicking inside your terminal window, then execute `cd` by itself. You should find that its “prompt” resembles the below.

```
$
```

Click inside of that terminal window and then type

```
mkdir population
```

followed by Enter in order to make a directory called `population` in your codespace. Take care not to overlook the space between `mkdir` and `population` or any other character for that matter!

Here on out, to execute (i.e., run) a command means to type it into a terminal window and then hit Enter. Commands are case-sensitive, so be sure not to type in uppercase when you mean lowercase

or vice versa.

Now execute

```
cd population
```

to move yourself into (i.e., open) that directory. Your prompt should now resemble the below.

```
population/ $
```

Click inside of that terminal window and then type

```
wget https://cdn.cs50.net/2022/fall/labs/1/population.c
```

followed by Enter in order to download a template file called `population.c` in your codespace. Take care not to overlook the space between `wget` and the following URL, or any other character for that matter! If all was successful, you should execute

```
ls
```

and see a file named `population.c`. Executing `code population.c` should open the file where you will type your code for this lab. If not, retrace your steps and see if you can determine where you went wrong!

Implementation Details

Complete the implementation of `population.c`, such that it calculates the number of years required for the population to grow from the start size to the end size.

- Your program should first prompt the user for a starting population size.
 - If the user enters a number less than 9 (the minimum allowed population size), the user should be re-prompted to enter a starting population size until they enter a number that is greater than or equal to 9. (If we start with fewer than 9 llamas, the population of llamas will quickly become stagnant!)
- Your program should then prompt the user for an ending population size.
 - If the user enters a number less than the starting population size, the user should be re-prompted to enter an ending population size until they enter a number that is greater than or equal to the starting population size. (After all, we want the population of llamas to grow!)
- Your program should then calculate the (integer) number of years required for the population to reach at least the size of the end value.
- Finally, your program should print the number of years required for the llama population to

reach that end size, as by printing to the terminal `Years: n`, where `n` is the number of years.

Walkthrough



Hints

- If you want to repeatedly re-prompt the user for the value of a variable until some condition is met, you might want to use a `do ... while` loop. For example, recall the following code from lecture, which prompts the user repeatedly until they enter a positive integer.

```
int n;  
do  
{  
    n = get_int("Positive Integer: ");  
}  
while (n < 1);
```

How might you adapt this code to ensure a start size of at least 9, and an end size of at least the start size?

- To declare a new variable, be sure to specify its data type, a name for the variable, and (optionally) what its initial value should be.
 - For example, you might want to create a variable to keep track of how many years have passed.

- To calculate how many years it will take for the population to reach the end size, another loop might be helpful! Inside the loop, you'll likely want to update the population size according to the formula in the Background, and update the number of years that have passed.
- To print an integer `n` to the terminal, recall that you can use a line of code like

```
printf("The number is %i\n", n);
```

to specify that the variable `n` should fill in for the placeholder `%i`.

How to Test Your Code

Your program should behave per these examples below.

```
$ ./population
Start size: 1200
End size: 1300
Years: 1
```

```
$ ./population
Start size: -5
Start size: 3
Start size: 9
End size: 5
End size: 18
Years: 8
```

```
$ ./population
Start size: 20
End size: 1
End size: 10
End size: 100
Years: 20
```

```
$ ./population
Start size: 100
End size: 1000000
Years: 115
```

► Not sure how to solve?

Execute the below to evaluate the correctness of your code using `check50`. But be sure to compile and test it yourself as well!

```
check50 cs50/labs/2023/x/population
```

Execute the below to evaluate the style of your code using `style50`.

```
style50 population.c
```

How to Submit

In your terminal, execute the below to submit your work.

```
submit50 cs50/labs/2023/x/population
```

