lab03 : Counting ducks: File I/O and flow control

num	ready?	description	assigned	due
lab03	true	Counting ducks: File I/O and flow control	Tue 10/29 09:00AM	Tue 11/05 11:59PM

Introduction

By the time you have completed this lab, you should be able to:

- Use for loops to count up and count down
- Check command line argument for input filename, and open that input file for reading
- Read every line of text in an input file and process it, using a while loop
- Use if-else statements to select blocks of code
- Count how many times a value occurs in a file
- Use Unix commands to create directories, navigate to directories, list files, and copy files
- Use the “make” command to compile simple stand-alone C++ programs (i.e. single source code file)
- Run simple C++ programs both with and without a single command line parameter

Step by Step Instructions

Step 1: Log on to CSIL and bring up a terminal window.

I hope I can safely assume that you have all gotten a CoE account. If your account is not working, get the attention of the instructor.

Log into your account to make sure it works. As a reminder to get to the terminal go to **Application** Menu, then **System Tools**, then **Terminal Window**.

In the steps below, and in most future labs, you will create files on your own account.

Step 2: Create a new repo, add your partner as collaborator and clone it to your local directory

In lab02, we have done the same thing. So if you don’t know to how to do that, please refer to lab02 for details. The basic steps are as follows:

- Create a git repo on github following the correct naming convention, e.g., if your github username is jgaucho and your partner’s is alily, your should name your repo lab03_agaucho_alily (usernames appear in alphabetical order). Don’t forget to make it ‘PRIVATE’.

- Add your partner as a collaborator for the repo.

- Go to your CS16 directory and clone the repo locally.

Step 3: Get the starter code from a local directory

Copy the skeleton to your local lab03 repo using the following command, **REMEMBER** to change the directory name in the commands below to your own directory’s name, in this lab we simply use lab03_agaucho_alily as a example for your local git directory:

```
  cp /cs/faculty/dimirza/cs16/labs/lab03/* ~/cs16/lab03_agaucho_alily/
```

After doing this command, if you cd into ~/cs16/lab03_agaucho_alily/ and use the ls command, you should see several .cpp files:

```
-bash-4.2$ ls
animals01.txt  animals02.txt  countDucks.cpp  sample01.cpp
-bash-4.2$
```

If you don’t see those files, work with your pair partner to go back through the instructions and make sure you didn’t miss a step. If you still have trouble, ask for assistance.

If so, you are ready to move on to the next step.

Step 4: Compile and run the first program for this assignment

The first program we are going to compile and run is one that demonstrates a for loop in C++.

In your lab03 directory, you should have a program called sample01.cpp that we copied in the previous step. Here’s how you can put yourself in that directory (though you should already be there):

```
-bash-4.2$ cd ~/cs16/lab03_agaucho_alily/
-bash-4.2$ pwd
/cs/faculty/dimirza/cs16/lab03_agaucho_alily/
-bash-4.2$
```

Then you can list out your files with the `ls` command:

```
-bash-4.2$ ls
animals01.txt  animals02.txt  countDucks.cpp  sample01.cpp
-bash-4.2$
```

Finally, use the Unix `cat` command to list the contents of the file sample01.cpp. (The reason this command is called "cat" has nothing to do with the animal that goes "meow". If you ask me in lecture and I'll tell you where the name comes from.)

```
-bash-4.2$ cat sample01.cpp
// sample01.cpp
#include <iostream>
using namespace std;

int main() {
    // Simple for loop that counts from 1 up to n

    int n=5;

    for (int i=1; i<=n; i++) {
       cout << "i=" << i << endl;
    }

    return 0;
}
-bash-4.2$
```

Compile this with the command `make sample01` and run it with the command `./sample01`. That looks like this:

```
-bash-4.2$ make sample01
g++     sample01.cpp   -o sample01
-bash-4.2$ ./sample01
i=1
i=2
i=3
i=4
i=5
-bash-4.2$
```

If you get that output, you are ready for the next step.

Step 5: Copy sample01.cpp to myProg01.cpp and make changes

Now we'll use the the Unix command `cp` *oldfile newfile* which copies files, to copy from sample01.cpp to a new file called myProg01.cpp, as shown here:

```
-bash-4.2$ cp sample01.cpp myProg01.cpp
-bash-4.2$ ls
animals01.txt  countDucks.cpp  sample01
animals02.txt  myProg01.cpp    sample01.cpp
-bash-4.2$
```

Now you have a new file called myProg01.cpp that is a copy of sample01.cpp. Open it up in a text editor and make the following changes:

1. Change the comment at the top of the file so that it says // myProg01.cpp
2. Change the second line of the file to be of the format "Author: your name"
3. Change the comment within the code to "Simple that counts down from n to 1"

4. Change the for loop as follows:

- Instead of initializing to 1, initialize to n
- Instead of testing `i<=n`, test whether `i>0`
- Instead of changing i by incrementing with `i++`, change it by decrementing with `i--`
- Remove the printing of "i=" each time. Instead just print the number. And Instead of printing a newline after each number, just print one space. We do that by changing `cout << "i=" << i << endl;` to `cout << i << " ";`
- Add a line that prints a newline at the very end, just after the for loop is over, but BEFORE the `return 0;` statement. *Note: As a reminder, you get out of vim or gvim with ESC:x or ESC:wq . You can get out of emacs with CTRL+X followed by CTRL+C. It will ask if you want to save changes; type y for yes.*

Compile and run myProg01.cpp with these changes. The output should look like this:

```
5 4 3 2 1
```

You are now ready to move to the next step.

Tip: *If you make a mistake that results in an "infinite loop", i.e. the window is just scrolling by without stopping, you can use CTRL+C (hold down Control and type C) to stop the program.*

Step 6: Reading from input files and counting ducks

The next files we are going to look at are not C++ code, but rather data files.

Use the "cat" command to look at the contents of animals01.txt and animals02.txt. You should get results like these:

```
-bash-4.2$ cat animals01.txt
duck
duck
goose
-bash-4.2$ cat animals02.txt
duck
duck
goose
duck
duck
cow
duck
duck
dog
-bash-4.2$
```

The next program we are going to look at will read input from files such as these. It is called `countDucks.cpp` and it will simply count the number of ducks in each file.

Before you look at the code, try compiling the program and running it, because this will help you understand what the program is trying to do. Compile with:

```
-bash-4.2$ make countDucks
g++     countDucks.cpp   -o countDucks
-bash-4.2$
```

Then try running it with just `./countDucks`. You'll see that you get a "Usage" message. This is telling us that the program expects a "command line argument", which is the name of the file to read:

```
-bash-4.2$ ./countDucks
Usage: ./countDucks inputFile
-bash-4.2$
```

IMPLEMENT THE COUNTDUCKS PROGRAM in countDucks.cpp so that when you run the program with argument animals01.txt as the filename, it produces the following output:

```
-bash-4.2$ ./countDucks animals01.txt
There were 2 ducks in animals01.txt
```

Alternatively if you give animals02.txt as the argument it should produce the following output.

```
-bash-4.2$ ./countDucks animals02.txt
There were 6 ducks in animals02.txt
-bash-4.2$
```

Once you've done that, you are ready for the next step.

Step 7: A more detailed counting program

Your job is now to copy countDucks.cpp to a file myProg02.cpp and make some changes.

First, let's stipulate that you may assume that everything in the input file is an animal, one per line—if someone adds "potato" or "bicycle" to the file, you can just assume that potato and bicycle are now to be considered types of animals.

1. Add a variable that will count ALL animals in the file. Give it an appropriate name and initialize it to zero.
2. Add a variable that will count ALL animals in the file that are NOT ducks. Give it an appropriate name and initialize it to zero.
3. Add code that will increment those counts when appropriate. It may help to know that C++ has an else clause for an if that looks like this:

```
   if (condition) {
     // lines of code here are
     // executed when condition is true
   } else {
     // lines of code here are
     // executed when condition is false
   }
```

Note that it is NOT required for every if to have an else clause.

Also note that the braces `{ }` are:

- OPTIONAL when there is a SINGLE statement inside a particular if or else block
- REQUIRED when there is more than one statement inside a particular if or else block

After making these changes, one more thing: change the lines that give the output so they look like the ones shown below.

```
Report for animals01.txt:
   Animal count:    3
   Duck count:      2
   Non duck count:  1
```

```
Report for animals02.txt:
   Animal count:    9
   Duck count:      6
   Non duck count:  3
```

It is IMPORTANT to be EXACT since the submit.cs system will compare your output with the expected output character-by-character. The spacing MATTERS! You can add extra spaces at the beginning and end of the string literals for `" Animal count: "` and `" Duck count: "` so that the spacing comes out right and matches the expected output below. I'm not going to tell you how many; you'll have to figure that out.

Note that we will also test your program on other input files, so you should too. Use the cp command to copy animals02.txt to animals03.txt and add some ducks and some other animals. Count by hand, and make sure that the count when you run your program matches what is expected.

When you are satisfied that the count is correct and that format of the output is precise, you are ready to submit your code for grading.

Step 8: Turn in your code on gradescope

- Navigate to your ~/cs16/lab03_agaucho_alily directory, the one containing your code for this week's lab.

```
-bash-4.2$ cd ~/cs16/lab03_agaucho_alily
```

- Use the `ls` command to list your files and to be sure that you have the myProg01.cpp and myProg02.cpp files in your directory. It is ok if there are other files (countDucks.cpp, animals01.txt, etc.) along with the executables. You only have to submit myProg01.cpp and myProg02.cpp.

```
-bash-4.2$ cd ~/cs16/lab03_agaucho_alily
-bash-4.2$ ls
animals01.txt  countDucks.cpp  myProg02      sample01
animals02.txt  myProg01.cpp    myProg02.cpp  sample01.cpp
-bash-4.2$
```

Submit all the cpp files to lab03 assignment on gradescope. Then visit gradescope and check that you have a correct score. If you are working with a partner, make sure both of you join a team on gradescope, otherwise only one of you will get credit for the lab

- You must check that you have followed these style guidelines:

1. Indentation is neat, consistent and follows good practice (see below)
2. Variable name choice: variables should have sensible names. More on indentation: Your code should be indented neatly. Code that is inside braces should be indented, and code that is at the same "level" of nesting inside braces should be indented in a consistent way. Follow the examples from lecture, the sample code, and from the textbook.

- Your submission should be on-time. If you miss the deadline, you are subject to getting a zero