Terminal Directory and New Horizons (Lab 01)

Partners

We do our laboratory exercises with partners. In this way you can collaborate with someone else and, together, can figure out the problem. The experience you gain in the lab will help you with the project work. Remember, laboratory exercises are collaborative, project is homework to be done alone! Choose a partner in the lab to work with on this exercise. Two people should work at each pair of computers. Occasionally switch who is typing.

Part 1: Learning About the Terminal Application

At the beginning of most labs, will be a section on new skills to make you more proficient at the command line. Many tasks can only be performed at the terminal (or are easier to perform at the terminal), so becoming comfortable there is important. Many of the classes after this one will expect at least a basic familiarity of the command line, so pay special attention to these sections.

Log into x2go or go to Mimir and open the IDE. Use the terminal to follow along.

**pwd**

The first command you need to learn is one of the simplest, *pwd*. At the terminal, type the command *pwd* and hit enter.

*pwd* is short for "print working directory". It outputs the path of the working directory, the directory your terminal currently has open. If you get lost (and it is easy to do with only the text-based terminal for navigation), *pwd* can show you where you are.

**ls**

The *ls* command outputs the names of all of the folders and files in the working directory. *ls* is short for "list" (as in "list" the contents of the directory).

Folder names end with a /, file names do not. Some terminals also add colorized output to *ls* to denote different types of files. For instance, Mimir's terminal has folders in blue, and files in white.

**cd**

The *cd* command is short for "change directory". It allows you to change your working directory to some different folder.

Let's say you run *pwd* and you get the output of:

```
/home/joshua/cse_232__summer_2017/lab01_new_horizons
```

Your current working directory is named "lab01_new_horizons". If you run *ls*, you would get:

```
lab01/
```
This means there is only one thing in this folder, a subfolder called "lab01". If you wanted to do things in that subfolder, you would use the cd command like so:

   cd lab01

Now you are in the folder named "lab01", you can confirm such with pwd and ls

**Special Directory Names**

There are a few "special" directory names that you need to know.

The first is the home directory. This directory's name is usually the username of the account, and it contains all of the user's files and folders. Inside the home folder is the "Desktop" folder (for all the things on your desktop), the "Downloads" folder and other folders as well (like "Music", "Videos", "Applications", etc.). The home folder is often specified using the tilde symbol (~). So if you want to cd to your home folder, run:

   cd ~

Note: On most systems, running cd with no arguments also takes you to the home folder.

Sometimes you want to go the parent folder of your working directory. In the example above, we moved from the folder "lab01_new_horizons" to its subfolder "lab01". Trying to get back by running:

   cd lab01_new_horizons

would fail, because there is no folder named "lab01_new_horizons" in "lab01".

To reach to the parent directory (in this case "lab01_new_horizons"), you need to run:

   cd ..

The .. directory (two dots) is a strange way to symbolize the parent directory. In fact, the . directory (single dot) denotes the current working directory, the present directory that pwd would print, a fact that will be useful to know in later labs.

The last "special" directory is the root directory. The root directory is the directory that is the ultimate parent of all the other directories on the computer. It is denoted by the single / symbol. In fact, you can specify any folder on the computer by starting with the root directory and working your way to the directory you actually want. Thus, the path

   /home/joshua/cse_232__summer_2017/lab01_new_horizons

is a fully qualified path. We can break this down as follows:

- In the root folder (the starting /)
- In the folder named "home" (a subdir of the root)
- In the folder named "joshua" (a subdir of home)
- In the folder named "cse_232__summer_2017" (a subdir of joshua)
- In the folder named "lab01_new_horizons" (a subdir of cse_232__summer_2017)

It is an unambiguous statement of location.
Part 2: Coding Application

Note on Mimir:

Mimir provides a series of tests that get run every time you hand in. It gives you feedback on how well your program works. When you pass all the tests, you are done! If you can't pass all the tests, turn in what you can and pass as many tests as possible. Always turn something in! You can turn in programs using Mimir as often as you want, though we strongly recommend testing programs on your own first.

Background

The New Horizons spacecraft, launched January 19th 2006, is the first earth spacecraft to have made contact with the planet Pluto. On January 1st, 2019 it is scheduled to make contact with the first Kuiper belt object KBO-2014-KU69. The NASA update page (http://pluto.jhuapl.edu/Mission/Where-is-New-Horizons/index.php) reports it (09/01/2017) at a distance of 39.33 Astronomical Units (AU) from the Sun, traveling away from the Sun at 14.24 km/sec, 8.85 mi/sec.

For this lab you will use the cin and cout streams along with some simple mathematics for calculating New Horizon’s distance. The important part of the project is to learn the skills needed to access the class web site, to access a project description, create a new program in C++ and finally to hand it in.

Your Task

The basic design of the first programs that you construct in this class consists of a prompt for information, receiving information, processing that information then producing a display of the results.

Your program will prompt the user for an integer number (a number without decimal points) which indicates the number of days after 09/01/2017, starting at the distance 39.33 AU from the sun. You will calculate the distance of New Horizons from the Sun using the numbers from 09/01/2017 (assume velocity is constant) plus the user provided number of days and report:

- Updated distance in A.U.
- Distance in kilometers (1 AU = 149,598,000 km) on a line by itself
- Distance in miles (1 AU = 92,955,800 mile) on a line by itself
- Round trip time for radio communication in hours. Radio waves travel at the speed of light, listed at 299,792,458 meters/second, on a line by itself
- provide 2 decimal points of accuracy using std::fixed and std::setprecision (the later requiring #include). You would use them as follows:
  o std::cout << std::fixed;
  o std::cout << std::setprecision(2);

At this point, you should show your TA your working program

Assignment Notes

There is a Mimir assignment labeled Lab01: New Horizons with 3 test cases. There is a starter file with the correct directory and name, lab01.cpp, but no content (nothing in the file).
There are some rounding issues here so be careful! To make the km calculations, use the constants (speed and distance) provided. To make the mile calculations, use the constants (speed and distance) provided. To make the round trip calculation, use your distance in km and the speed of light constant provided. You'll get slightly different answers if you try to convert the two distance or the two speeds.

To work locally, you can use the data from the Mimir test cases. Look at the test case and the input and output. Save them into pairs of files, for example inXX.txt and outXX.txt examples, where XX is the test number: in1.txt should match out1.txt, int2.txt should match out2.txt and so on. The output should match exactly. Working locally is a good idea — use Visual Studio as we discussed in Lab00

You will need to work with the cin and cout streams and their operators >> (for cin) and << (for cout). You will also need to declare the appropriate variables: long (a 64 bit integer) for values like days and double for floating point calculation values.

cout takes either variable values or strings (between " ") and outputs them to the console. You can use multiple << operators on the same cout stream, usually ending with endl. Assuming the variable int_var has the value 23:

```cpp
std::cout << "This is a string: " << int_var << " the end" << std::endl
```

would output:

This is a string: 23 the end

cin will input a value from the command line into a variable of a particular type. It does so until it hits a space (space separated values) or an end of line. For example:

```cpp
std::cin >> int_var;
```

If you enter a value on the command line, an integer, it will be read into the variable (no conversion required). The following code demonstrates how this works:

```cpp
int multiplier, number;
std::cin >> number;
std::cin >> multiplier;

std::cout << "The number " << number << " times " << multiplier << " is " << number * multiplier << std::endl;
```

With inputs 10 and 2 respectively, you would get as output:

The number 10 times 2 is 20

The basic operations on numbers are, respectively: + (sum), - (difference), * (product), / (division) and % (remainder, integer only). The last two deserve special comment.

If an integer is divided by another integer, the result is an integer. Thus the result of 6/4 is 1. In contrast, 6.0/4 is 1.5. That is, the / operation results in the integer quotient if using integers, floats if using floats.
The result of 6%4 is the integer remainder of the division, thus 2 (6 divided by 4 is 1 with a remainder of 2). There is no equivalent for % in floating point math.

**Getting Started**

1. Create a new directory and call it lab01
2. Create a new Visual Studio project like we did in lab00 and create a file in this project named “lab01.cpp” to write code in
3. Using the example from earlier in the lab, and the hello world programs from last week, write the code to perform the tasks.
4. Don't worry if you get an error that looks long and difficult. C++ is like that. Visual Studio will often point out the line that the error is coming from. That line and the lines around it are your best clue at finding what is wrong.
5. Run the program
6. Edit the program to fix any errors or add functionality
7. Now you enter a cycle of edit-save-compile-run to incrementally develop your program.

**Questions for you to consider**

1. What happens when you try to divide by zero when you run your program?
2. What happens when you give std::cin a letter instead of a number?