SYDE 121

Lab Number 3

You are expected to continue using the design methodology used in the previous labs. Although this design will not be submitted nor graded, the onus is on the student to ensure that their designs (e.g. problem analysis, top-down decomposition, pseudo-code) are prepared prior to trying to code the solution. In general, this will save time and hair pulling in front of the computer. It is **highly recommended** that you develop the habit of planning your program before you type it into the computer.

Note that there are some marks allocated to the programming exercises with regards to "code conciseness and readability" so consult the Style Guide to start building good habits with regards to this.

1. Exercise 1: Weeks, Days, Hours, Minutes, Seconds

Learning Objectives: To apply integer division and the % (modulus) operator; to gain more experience with input, computation with assignment statements and arithmetic expressions, and output.

Read This First

Converting an integer number of seconds to weeks, days, hours, minutes and seconds is similar to converting an integer number of cents into numbers of bills and coins.

What To Do

Write a program (call it "*lab0301.cpp*") that asks a user for an integer number of seconds. The program should print out the length of the time interval in weeks, days, hours, minutes and seconds, where the number of hours is ≥ 0 and < 24, and the numbers of minutes and seconds are each ≥ 0 and < 60. You must utilize the modulus operator to perform these calculations.

Ensure that the number of seconds entered is greater than zero. You can do this using an `if` operator and simply exiting (by using the `return` command) if the number is not appropriate. A more clever way of dealing with this is to use the `while` loop (Chapter 2) which, would allow you to iterate until the proper input is entered. Either method is appropriate for this lab, but in the future you would be expected to use the `while` loop to ensure appropriateness of data entry.

Create a "*README_Lab0301.txt"* file, and create detailed instructions on how to run your *lab0301.cpp* program, including what numeric type and units that input should be entered as (see Previous Labs for instructions on README files.)

What To Submit

Submit both your *lab0301.cpp* files and *README_Lab0301.txt* files to the Lab Assignment 3 LEARN dropbox.

---

2. Exercise 2: Run-Time Errors

Learning Objectives: To learn to recognize signs that illegal arithmetic operation has been attempted.

Read This First

Certain arithmetic and math operations do not make sense. One example is dividing by zero. Another example — assuming that complex results are not allowed — is taking the square root of a negative number. Errors of this sort fall into the category of *run-time errors*, because they occur while a program is running, not when the program is being compiled.

The response of a C or C++ program to errors such as division by zero and illegal arguments to math functions depends on the computer hardware, operating system, and compiler. It is very helpful to be able to recognize the effects of arithmetic and math run-time errors in C or C++ programs compiled with your particular system.

What To Do

Download the file 'runtime_errors.cpp' from the Lab Assignment 3 LEARN dropbox. Build the executable and run it. Make sure it produces correct answers when you enter positive numbers to all of the prompts.

Run the program a few more times in order to determine the effect of the following run-time errors:

- division of an int by the int value of 0
- division of a double by the double value of 0.0
- trying to compute the square root of a negative number

For each of the above cases, answer these questions:

1. Does the program keep running after the error occurs?
2. What output appears as the result of the computation, if any?

What To Submit

Submit your responses in a *lab0302.txt* file to the Lab Assignment 3 LEARN dropbox.

Exercise 3: Wind-Chill Index

Learning Objectives: Practice using **if** … **else commands**. Use of sqrt function.

Read This First

The temperature reported by TV stations usually refers to the air temperature. However, even light to moderate wind can cause an *effective temperature* much lower than that of the air temperature. It would be useful to be able to determine the wind-chill, based on the air temperature and wind speed.

An empirical (not so effective!) formula for the wind-chill index is:

$$\text{wind-chill index} = \begin{cases} temperature & wind \le 4\,\text{mph} \\ 91.4 - (10.45 + 6.69\sqrt{wind} & \\ \quad - 0.447\,wind) \times \dfrac{91.4 - temperature}{22.0} & 4\,\text{mph} < wind \le 45\,\text{mph} \\ 1.6 \times temperature - 55.0 & wind > 45\,\text{mph} \end{cases}$$

where,

temperature is the temperature in degrees Fahrenheit, and

wind is the wind speed in miles per hour (mph).

Note: second function is stretched over two lines.

What To Do

As usual, you are expected to design your program before you try to enter it into a cpp file. Also, create a few hand calculated examples for program testing prior to coding your solution. Use temperatures typical to Canadian winters.

Write a program, *lap0303.cpp*, that calculates the wind-chill index for a given combination of temperature and wind. Your main function should ask the user for the air temperature in **Celsius**, and the wind speed in **kilometres per hour (km/h)** (ensure that the wind speed is positive), perform the necessary unit conversions, and tell the user the computed wind-chill index (in the appropriate units i.e. convert your calculation back to Celsius).

Note: do not be concerned about the use of this model in practice – it is not a real conversion, just one that has been made up for this lab.

Create a "*README_Lab0303.txt"* file, and create detailed instructions on how to run your *lab0303.cpp* program, including what numeric type and units that input should be entered as (see Previous Labs for instructions on README files.)

Some Help

You will need to take the square root of a number. This is a *predefined function* call in C++ and it is determined as follows:

```
a = sqrt( b );
```

where **a** will be assigned the square root of **b** and both **a** and **b** are type `double`. This function is included in the math library, so you will have to include <cmath> using an 'include' compiler directive, as below:

```
#include <cmath>
```

A description of predefined functions is provided in your Savitch text (we will cover this material in detail in Section 5 of the course).

Conversions

1 km/h = 0.6215 mph

Fahrenheit = (9/5) * Celsius + 32;

What To Submit

Submit both your *lab0303.cpp* files and *README_Lab0303.txt* files to the Lab Assignment 3 LEARN dropbox.

Due Date

All material for each exercise must be submitted to LEARN by **Friday, October 4, 1:30pm**.