

CSCI 1300/1310 Introduction to Computer Programming
Instructors: Knox/Hoenigman
Assignment 1
Due Friday, September 2 by 12:30pm

For the following problems, write the algorithm using the building blocks discussed in the Lecture 2-3 (Five_building_blocks_of_representation.pdf). Use the commands from the section in the notes titled *Lecture 2 – Representation*. In your algorithms, use indentation to show that a line in the algorithm is inside of a loop or a conditional. For example:

```
    if x equals y
        output(x)
```

The line *output(x)* only executes if *x equals y* is true and is therefore indented under the if statement.

How to Submit:

Your work should be typed. Submit your assignment as a PDF on Moodle using the Assignment 1 Submit link.

Problems:

1. The U.S. Census provides information about the current U.S. population as well as approximate rates of change. Three rates of change are provided:
 - a. There is a birth every 7 seconds
 - b. There is a death every 13 seconds
 - c. There is a new immigrant every 35 seconds

Using those three rates of change, and a current U.S. population of 307,357,870, write a program to calculate the U.S. population in exactly one year (365 days). Your algorithm should output the result of your calculations.

2. A day has 86,400 seconds ($24 \cdot 60 \cdot 60$). Given a number of seconds in the range of 0 to 86,400, output the time as hours, minutes, and seconds for a 24-hour clock. For example, 70,000 seconds is 19 hours, 26 minutes, and 40 seconds. Your program should have user input that is the number of seconds to convert, and then use that number in your calculations. Your output should be displayed as *"The time is X hours, Y minutes, and Z seconds"*.
3. In science, temperature is always described in Celsius, but in the U.S. we tend to use Fahrenheit temperatures. Write an algorithm to convert a Fahrenheit temperature into Celsius. (Subtracting thirty-two from the Fahrenheit value and taking five ninths of that result will provide the Celsius value)

4. Write an algorithm that asks a user to enter a number between 1 and 10. (This range includes the numbers 1 and 10.) When they enter the number, check that it is actually between 1 and 10. If it is not, ask them to enter a number again. Continue to ask until they enter a valid number. Once their number is valid, output the number.
5. Write an algorithm that asks a user the miles per gallon of their car. If they say greater than 30, output "Nice job". If they say between 15 and 29, output "Not great, but okay." If they say less than 15, output "So bad, so very, very bad."
6. In text-based choose your own adventure games, the game player is presented with choices throughout the game and then the game responds based on the user's choice.

Write the algorithm for a simple choose your own adventure game where the user has three choices:

- a. Fight the dragon
- b. Go home
- c. Save the princess

The game should repeatedly ask the user which of the three options they want to do until the user says "Go home". When "Go home" is selected, the loop should exit, which effectively ends the game.

If the user selects "Fight the dragon", the algorithm should output "You win!". If "Save the princess" is selected, the algorithm should output "You saved the princess". If "Go home" is selected, the algorithm should output "Wimp".

You can set up your algorithm to check for the user's input in any way you like. Checking for the actual words, such as "Go home" is one option. If you want to assign a number to each option and check for the number, that also works.

7. Write an algorithm for playing a robotic version of the "treasure hunt" game. The "treasure hunt" game involves players receiving clues that guide them to other clues, until eventually the last clue leads them to a "treasure." For example, at the beginning of the game the players might receive a slip of paper that says "go to the big oak tree," at the oak tree they might find another slip of paper that says "try swimming," so they go to the swimming pool to look for a third clue, and so forth.

Robots can't read slips of paper, so the "clues" for a robot treasure hunt are represented by the colors of tiles. Furthermore, robots aren't smart enough to think of (or find) oak trees or swimming pools, so every clue in a robot treasure hunt just requires a robot to move one tile in some direction. After doing so, the robot examines the color of the tile it is then standing on to figure out where to go next, and so forth.

Here are the specific rules for interpreting tile colors as clues:

- ☐ White tile means that the next clue is the tile directly in front of the robot
- ☐ Blue tile means that the next clue is the tile to the robot's left
- ☐ Green tile means that the next clue is the tile to the robot's right
- ☐ Black tile means that the robot should move two tiles backward, then continue interpreting clues based on its new heading
- ☐ Yellow tile is a treasure — it marks the end of a part of a treasure hunt

Rules for your algorithm:

- ☐ Robot can only move forward, turn left, or turn right.
- ☐ The robot can detect the color of the position it is current on.

8. **Challenge exercise** (not required)

Once you get the algorithm working for single game, add in another tile:

- ☐ **Red** tile means that the treasure hunt splits into two parts — the next clue for the first part is the tile to the robot's left, and the next clue for the second part is the tile to the robot's right