San José State University
Department of Computer Science

CS 144
Advanced C++ Programming

Spring 2019
Instructor: Ron Mak

Assignment #1

Assigned:	Tuesday, January 29
Due:	Tuesday, February 5 at 8:30 AM
CodeCheck:	http://codecheck.it/files/1901290548du16pr08p345kq3g6lhnwsx2g **Note:** Use Firefox or Chrome, not Safari.
Canvas:	Assignment 1. Watering Plans
Points:	100

Watering plans

This assignment will give you practice using control statements, including nested loops, to implement some programming logic. Your program will simulate two different watering plans for your garden and determine which plan is better.

At the above CodeCheck URL, complete the program **WateringPlans.cpp** in CodeCheck's edit box, and then press the "Submit" button. CodeCheck will compile and run your program and compare your output to a master report. You can type into CodeCheck's edit box directly, or you can first edit and test your program in an IDE such as Eclipse, and then cut and paste it into CodeCheck.

Choose descriptive variable names. Include meaningful comments in your code, but don't over-comment. Include your name in a comment at the top of your program.

See the tutorials at http://www.cs.sjsu.edu/~mak/tutorials/index.html for how to install and configure Eclipse for C++ development, and, if necessary, VirtualBox and Ubuntu.

Academic integrity

You may study together and discuss the assignments, but what you turn in must be your individual work. Assignment submissions will be checked for plagiarism using Moss (http://theory.stanford.edu/~aiken/moss/). **Copying another student's program or sharing your program is a violation of academic integrity.** Moss is not fooled by renaming variables, reformatting source code, or re-ordering functions.

Violators of academic integrity will suffer severe sanctions, including academic probation. Students who are on academic probation are not

How to water your garden
You have a simple garden with *N* plants in a straight row, and a faucet at one end:

Which watering plan does this illustrate?

At the faucet, you fill a watering can completely with water and then walk to water your plants one at a time. When the can becomes empty, you walk back to the faucet to refill it. After you've watered the last plant, you walk back to the faucet with an empty or partially filled can. Leaving the faucet, watering plants, and returning to the faucet constitute one trip.

To keep things simple, assume that the watering can holds three units of water, and each plant requires one unit of water. Each plant is one step away from the next plant, and the faucet is one step from the first plant. The plant next to the faucet is labeled #1, the next plant is labeled #2, etc.

You have two different watering plans to accomplish each trip.

Watering Plan Near: After filling the watering can at the faucet, you always walk to the nearest unwatered plant and proceed to water each unwatered plant as you walk away from the faucet. When the can becomes empty, you walk back to the faucet with the empty can to refill it completely for another trip. After you've watered the last unwatered plant, you walk back to the faucet with the can, which can be empty or partially filled.

For example, you have five plants. On the first trip, you walk to plant #1 and water it, you water #2 and #3, and then you walk back to the faucet with an empty can. On your second trip, you walk to plant #4 and water it and #5, and then walk back to the faucet with one unit of water remaining in the can.

Watering Plan Far: After filling the watering can at the faucet, you always walk to the farthest unwatered plant and proceed to water each unwatered plant as you walk back towards the faucet. When the can becomes empty, you walk back to the faucet with the empty can to refill it completely for another trip. After you've watered the last unwatered plant, you walk back to the faucet with the can, which can be empty or partially filled.

For example, you have five plants. On the first trip, you walk to plant #5 and water it, you water #4 and #3, and then you walk back to the faucet with an empty can. On your second trip, you walk to plant #2 and water it and #1, and then you walk back to the faucet with one unit of water remaining in the can.

In each watering plan, you always fill the can completely before each trip.

Which plan is better? For each plan, you can count the total number of steps required to water all the plants. But a better measure is the total amount of weight you had to carry, measured in “step-units”. If you walk 3 steps to a plant with a full can (3 units), that’s 9 step units. Water the plant (now the can contains 2 units) and step to the next plant, which adds 2 step-units, for a cumulative total of 11 step-units. Walking with an empty can adds 0 step-units. Which watering plan results in fewer total step-units?

Before you write the program to give you some answers, which plan does your intuition tell you is better?

The simulations

Write a C++ program that does a series of simulations. Each simulation involves *N* plants, where *N* is an integer greater than 0 read from an input file. The input file contains several different values for *N* separated by spaces, and the last value is 0 as the end-of-data sentinel. As mentioned above, your program should assume the watering can holds three units of water, each plant requires one unit, and there is one step from the faucet to plant #1 and one step from one plant to the next. Always fill the can completely at the start of each trip.

For each value of *N* that your program reads, it should first simulate Watering Plan Near and then Watering Plan Far. During each simulation, your program should print which plant is being watered, how many steps have accumulated up to that point, how much water is in the can, and how many step-units have accumulated. Show walks back to the faucet.

After simulating both watering plans for a given value of *N*, your program should print which plan was better based on the total number of step-units.

Your program should not require any functions other than the main function. We’ll discuss programmer-defined functions later.

What do your simulations tell you? Can you explain the results? Is there a pattern?

Sample input file `counts.txt`

```
5 6 7 0
```

Expected output for the sample input file

```
==========================
 Plan Near with 5 plants
==========================

Where      Cum. steps  Water amt.  Cum. step-units
---------  ----------  ----------  ---------------
Plant  1       1          3              3
Plant  2       2          2              5
Plant  3       3          1              6
FAUCET         6          0              6
Plant  4      10          3             18
Plant  5      11          2             20
FAUCET        16          1             25

Plan Near: Total steps = 16, total step-units = 25

=========================
 Plan Far with 5 plants
=========================

Where      Cum. steps  Water amt.  Cum. step-units
---------  ----------  ----------  ---------------
Plant  5       5          3             15
Plant  4       6          2             17
Plant  3       7          1             18
FAUCET        10          0             18
Plant  2      12          3             24
Plant  1      13          2             26
FAUCET        14          1             27

Plan Far: Total steps = 14, total step-units = 27

*** With 5 plants, Plan Near is better with 2 fewer step-units.

==========================
 Plan Near with 6 plants
==========================

etc.
```

Submission into Canvas
When you're satisfied with your program in CodeCheck, click the "Download" link at the very bottom of the Report screen to download a signed zip file of your solution. Submit this zip file into Canvas. You can submit as many times as you want until the deadline, and the number of submissions will not affect your score. Only your last submission will be graded.

Note: Input file `counts.txt` has already been uploaded into CodeCheck.

Submit the signed zip file from CodeCheck into Canvas:
Assignment 1. Watering Plans.

Note: You must submit the signed zip file that you download from CodeCheck, or your submission will not be graded. Do not rename the zip file.

Rubric
Your program will be graded according to these criteria:

Criteria	Maximum points
Good output (as determined by CodeCheck) • Correct output values. • Correct output format.	40 • 20 • 20
Good program design • Good use of control statements. • Good use of nested loops.	30 • 20 • 10
Good program style • Descriptive variable names. • Meaningful comments.	30 • 15 • 15