Design While Loop

Hui Chen

Department of Computer & Information Science

Brooklyn College

Objectives

- To follow the loop design strategy to develop loops (§5.4).
- To control a loop with a sentinel value (§5.5).
- To obtain large input from a file using input redirection rather than typing from the keyboard (§5.5).

Outline

- Discussed
  - Loops and While loops
- Design while loops
  - Design strategy
  - Controlling loop
    - User confirmation
    - Sentinel value
  - Algorithm
    - Compute the sum of multiple numbers
  - Operating systems trick
    - Using input and output redirection to test your loops quickly

Design While Loop

- General strategy
  1. Identify the statements that need to be repeated
  2. Wrap these statements in a loop

```
while (true) {
        Statements-needed-to-be-repeated
}
```

  3. Design “loop-continuation-condition” and “loop-control-statement”

```
while (loop-continuation-condition) {
        Statements-needed-to-be-repeated
        loop-control-statement
}
```

Problem. Guess Numbers

- Write a program that randomly generates an integer between 0 and 100, inclusive. The program prompts the user to enter a number continuously until the number matches the randomly generated number. For each user input, the program tells the user whether the input is too low or too high, so the user can choose the next input intelligently.

Problem. MathTest

- Consider to write a program to generate a test to test students' ability to do single digit addition and subtraction. In the program, we generate N questions repeatedly. For instance, we can generate 5 questions randomly, prompt the user with the 5 questions, collection the student's answers, and report the score.

Questions?

Ending a Loop

- What if the number of times a loop is executed is not predetermined?
  - User control
  - Sentinel value

Let the user decide

- Prompt the user whether the user wishes to continue

Using Sentinel Value

- Use an input value to signify the end of the loop.
  - Such a value is known as a *sentinel value*.

Problem. Compute the sum

- Algorithm
  - Compute the sum of multiple numbers
- Write a program that reads and calculates the sum of an unspecified number of integers.
- Strategy 1
  - User control. Prompt the user whether to continue to enter next number
- Strategy 2
  - Using a sentinel value. The input 0 signifies the end of the input.

Questions?

Problem. How about compute sum of more than 100 numbers?

- It is tedious to test our programs. What to do?

Using Input Redirection

- Modern operating systems support input redirection
  - Instead of repeatedly entering the input, save the input in a file, and redirect the file as the input to the program
  - Example
    - java ComputeSum < input.txt
  - where the content of input.txt should match the expectation of the program
    - What does this mean? Compare two implementation strategies
      1. line = scanner.nextLine(); n = Integer.parseInt(line);
      2. n = scanner.nextInt();

Problem. What if we wish to save the output of the program?

- We can copy and paste the output of the program, but is there any other way?

Using Output Redirection

- Modern operating systems support input redirection
  - We redirect the standard output of the program to a file
  - Example
    - java HelloWorld > output.txt

Using Both Input and Output Redirection

- We can use both input and output redirection
- Example
  - java ComputeSum < input.txt > output.txt

Questions

Pitfalls and Errors

- Don’t use floating-point values for equality checking in a loop control.
- Since floating-point values are approximations for some values, using them could result in imprecise counter values and inaccurate results.
- Example. Consider the following code for computing 1 + 0.9 + 0.8 + ... + 0.1, what is the result?

Will the loop end?

```
double item = 1; double sum = 0;
while (item != 0) {
  sum += item;
  item -= 0.1;
}
System.out.println(sum);
```

Questions?