Indefinite Loops
and Boolean Expressions

Computer Science S-111
Harvard University
David G. Sullivan, Ph.D.

Review: Definite Loops

- The loops that we’ve seen thus far have been *definite loops*.
  - we know exactly how many iterations will be performed before the loop even begins

- In an *indefinite loop*, the number of iterations is either:
  - not as obvious
  - impossible to determine before the loop begins
Sample Problem: Finding Multiples

- Problem: Print all multiples of a number (call it num) that are less than 100.
  - output for num = 9:
    9 18 27 36 45 54 63 72 81 90 99

- Pseudocode for one possible algorithm:
  ```
mult = num
repeat as long as mult < 100:
  print mult + "  
  mult = mult + num
print a newline
```

Sample Problem: Finding Multiples (cont.)

- Pseudocode:
  ```
mult = num
repeat as long as mult < 100:
  print mult + "  
  mult = mult + num
print a newline
```

- Here's how we would write this in Java:
  ```java
  int mult = num;
  while (mult < 100) {
    System.out.print(mult + "  
    mult = mult + num;
  }
  System.out.println();
  ```
**while Loops**

- In general, a while loop has the form
  
  ```java
  while (<test>) {
    <one or more statements>
  }
  ```

- As with for loops, the statements in the block of a while loop are known as the *body* of the loop.

---

**Evaluating a while Loop**

**Steps:**
1. evaluate the test
2. if it's false, skip the statements in the body
3. if it's true, execute the statements in the body, and go back to step 1
Tracing a while Loop

• Let's trace through our code when num has the value 15:

```java
int mult = num;
while (mult < 100) {
    System.out.print(mult + " ");
    mult = mult + num;
}
```

<table>
<thead>
<tr>
<th>Output thus far</th>
<th>mult</th>
</tr>
</thead>
<tbody>
<tr>
<td>before entering the loop</td>
<td>15</td>
</tr>
<tr>
<td>after the first iteration</td>
<td>15 30</td>
</tr>
<tr>
<td>after the second iteration</td>
<td>15 30 45</td>
</tr>
<tr>
<td>after the third iteration</td>
<td>15 30 45 60</td>
</tr>
<tr>
<td>after the fourth iteration</td>
<td>15 30 45 60 75</td>
</tr>
<tr>
<td>after the fifth iteration</td>
<td>15 30 45 60 75 90</td>
</tr>
<tr>
<td>after the sixth iteration</td>
<td>15 30 45 60 75 90 105</td>
</tr>
</tbody>
</table>

and now (mult < 100) is false, so we exit the loop

Comparing if and while

• The true block of an if statement is evaluated at most once.
• The body of a while statement can be evaluated multiple times, provided the test remains true.
Typical while Loop Structure

• Typical structure:

\[
\begin{align*}
\text{initialization statement(s)} \\
\text{while (test) } & \{ \\
\text{other statements} \\
\text{update statement(s)} \\
\} \\
\end{align*}
\]

• In our example:

```java
int mult = num;                    // initialization
while (mult < 100) {
    System.out.print(mult + " ");
    mult = mult + num;              // update
}
```

Comparing for and while loops

• while loop (typical structure):

\[
\begin{align*}
\text{initialization} \\
\text{while (test) } & \{ \\
\text{other statements} \\
\text{update} \\
\} \\
\end{align*}
\]

• for loop:

```java
for (initialization; test; update) {
    one or more statements
}
```
Infinite Loops

• Let's say that we change the condition for our while loop:

```java
int mult = num;
while (mult != 100) {    // replaced < with !=
    System.out.print(mult + " ");
    mult = mult + num;
}
```

• When `num` is 15, the condition will always be true.
  • why?

    • an infinite loop – the program will hang (or repeatedly output something), and needs to be stopped manually
    • what class of error is this (syntax or logic)?

• It's generally better to use `<, <=, >, >=` in a loop condition, rather than `==` or `!=`

Infinite Loops (cont.)

• Another common source of infinite loops is forgetting the update statement:

```java
int mult = num;
while (mult < 100) {
    System.out.print(mult + " ");
    // update should go here
}
```
A Need for Error-Checking

• Let's return to our original version:
  
  ```java
  int mult = num;
  while (mult < 100) {
      System.out.print(mult + " ");
      mult = mult + num;
  }
  ```

  • This could still end up in an infinite loop! How?

Using a Loop When Error-Checking

• We need to check that the user enters a positive integer.

• If the number is <= 0, ask the user to try again.

• Here's one way of doing it using a while loop:
  
  ```java
  Scanner console = new Scanner(System.in);
  System.out.print("Enter a positive integer: ");
  int num = console.nextInt();
  while (num <= 0) {
      System.out.print("Enter a positive integer: ");
      num = console.nextInt();
  }
  ```

  • Note that we end up duplicating code.
Error-Checking Using a do-while Loop

- Java has a second type of loop statement that allows us to eliminate the duplicated code in this case:
  ```java
  Scanner console = new Scanner(System.in);
  int num;
  do {
    System.out.print("Enter a positive integer: ");
    num = console.nextInt();
  } while (num <= 0);
  ```

- The code in the body of a do-while loop is always executed at least once.

do-while Loops

- In general, a do-while statement has the form
  ```java
  do {
    <one or more statements>
  } while (<test>);
  ```

- Note the need for a semi-colon after the condition.
- We do not need a semi-colon after the condition in a while loop.
  - beware of using one – it can actually create an infinite loop!
Evaluating a do-while Loop

Steps:
1. execute the statements in the body
2. evaluate the test
3. if it's true, go back to step 1
   (if it's false, continue to the next statement)

Formulating Loop Conditions

• We often need to repeat actions until a condition is met.
  • example: keep reading a value until the value is positive
  • such conditions are termination conditions – they indicate when the repetition should stop

• However, loops in Java repeat actions while a condition is met.
  • they use continuation conditions

• As a result, you may need to convert a termination condition into a continuation condition.
Which Type of Loop Should You Use?

• Use a for loop when the number of repetitions is known in advance – i.e., for a definite loop.

• Otherwise, use a while loop or do-while loop:
  • use a while loop if the body of the loop may not be executed at all
    • i.e., if the condition may be false at the start of the loop
  • use a do-while loop if:
    • the body will always be executed at least once
    • doing so will allow you to avoid duplicating code

Find the Error…

• Where is the syntax error below?
  Scanner console = new Scanner(System.in);
  do {
    System.out.print("Enter a positive integer: ");
    int num = console.nextInt();
  } while (num <= 0);
  System.out.println("The multiples of " + num + " less than 100 are:");
  int mult = num;
  while (mult < 100) {
    System.out.print(mult + " ");
    mult = mult + num;
  }
  System.out.println();
Practice with while loops

• What does the following loop output?

```java
int a = 10;
while (a > 2) {
    a = a - 2;
    System.out.println(a * 2);
}
```

<table>
<thead>
<tr>
<th>a &gt; 2</th>
<th>a</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>false</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>false</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>false</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>false</td>
<td>-4</td>
<td></td>
</tr>
</tbody>
</table>

boolean Data Type

• A condition like `mult < 100` has one of two values: `true` or `false`

• In Java, these two values are represented using the `boolean` data type.
  • one of the primitive data types (like `int`, `double`, and `char`)
  • `true` and `false` are its two literal values

• This type is named after the 19th-century mathematician George Boole, who developed the system of logic called `boolean algebra`. 
boolean Expressions

- We have seen a number of constructs that use a "test".
  - loops
  - if statements

- A more precise term for a "test" is a boolean expression.

- A boolean expression is any expression that evaluates to true or false.
  - examples:
    - num > 0
    - false
    - firstChar == 'P'
    - score != 20

boolean Expressions (cont.)

- Recall this line from our ticket-price program:
  
  ```java
  if (choice.equals("orchestra")) ...
  ```

  a boolean expression, because it evaluates to true or false

- if we look at the String class in the Java API, we see that the equals method has this header:

  ```java
  public boolean equals(...) 
  ```

  it returns either true or false
Forming More Complex Conditions

- We often need to make a decision based on more than one condition – or based on the opposite of a condition.
- examples in pseudocode:
  - if the number is even AND it is greater than 100…
  - if it is NOT the case that your grade is > 80…

- Java provides three **logical operators** for this purpose:

<table>
<thead>
<tr>
<th>operator</th>
<th>name</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;&amp;</td>
<td>and</td>
<td>age &gt;= 18 &amp;&amp; age &lt;= 35</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>not</td>
<td>!(grade &gt; 80)</td>
</tr>
</tbody>
</table>

Truth Tables

- The logical operators operate on boolean expressions.
- let \( a \) and \( b \) represent two such expressions

- We can define the logical operators using **truth tables**.

<table>
<thead>
<tr>
<th>( a )</th>
<th>( b )</th>
<th>( a &amp;&amp; b )</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( a )</th>
<th>( b )</th>
<th>( a || b )</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( a )</th>
<th>( \neg a )</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
</tr>
</tbody>
</table>
Truth Tables (cont.)

- Example: evaluate the following expression:
  \[(20 \geq 0) \land (30 \% 4 == 1)\]

- First, evaluate each of the operands:
  \[(20 \geq 0) \land (30 \% 4 == 1)\]
  \[true \land false\]

- Then, consult the appropriate row of the truth table:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a \land b</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

- Thus, \((20 \geq 0) \land (30 \% 4 == 1)\) evaluates to false

Practice with Boolean Expressions

- Let's say that we wanted to express the following English condition in Java:
  "num is not equal to either 0 or 1"

- Which of the following boolean expression(s) would work?
  a) \(num \neq 0 \lor 1\)
  b) \(num \neq 0 \lor num \neq 1\)
  c) \(! (num == 0 \lor num == 1)\)

- Is there a different boolean expression that would work here?
boolean Variables

• We can declare variables of type boolean, and assign the values of boolean expressions to them:

```java
int num = 10;
boolean isPos = (num > 0);
boolean isDone = false;
```

• these statements give us the following picture in memory:

<table>
<thead>
<tr>
<th>isPos</th>
<th>true</th>
</tr>
</thead>
<tbody>
<tr>
<td>isDone</td>
<td>false</td>
</tr>
</tbody>
</table>

• Using a boolean variable can make your code more readable:

```java
if (value % 2 == 0) {
...
}

boolean isEven = (value % 2 == 0);
if (isEven == true) {
...
}
```

boolean Variables (cont.)

• Instead of doing this:

```java
boolean isEven = (num % 2 == 0);
if (isEven == true) {
...
}
```

you could just do this:

```java
boolean isEven = (num % 2 == 0);
if (isEven) {
...
}
```

The extra comparison isn't necessary!

• Similarly, instead of writing:

```java
if (isEven == false) {
...
}
```

you could just write this:

```java
if (!isEven) {
...
}
```
Input Using a Sentinel

• Example problem: averaging an arbitrary number of grades.

• Instead of having the user tell us the number of grades in advance, we can let the user indicate that there are no more grades by entering a special sentinel value.

• When we encounter the sentinel, we break out of the loop
  • example interaction:
    Enter grade (-1 to end): 10
    Enter grade (-1 to end): 8
    Enter grade (-1 to end): 9
    Enter grade (-1 to end): 5
    Enter grade (-1 to end): -1
    The average is: 8.0

Input Using a Sentinel (cont.)

• Here’s one way to do this:
    Scanner console = new Scanner(System.in);
    int total = 0;
    int numGrades = 0;

    System.out.print("Enter grade (or -1 to quit): ");
    int grade = console.nextInt();
    while (grade != -1) {
        total += grade;
        numGrades++;
        System.out.print("Enter grade (or -1 to quit): ");
        grade = console.nextInt();
    }

    if (numGrades > 0) {
        System.out.print("The average is ");
        System.out.println((double)total/numGrades);
    }
Input Using a Sentinel and a Boolean Flag

• Here's another way, using what is known as a boolean flag, which is a variable that keeps track of some condition:

```java
Scanner console = new Scanner(System.in);
int total = 0;
int numGrades = 0;
boolean done = false;

while (!done) {
    System.out.print("Enter grade (or -1 to quit): ");
    int grade = console.nextInt();
    if (grade == -1) {
        done = true;
    } else {
        total += grade;
        numGrades++;
    }
}
if (numGrades > 0) {
    ...
```

Input Using a Sentinel and a break Statement

• Here's another way, using what is known as a break statement, which "breaks out" of the loop:

```java
Scanner console = new Scanner(System.in);
int total = 0;
int numGrades = 0;

while (true) {
    System.out.print("Enter grade (or -1 to quit): ");
    int grade = console.nextInt();
    if (grade == -1) {
        break;
    } else {
        total += grade;
        numGrades++;
    }
}
// after the break statement, the flow of control resumes here...
if (numGrades > 0) {
    ...
```