Computer Science AP

Control Statements Continued

Chapter 7

Review

- Logical Operators – note their precedence

  `!` `&&` `||`

- boolean variables

```
boolean passing = grade >= 65;
```

- Short-Circuit Evaluation

```
if (x > 5 && y != 35)
if (a == 3 || b > 4)
```

Boolean Equivalences

```
!(p || q)       ->   !p && !q

!(p && q)       ->   !p || !q

p || (q && r)   ->   (p || q) && (p || r)

p && ( q || r) ->   (p && q) || (p && r)
```

Robust

- A program that produces correct results when it has valid inputs is not good enough.
- Must test using invalid data.
- A program that tolerates and recovers from errors in input is robust.

Testing `if` Statements

- Quality Assurance — The process of making sure a software product is developed to the highest standard possible
- Complete code coverage — Providing test data that executes EVERY line of the program
- Equivalence class — Sets of test data that follow the same lines of code
- Boundary conditions — Test data that is on or near the boundaries of equivalence classes
- Extreme conditions — Choosing test data that is AT the limits of validity (ie: 7 X 24 = 168 for hours worked)

Case Study 1 – page 236

- Goal - create a payroll system
- Type 1 - employees are full time
  - for hours worked <= 40, pay is at a base rate
  - for hours worked > 40, pay is twice base rate
  - Base rate is between $6.75 and $30.50
- Type 2 - employees are part time
  - pay is always at base rate
  - hours worked are between 1 and 60

Test Data

- Equivalence class – test data that follows the same code
  - Hours = 29, 30 or 31 OR Hours = 49, 50 or 51
- Boundary conditions – test data on or near boundaries of the equivalence class
  - Hours = 39, 40 or 41
- Extreme conditions – test data at the limit of validity
  - Hours = 0 or 168 . What is the significance of 168?
- Complete code coverage – test data that runs through each line of code at least once. For this case study it is easy.

Nested `if` Statements

- Use of braces
  - Better to overuse than to underuse
  - Use braces and indentation for clarity and readability

Example 1 – Nested `if` Statements

```
if (t > 80)
      S.O.P (“It is hot”);
else
      if (t > 60)
            S.O.P. (“It is nice”);
      else
            if (t > 30)
                  S.O.P. (“It is cool”);
            else
                  S.O.P. (“It’s COLD!”);
```

Example 2 – Nested `if` Statement

```
if (t > 80)
      S.O.P (“It is hot”);
else if (t > 60)
      S.O.P. (“It is nice”);
else if (t > 30)
      S.O.P. (“It is cool”);
else
      S.O.P. (“It’s COLD!”);
```

Misplaced Braces

```
//version 1
if ( isRaining() ) {
      if ( haveUmbrella() )
            walk();
      else
            run();
}
```

Misplaced Braces

```
//version 2
if ( isRaining() ) {
      if ( haveUmbrella() )
            walk();
}
else
      run();
```

No Braces

```
//version 3
if ( isRaining() )
      if ( haveUmbrella() )
            walk();
      else
            run();
```

No Braces

```
//version 4
if ( isRaining() )
      if ( haveUmbrella() )
            openUmbrella();
            walk();
      else
            run();
```

With Braces

```
//version 5
if ( isRaining() )
      if ( haveUmbrella() ) {
            openUmbrella();
            walk();
      }
      else
            run();
```

With More Braces

```
//version 6
if ( isRaining() ) {
      if ( haveUmbrella() ) {
            openUmbrella();
            walk();
      }
      else {
            run();
      }
}
```

Which nested `if` is correct?

- 10% commission if sales greater than or equal to $5000
- 20% commission if sales greater than or equal to $10,000

```
if (sales >= 5000)
  C = sales * 0.1;
else if (sales >= 10000)
  C = sales * 0.2;
```

```
if (sales >= 10000)
  C = sales * 0.2;
else if (sales >= 5000)
  C = sales * 0.1;
```

Nested Loops

- A loop within a loop
- Inner loop must complete before continuing with the outer loop
- Example – What does this do?

```
for (int k = 1; k <= 3; k++)

      for (int j = 1; j <= 3; j++)

            S.O.P.(j + " ");
```

Your turn

- Output the numbers 1 to 25 in consecutive order, using five rows of five numbers each.

```
for (int j = __; j < __; j++) {
    for (int k = __; k < __; k++) {
        System.out.print( j*5 + k + "   ");
    }
    System.out.println ();
}
```

Combinatorial Explosion

- The amount of data needed to test a program increases, as the complexity of the program increases.
- It is unlikely that parts of a program act independently of each other
- We call this multiplicative growth in test cases a combinatorial explosion, and it pretty much guarantees the impossibility of exhaustively testing large complex programs; however, programmers still must do their best to test their programs intelligently and well.

Case Study 2 Fibonacci Numbers

- User input should e a positive integer or -1 to quit
- All other inputs are rejected
- Example
  - `Enter positive integer or –1 to quit:  8`
  - `Fibonacci number 8 is 21`
- Page 257 – program looks correct
- Unexpected problem occurs for input equal to 80
- http://www.nytimes.com/1996/12/01/magazine/little-bug-big-bang.html
- https://www.wired.com/2005/11/historys-worst-software-bugs/

The `assert` Statement

- A statement that contains a boolean expression and halts the program when the expression is false
- It is used only during testing because it slows down the JVM

```
assert x >= 0 && x <= MAX;
```

Assertions with loops

- **Comments**
- **Input assertions:** state what should be true before a loop is entered.
- **Output assertions:** state what should be true when the loop is exited.

Example - Assertions with loops

- Task: Sum the proper divisors of num

```
divisorSum = 0;
for (testDiv = 1; testDiv <= num/2; testDiv++)
    if (num % testDiv == 0 )
        divisorSum += testDiv;
```

- Input assertion:
  - 1. num is a positive integer
  - 2. divisorSum == 0
- Output assertion:
  - divisorSum is the sum of all proper divisors of num

Example - Assertions with loops

```
divisorSum = 0;
assert num > 0 && divisorSum == 0;
for (testDiv = 1; testDiv <= num/2; testDiv++)
    if (num % testDiv == 0 )
        divisorSum += testDiv;
//Output assertion: divisorSum is the sum of all
//proper divisors of num
```

Invariant and Variant Assertions

- **Loop invariant:** an assertion that exposes a relationship between variables that remains constant throughout all loop iterations.
  - True before the loop is entered, and after each pass.
- **Loop variant:** an assertion whose truth changes between the first and final execution of the loop.
  - Guarantees the loop is exited.

```
divisorSum = 0;
// 1. num is a pos. integer.     (input assertion)
// 2. divisorSum == 0
assert num > 0 && divisorSum == 0;
for (testDiv = 1; testDiv <= num/2; testDiv++)
//                                    (variant assertion)
// testDiv is incremented by 1 each
// time through the loop.  It eventually
// exceeds the value (num/2), at which
// point the loop is exited
   if (num % testDiv == 0)
      divisorSum += testDiv;
//                                  (invariant assertion)
//divisorSum is the sum of proper divisors
//of num that are less than or equal to testDiv

//                                   (output assertion)
// divisorSum is the sum of
// all proper divisors of num
```

Advanced Operations on Strings

- `int indexOf (char)`
- `int indexOf (char, int)`
- `int indexOf (String)`
- `int indexOf (String, int)`
- `int length()`
- `String substring (int)`
- `String substring (int, int)`

More String operations

- `String toLowerCase()`
- `String toUpperCase()`
- `String trim()`
- Where are the setters (or mutators)??

  String objects are immutable.
  Once a string is created, its length cannot change and its characters cannot be modified.

Your Turn

1. What operator returns true if and only if both of its operands are true?
2. What operator returns false if and only if both of its operands are false?
3. Write an if statement that displays whether or not a given number, `x`, is between a lower bound `min` and an upper bound `max`, inclusive. Use a logical operator.
4. Rewrite the if statement in #3 using a nested if statement

Consider the following code segment.

```
int k = a random number such that 1 ≤ k ≤ n

for (int p = 2; p <= k; p++)
   for (int r = 1; r < k; r++)
  System.out.println ("Hello");
```

What is the minimum number of times "Hello" will be printed?

A) 0
B) 1
C) 2
D) n-1
E) n-2

What is the maximum number of times "Hello" will be printed?

A) 2
B) n-1
C) n-2
D) $(n-1)^2$
E) n^2

Consider the following method.

```
public void numberCheck (int maxNum)
{
   int typeA = 0;
   int typeB = 0;
   int typeC = 0;

   for (int k = 1; k <= maxNum; k++)
   {
  if (k % 2 == 0 && k % 5 == 0)
         typeA++;
      if (k % 2 == 0)
         typeB++;
      if (k % 5 == 0)
         typeC++;
   }
  System.out.println (typeA + " " + typeB + " " + typeC);
 }
```

What is printed as a result of the call `numberCheck(50)` ?

A) 5 20 5
B) 5 20 10
C) 5 25 5
D) 5 25 10
E) 30 25 10