Algorithms

An algorithm is a specific set of instructions for carrying out a procedure or solving a problem, usually with the requirement that the procedure terminate at some point. Specific algorithms sometimes also go by the name method, procedure, or technique. The word "algorithm" is a distortion of al-Khwārizmī, a Persian mathematician who wrote an influential treatise about algebraic methods.

Sources: [MathWorld](http://mathworld.wolfram.com/Algorithm.html) and [Wikipedia](http://en.wikipedia.org/wiki/Muhammad_ibn_Musa_al-Khwarizmi)

Algorithm Example: **Input-Compute-Output** pattern

**GPA problem:** Write a program that computes and outputs the GPA, given the credits and quality points earned.

**Variables:** qp, credits, gpa

**Algorithm:**
1. Input qp
2. Input credits
3. gpa = qp / credits
4. Print gpa
Java Program

```java
import java.util.Scanner;
public class GPA {
    public static void main(String[] args) {
        double qp, credits, gpa;
        Scanner scan = new Scanner(System.in);
        System.out.print("Enter Quality Points > ");
        qp = scan.nextInt();
        System.out.print("Enter Credits > ");
        credits = scan.nextInt();
        gpa = qp / credits;
        System.out.println("GPA: "+ gpa);
    }
}
```

Interactive Programs – Input/Output

- In Java, you first need to create a Scanner object

```java
int age;
String name;
Scanner scan = new Scanner(System.in);
System.out.print("Enter your name");
name = scan.nextLine();
System.out.print("Enter your age");
age = scan.nextInt();
```

Reading Input

- The Scanner class is part of the java.util class library, and must be imported into a program in order to be used
- The import statement goes at beginning of your program (above class definition)

```java
import java.util.Scanner;
```

Using the Scanner class

1. import the class, i.e., add this before the class definition of your program:

```java
import java.util.Scanner;
```

2. In your main method, before doing any input, declare and initialize the Scanner object

```java
Scanner scan = new Scanner(System.in);
```

3. Input away!

```java
System.out.print("Enter your name");
name = scan.nextLine();
System.out.print("Enter your age");
age = scan.nextInt();
```
import java.util.Scanner;

public class TellMeAboutYou {
    public static void main(String[] args) {
        int age;
        String name;
        Scanner scan = new Scanner(System.in);
        System.out.print("Enter your name");
        name = scan.nextLine();
        System.out.print("Enter your age");
        age = scan.nextInt();
        System.out.println("Pleased to meet you, "+ name + ");
        System.out.println("Your age in dog years: "+ age*10.5);
    }
}

Example
Inspired by: http://www.onlineconversion.com/dogyears.htm

Enter your name: Fiona
Enter your age: 17
Pleased to meet you, Fiona!
Your age in dog years is 178.5

Input methods
• nextInt() → input an int
• nextDouble() → input a double
• nextLine() → input a String (until end of line)
• next() → input a String token (one word or other delimited “chunk” of text)

White space (space, tab, new line) are used to separate input tokens

Control flow
Sequence of statements that are actually executed in a program

Conditional and Repetition statements: enable us to alter control flow

Java Conditional statements alter the linear flow of control. They use boolean expressions to determine what to do next.

Example:
if (credits == 0)
    System.out.println ("GPA: None");
else
    
gpa = qp / credits;
    System.out.println ("\n\tGPA: "+ gpa);
Java relational operators

- relational operators can be used with numeric types and produce boolean results:
  
  ```
  ==  equal to
  !=  not equal to
  <   less than
  >   greater than
  <=  less than or equal to
  >=  greater than or equal to
  ```

- Note the difference between the equality operator (==) and the assignment operator (=)

Example:

How do we fix output to use singular/plural as appropriate?

For example:

Enter the total amount to be given as change: 18
That amount can be given as:
0 quarters
1 dimes
1 nickels
3 pennies

Nested loops

```java
statement 1;
if (condition 1)
    statement 2;
else
    if (condition 2)
        statement 4;
    else
        statement 3;
else
    statement 5;
```
Another example:

Create an application called Vacation that prompts for and inputs an integer representing someone’s age and then suggests an appropriate vacation destination. One of three destinations should be suggested depending on whether the answer is less than 20, between 20 and 50, or over 50.

For example, a run of the program might look like this:

```
How old is the traveler > 15
You should consider Hershey Park.
```

Java Logical Operators

- logical operators can be used with boolean operands to express more complex boolean conditions:

  ```
  ! Logical NOT
  && Logical AND
  || Logical OR
  ```

Vacation example revisited:

Create an application called Vacation that prompts for and inputs an integer representing someone’s age and then suggests an appropriate vacation destination. One of three destinations should be suggested depending on whether the answer is less than 20, between 20 and 50, or over 50.

For example, a run of the program might look like this:

```
How old is the traveler > 15
You should consider Hershey Park.
```

Boolean Expressions

- The reserved words true and false are the only valid values for a boolean type

- Example: boolean variables:

  ```
  boolean aboveAgeLimit = false;
  boolean usePlural = hours > 1;
  ```
Example

```java
if (total < MAX+5 && !found)
    System.out.println("Processing.");
```

- All logical operators have lower precedence than the relational operators
- The `!` operator has higher precedence than `&&` and `||`

Logical NOT

- The logical NOT operation is also called logical negation or logical complement
- If some boolean condition `a` is true, then `!a` is false; if `a` is false, then `!a` is true
- Logical expressions can be shown using a truth table:

<table>
<thead>
<tr>
<th><code>a</code></th>
<th><code>!a</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>

Logical AND and Logical OR

- The logical AND expression
  ```java
  a && b
  ```
  is true if both `a` and `b` are true, and false otherwise
- The logical OR expression
  ```java
  a || b
  ```
  is true if `a` or `b` or both are true, and false otherwise

Logical AND and Logical OR

- A truth table shows all possible true-false combinations of the terms
- Since `&&` and `||` each have two operands, there are four possible combinations of conditions `a` and `b`:

| `a` | `b` | `a && b` | `a || b` |
|-----|-----|---------|---------|
| true| true| true    | true    |
| true| false| false   | true    |
| false| true| false   | true    |
| false| false| false   | false   |
Quick Check 1
What do the following statements do?

```java
if (total != stock + warehouse)
    inventoryError = true;

if (found || !done)
    System.out.println("Ok");
```

<table>
<thead>
<tr>
<th>total</th>
<th>stock</th>
<th>warehouse</th>
<th>inventoryError</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>8</td>
<td>12</td>
<td>false</td>
</tr>
<tr>
<td>found</td>
<td>false</td>
<td>done</td>
<td></td>
</tr>
</tbody>
</table>

Quick Check 2
Try again with different values

```java
if (total != stock + warehouse)
    inventoryError = true;

if (found || !done)
    System.out.println("Ok");
```

<table>
<thead>
<tr>
<th>total</th>
<th>stock</th>
<th>warehouse</th>
<th>inventoryError</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>7</td>
<td>12</td>
<td>false</td>
</tr>
<tr>
<td>found</td>
<td>false</td>
<td>done</td>
<td></td>
</tr>
</tbody>
</table>

Quick Check 3
Try again with different values

```java
if (total != stock + warehouse)
    inventoryError = true;

if (found || !done)
    System.out.println("Ok");
```

<table>
<thead>
<tr>
<th>total</th>
<th>stock</th>
<th>warehouse</th>
<th>inventoryError</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>8</td>
<td>12</td>
<td>true</td>
</tr>
<tr>
<td>found</td>
<td>true</td>
<td>done</td>
<td></td>
</tr>
</tbody>
</table>

Boolean Expressions
- using truth tables – let’s try this one:

| found | done | !done | found || !done |
|-------|------|-------|--------|----------|
| false | false|       | false  | false    |
| false | true |       | true   | true     |
| true  | false|       | false  | true     |
| true  | true |       | true   | true     |
Boolean Expressions

• using truth tables – another example:

<table>
<thead>
<tr>
<th>total &lt; MAX</th>
<th>found</th>
<th>!found</th>
<th>total &lt; MAX &amp;&amp; !found</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td></td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td></td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td></td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td></td>
<td>true</td>
</tr>
</tbody>
</table>

How much of a boolean expression do we need to evaluate before determining its value?

*** Short-Circuited Operators

• The processing of `&&` and `||` is “short-circuited” in cases where the left operand is sufficient to determine the result (the right operand is not evaluated at all)

• This can be both useful and dangerous!

```java
if (count != 0 && total/count > MAX)
    System.out.println("Testing.");
```

Indentation Revisited

• Remember that indentation is for the human reader, and is ignored by the computer

```java
if (total > MAX)
    System.out.println("Error!");
    errorCount = errorcount + 1;
```

Despite what is implied by the indentation, the increment will occur whether the condition is true or not

Selection structures in Java

• Conditional statement:

```java
if (n > 0)
    System.out.print("positive");
else
    System.out.print("negative");
```

• Other selection structures (Chapter 6 in text)
  – the `conditional` operator
  – the `switch` statement
The Conditional Operator Syntax

(condition ? expression1 : expression2)

- If the condition is true, expression1 is evaluated; if it is false, expression2 is evaluated
- The value of the entire conditional operator is the value of the selected expression
- Example: Rewrite this →

```java
if (n > 0)
    System.out.print("positive");
else
    System.out.print("negative");
```

More examples:

```java
int bit = (ans == 'Y')? 1: 0;
String status = (age < 18 ? "child" : "adult");
```

- The conditional operator requires three operands so it is sometimes called the ternary operator

Quick Check

```java
System.out.println("Your change is " +
    num + "Dimes");
```

Rewrite this statement so that "Dime" is printed if num equals 1.
Recall: Logic of an if-else statement

The switch Statement: multi-way branches

public String toString()
{
    String result = "";
    switch (faceValue)
    {
        case 1:
            result = "one";
            break;
        case 2:
            result = "two";
            break;
        case 3:
            result = "three";
            break;
        default:
            return result;
    }
    return result;
}
So... the logic of the switch is more like this:

```
expression evaluated

1 2 3

statements1
statements2
statements3
```

Note: this is still a simplified flowchart of the logic of the switch statement.

---

**GradeReport.java**

```java
import java.util.Scanner;

public class GradeReport {

    public static void main (String[] args) {

        int grade, category;
        Scanner scan = new Scanner (System.in);

        System.out.print("Enter a numeric grade (0 to 100): ");
        grade = scan.nextInt();

        category = grade / 10;

        System.out.print("That grade is ");

        continue

        switch (category) {
            case 10:
                System.out.println("a perfect score. Well done.");
                break;
            case 9:
                System.out.println("well above average. Excellent.");
                break;
            case 8:
                System.out.println("above average. Nice job.");
                break;
            case 7:
                System.out.println("average.");
                break;
            case 6:
                System.out.println("below average. You should see the instructor to clarify the material that is presented in class.");
                break;
            default:
                System.out.println("not passing.");
        }
    }
}
```

**Sample Run**

```
Enter a numeric grade (0 to 100): 91
That grade is well above average. Excellent.
```

**Hands on:** try removing one of the break statements.

---

**Control flow: Repetition**

- Sequence of statements that are actually executed in a program
- Conditional and Repetition statements: enable us to alter control flow
Example

• Investment problem: You put $10,000 into a bank account that earns 5% interest per year.

<table>
<thead>
<tr>
<th>year</th>
<th>interest</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>$10,000.00</td>
</tr>
<tr>
<td>1</td>
<td>$500.00</td>
<td>$10,500.00</td>
</tr>
<tr>
<td>2</td>
<td>$525.00</td>
<td>$11,025.00</td>
</tr>
<tr>
<td>3</td>
<td>$551.25</td>
<td>$11,576.25</td>
</tr>
<tr>
<td>4</td>
<td>$578.81</td>
<td>$12,155.06</td>
</tr>
</tbody>
</table>

• ... How many years does it take for the account balance to be double the original?

This example is adapted from Cay Horstmann’s Big Java, 5th edition.

Example

• Investment problem: You put $10,000 into a bank account that earns 5% interest per year. How many years does it take for the account balance to be double the original?

• Algorithm:

The while Statement

• A while statement has the following syntax:

```java
while ( condition )
    statement;
```

• If the condition is true, the statement is executed
• Then the condition is evaluated again, and if it is still true, the statement is executed again
• The statement is executed repeatedly until the condition becomes false

Logic of a while Loop

[Diagram showing the logic of a while loop]
Example

- A counting loop that prints the numbers 1, 2, 3,…

Algorithm:

- initialize a counter to 1
- while the counter <= upper limit
  - print counter
  - increment counter

The while Statement

```java
int count = 1;
while (count <= 3)
{
    System.out.println(count);
    count++;
}
```

The while Statement “unraveled”

```
int count = 1;
while (count <= 3) { Test: (count <= 3) ➔ true
{
    System.out.println(count);
    count++;
    // count is now greater than 3
    // Break or continue outside of this loop
    break;
}
```

Example of squares and cubes:

<table>
<thead>
<tr>
<th>N</th>
<th>N^2</th>
<th>N^3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>27</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>64</td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td>125</td>
</tr>
</tbody>
</table>
Example
- **Table of powers:** Compute the powers of 2 and the powers of 3 and print a table like this:

<table>
<thead>
<tr>
<th>N</th>
<th>$2^N$</th>
<th>$3^N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>27</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>81</td>
</tr>
</tbody>
</table>

What’s wrong with this code?

```java
int count = 1;
while (count <= 10)
    System.out.println (count);
    count++;
```
What’s wrong with this code?

```java
int count = 1;
while (count <= 10)
{
    System.out.println (count);
    count++;
}
```

If the condition of a `while` loop is false initially, the statement is never executed.

```java
System.out.println("input a number >5");
int num = scan.nextInt();
// input validation
while (num <= 5)
{
    System.out.println ("type a number >5");
    num = scan.nextInt();
}
```

• Therefore, the body of a `while` loop will execute zero or more times.

Nested loops

*Example: Investment problem repetition*

- the repeated action (calculating the number of years it takes for investment to double) involves repetition
- General pattern for algorithms: A **nested loop**
  - while (condition for repeating action)
  - initialize variables
  - while (condition for reaching goal)
  - calculations
  - print results

Quick Check

How many times will the string "Here" be printed?

```java
count1 = 1;
while (count1 <= 10)
{
    count2 = 1;
    while (count2 < 20)
    {
        System.out.println ("Here");
        count2 = count2++;
    }
    count1 = count1++;
}
Introduction

Java Bootcamp - Villanova University

More repetition structures: do & for loops

Repetition structure in Java, so far: while loop

```java
int count = 0;
while (count < 5)
{
    System.out.println (count);
    count++;
}
```

- Other repetition structures (Chapter 6 in text)
  - the do loop
  - the for loop

The do Statement in Java

- A do statement has the following syntax:

```java
do
{
    statement-list;
} while (condition); //end do
```

- The statement-list is executed once initially, and then the condition is evaluated
- The statement-list is executed repeatedly until the condition becomes false

The while and do loops are similar.

```java
int count = 0;
while (count < 5)
{
    System.out.println (count);
    count++;
}
```

```java
int count = 0;
do {
    System.out.println (count);
    count++;
} while (count < 5);
```

Similar – but not the same:

- The body of a do loop executes at least once

Dr. Papalaskari
For some things the **do** loop is more appropriate:

```java
System.out.println("input a number >5");
int num = scan.nextInt();
while (num <= 5) {
    System.out.println("type a number >5");
    num = scan.nextInt();
}
```

---

**input validation**

**repeating a computation**

---

Copyright © 2012 Pearson Education, Inc.
The for Statement

- A for loop is functionally equivalent to the following while loop structure:

```
initialization;
while (condition)
{
    statement;
    increment;
}
```

- The for statement has the following syntax:

```
for (initialization; condition; increment)
statement;
```

  - The initialization is executed once before the loop begins.
  - The statement is executed until the condition becomes false.
  - The increment portion is executed at the end of each iteration.

- A for statement has the following syntax:

```
for (int count = 0; count < 5; count++)
    System.out.println (count);
```

  - The initialization is executed once before the loop begins.
  - The statement is executed until the condition becomes false.
  - The increment portion is executed at the end of each iteration.

- A for loop is well suited for executing statements a specific number of times that can be calculated or determined in advance.

```
for (int num=100; num > 0; num -= 5)
    System.out.println (num);
```

  - The increment section can perform any calculation.
Example:  

```java
//********************************************************************
// Stars.java       Author: Lewis/Loftus
// Demonstrates the use of nested for loops.
//********************************************************************
public class Stars
{
    // Prints a triangle shape using asterisk (star) characters.
    public static void main (String[] args)
    {
        final int MAX_ROWS = 10;
        for (int row = 1; row <= MAX_ROWS; row++)
        {
            for (int star = 1; star <= row; star++)
            {
                System.out.print (*");
            }
            System.out.println();
        }
    }
}

Output
*
**
***
****
*****
******
*******
********
*********
**********

Example: can you make it print the row number in (1, 2, 3… ) at the beginning of each line?
```