Intro to Programming

Laboratory 4

# Names \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# lab8b

# General Lab Procedures

* You should have a directory (folder) in your home account called csis120. At the beginning of each lab, create a new sub-directory called labX, where X is the lab number.
* Files used in the lab can be found on the course blackboard webpages.
* Turn in this lab stapled to print outs of the code you produce in each assigned section from the laboratory manual. These sheets must be in order.
* You can find documentation for the Java class libraries at the URL

<http://java.sun.com/javase/6/docs/api/>

## Lab 3.5

#### Objectives:

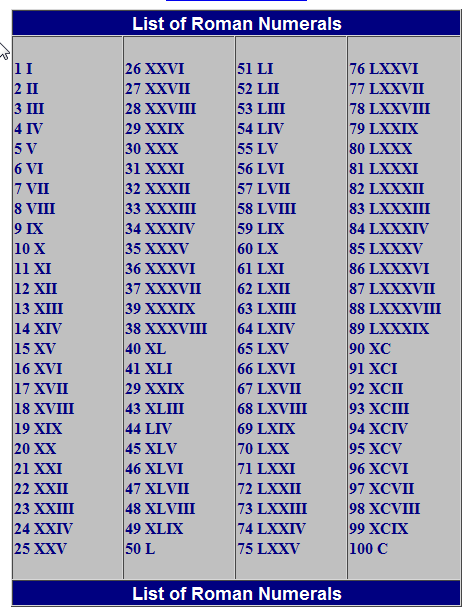
* To practice the use of conditional statements (if…else)
* To understand the effect of short circuit evaluation
* To understand the effects of algorithmic structure when multiple decisions are required

#### Part 1: Is the glass half empty or half full?

In this part of the lab, you will complete a class which converts integer numbers into their Roman numeral equivalent. Then, you will analyze various ways of structuring an algorithm when multiple decisions are required to determine which is the most efficient method.

* Open the romanConverter project. Finish the convert method that outputs the equivalent Roman Numeral for a given decimal number in the range of 1 to 100.  **Be sure to test your class and add your names in the banner comment before printing the class and a snapshot of the terminal window showing adequate test samples. Attach your hardcopy to this lab handout.**
  + HINTS: Use division and modulus operators to separate the number into the tens digit and ones digit.
  + Use 2 if statements:
    - the tens digit conversion
    - the ones digit conversion
  + Remember that (I = 1, V = 5, X = 10, L = 50, and C = 100). Additionally, 99 = XCIX ≠ IC.
  + Be sure you know the difference between *System.out.print* and *System.out.println*.

For Your Reference from <http://www.list-of.org.uk/list-of-roman-numerals.htm>.



* If three unique integers are input for the following two programs, then the output will be the largest of the three input values.

|  |  |
| --- | --- |
| **//Max of 3**  **public int max (int a, int b, int c)**  **{**  **if ((a > b) && (a > c)) {**  **return a;**  **}**  **if ((b > a) && (b > c)) {**  **return b;**  **}**  **if ((c > a) && (c > b)) {**  **return c;**  **}**  **}** | **// Nested Max of 3**  **public int max (int a, int b, int c)**  **{**  **if (a > b) {**  **if (a > c) {**  **return a;**  **}**  **else {**  **return c;**  **}**  **}**  **else if (b > c) {**  **return b;**  **}**  **else {**  **return c;**  **}**  **}** |

* Determine the number of comparisons required to compute the max of 3 unique integers using these programs for each test case. (Remember that Java does not evaluate the second operand of an “and” expression if the first operand is false):

|  |  |  |
| --- | --- | --- |
| Case | Max of 3 | Nested Max of 3 |
| 1 2 3 |  |  |
| 1 3 2 |  |  |
| 2 1 3 |  |  |
| 2 3 1 |  |  |
| 3 1 2 |  |  |
| 3 2 1 |  |  |

* In the worst, best and average cases, what is the number of comparisons?

|  |  |  |
| --- | --- | --- |
|  | Max of 3 | Nested Max of 3 |
| Worst Case |  |  |
| Best Case |  |  |
| Average Case |  |  |

* If three unique integers are input for the following two programs, then the output will be the three input values in ascending order. Assume the input values are random.

|  |  |
| --- | --- |
| **// Order 3**  **public void sort (int a, int b, int c)**  **{**  **if ((a < b) && (b < c)) {**  **System.out.println (“ “ + a + b + c);**  **}**  **if ((a < c) && (c < b)) {**  **System.out.println (“ “ + a + c + b);**  **}**  **if ((b < a) && (a < c)) {**  **System.out.println (“ “ + b + a + c);**  **}**  **if ((b < c) && (c < a)) {**  **System.out.println (“ “ + b + c + a);**  **}**  **if ((c < a) && (a < b)) {**  **System.out.println (“ “ + c + a + b);**  **}**  **if ((c < b) && (b < a)) {**  **System.out.println (“ “ + c + b + a);**  **}**  **}** | **// Nested Order 3**  **public void sort (int a, int b, int c)**  **{**  **if (a < b) {**  **if (a < c) {**  **if (b < c) {**  **System.out.println (“ “ + a + b + c);**  **}**  **else {**  **System.out.println (“ “ + a + c + b);**  **}**  **}**  **else {**  **System.out.println (“ “ + c + a + b);**  **}**  **}**  **else if (b < c) {**  **if (a < c) {**  **System.out.println (“ “ + b + a + c);**  **}**  **else {**  **System.out.println (“ “ + b + c + a);**  **}**  **}**  **else {**  **System.out.println (“ “ + c + b + a);**  **}**  **}** |

* Determine the number of comparisons required to sort 3 unique integers using these programs for each test case. (Remember that Java does not evaluate the second operand of an “and” expression if the first operand is false):

|  |  |  |
| --- | --- | --- |
| Case | Order 3 | Nested Order 3 |
| 1 2 3 |  |  |
| 1 3 2 |  |  |
| 2 1 3 |  |  |
| 2 3 1 |  |  |
| 3 1 2 |  |  |
| 3 2 1 |  |  |

* In the worst, best and average cases, what is the number of comparisons?

|  |  |  |
| --- | --- | --- |
|  | Order 3 | Nested Order 3 |
| Worst Case |  |  |
| Best Case |  |  |
| Average Case |  |  |

* Use your results from parts E2and E3 to fill in the following table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm | Max 3 | Nested Max of 3 | Order 3 | Nested Order 3 |
| Worst Case Number of Comparisons |  |  |  |  |

* Based on the results of your analysis of the algorithms in the table, write a hypothesis about the algorithmic structure of solutions to problems requiring multiple decisions.

**Part 2 – Truth Tables and Boolean Logic**

In this section, you will be working with truth tables in order to better understand fundamental Boolean logic.

1. Fill out the truth table below which describes the associativity property of the “or” operation. That is, the property that the order of operations does not matter among OR operations.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| **A** | **B** | **C** | **A || B** | **(A || B) || C** | **B || C** | **A || (B || C)** |
| false | false | false |  |  |  |  |
| false | false | true |  |  |  |  |
| false | true | false |  |  |  |  |
| false | true | true |  |  |  |  |
| true | false | false |  |  |  |  |
| true | false | true |  |  |  |  |
| true | true | false |  |  |  |  |
| true | true | true |  |  |  |  |

Which columns should be identical in the above truth table?

Answer: \_\_\_\_

## Create a truth table below to show that for Boolean expressions A and B,

## !(A && B) = (!A || !B)

* The example above is one of De Morgan’s laws of Boolean logic.
* Be sure to include all secondary steps in your truth table. For example, first show the column for A && B, and then show the column for !(A && B). Similarly, show two columns for !A and !B, and then show (!A || !B).

1. For each of the following, write Truth Tables to verify (prove) the follow laws of Boolean logic. Be sure to note which columns that should be identical in order to prove the result.
   * The Identity Law. A || A = A

Answer:

* + Distributive Law*.* A && (B || C) = A && B || A && C

Answer:

* + A || !A && B = A || B.

Answer:

* + (A || B) && (A || C) = A || B && C

Answer:

