Department of Electrical and Computer Engineering
ECSE 202 – Introduction to Software Development
Assignment 1
Introduction to Programming in Java

Problem Description

The objective of this assignment is to learn how to write a simple Java program that prompts the user for data, performs a computation on this data, and prints a result on the computer display. Admittedly, this is rather like teaching someone how to swim by throwing them into a pool, but it is an effective way to introduce the problem of developing software. Specifically, this program, which we'll call `dec2base` (*decimal to base*) will convert an input number in Base-10 to an integer value in Base-X where X represents the target base.

Example: convert 1278 to Base-2

```
Java base conversion demo:
Enter number to be converted: 1278
Enter base to convert to: 2
1278 is represented in Base-2 as 10011111110
```

Now Base-5

```
Java base conversion demo:
Enter number to be converted: 1278
Enter base to convert to: 5
1278 is represented in Base-5 as 20103
```

Where Do I Begin?

The lectures for Weeks 1 and 2 provide examples of simple Java programs that are very similar in form to the one you are asked to write. In particular, Add2Integers program should provide a lot of the structure that you need for this assignment. The heart of the program is the *method* that actually does the base conversion, so a good place to start is to look at the *algorithm* behind it.

Decimal to Binary Conversion

The ulterior motive behind this assignment (besides figuring out how to do basic coding) is to explore the internal representation in most standard digital computers, i.e., binary. When you use a decimal constant in your program, the compiler translates it into its corresponding binary representation. Equivalently, when you read in a decimal number from the input stream, a function is called which converts from decimal to binary representation.

Let's begin by considering binary numbers:

$124_{10} = 01111100_2$

Note the convention of using subscripts to denote base of representation. If no base is specified, the default is to assume Base-10. In more detail,

$$124_{10} = 01111100_2 = 0\times 2^7 + 1\times 2^6 + 1\times 2^5 + 1\times 2^4 + 1\times 2^3 + 1\times 2^2 + 0\times 2^1 + 0\times 2^0 \ .$$

The general form of this representation is

$$\begin{aligned} I_{10} &= b_{n-1}\times 2^{n-1} + b_{n-2}\times 2^{n-2} + \cdots + b_1\times 2^1 + b_0\times 2^0, \\ &= \sum_{i=0}^{i=n-1} 2^i \times b_i . \end{aligned} \quad . \tag{1}$$

So the problem of decimal to binary conversion reduces to finding the coefficients of the expansion for I_{10} above, where n represents the number of bits in the binary representation. We start off by writing I as an even number plus 0 or 1 as appropriate,

$$I = Q^{(0)}\times 2 + R^{(0)} \ . \tag{2}$$

Similarly we can write

$$Q^{(0)} = Q^{(1)}\times 2 + R^{(1)}, \tag{3}$$

or

$$I = (Q^{(1)}\times 2 + R^{(1)})\times 2 + R^{(0)} \ . \tag{4}$$

We can continue expanding I recursively,

$$\begin{aligned} I &= (Q^{(1)}\times 2 + R^{(1)})\times 2 + R^{(0)} \\ &= ((Q^{(2)}\times 2 + R^{(2)})\times 2 + R^{(1)})\times 2 + R^{(0)} \\ &= (((Q^{(3)}\times 2 + R^{(3)})\times 2 + R^{(2)})\times 2 + R^{(1)})\times 2 + R^{(0)} \end{aligned} \quad . \tag{5}$$

Expanding and collecting terms we obtain

$$\begin{aligned} I &= 2\times 2\times 2\times 2\times Q^{(3)} + 2\times 2\times 2\times R^{(3)} + 2\times 2\times R^{(2)} + 2\times R^{(1)} + R^{(0)} \\ &= 2^4\times Q^{(3)} + 2^{(3)}\times R^{(3)} + 2^{(2)}\times R^{(2)} + 2^{(1)}\times R^{(1)} + R^{(0)} \\ &= \sum_{i=0}^{n-1} 2^i \times R^{(i)} \end{aligned} \quad , \tag{6}$$

which is exactly the same form as Equation (1) at the top of the page for a binary number. In other words, the remainder terms, $R^{(i)}$, are the binary coefficients, b_i.

Decimal to Binary Conversion Algorithm

The key to a decimal to binary conversion algorithm can be seen in Equation (5). The superscripts attached to *Q* and *R* can be thought of as steps of an iteration, starting at 0 and progressing to *n-1*, where *n* is the number of bits in the binary number. Let's try this out starting with 0, and let 13 be the "input" to our algorithm:

$13 = Q^{(0)} \times 2 + R^{(0)} = 6 \times 2 + 1$	$Q^{(0)} = 6,\ R^{(0)} = 1$
$6 = 3 \times 2 + 0$	$Q^{(1)} = 3,\ R^{(1)} = 0$
$3 = 1 \times 2 + 1$	$Q^{(2)} = 1,\ R^{(2)} = 1$
$1 = 0 \times 2 + 1$	$Q^{(3)} = 0,\ R^{(3)} = 1$

Since $Q^{(3)} = 0$, the algorithm can go no further. The result is 1101which is the binary representation of decimal value 13.

Here is the decimal to binary conversion algorithm in psuedocode. Note that R[i] here represents an array of integers.

```
assign lastQ = number to be converted
assign i = 0

while (lastQ > 0) {
     Q = integer (lastQ/2)
     R[i] = remainder (lastQ/2)
     lastQ = Q;
     i  = i+1;
}
```

Let's "try" this algorithm with 124 as input.

```
lastQ = 124,   I = 0

Iteration 0:   Q = 124/2 = 62,     R[0] = 0
Iteration 1:   Q = 62/2 = 31,      R[1] = 0
Iteration 2:   Q = 31/2 = 15,      R[2] = 1
Iteration 3:   Q = 15/2 = 7,       R[3] = 1
Iteration 4:   Q = 7/2 = 3,        R[4] = 1
Iteration 5:   Q = 3/2 = 1,        R[5] = 1
Iteration 6:   Q = 1/2 = 0,        R[6] = 1
```

Result = 1111100

The code in the `while` clause of the psuedocode executes repeatedly as long as the control variable, lastQ, is greater than 0. Each pass through the code is referred to as an iteration, so we can see that it takes 7 iterations to convert 124 to binary. Let's take a closer look at the code in the `while` clause.

The first line computes the current value of Q based on its last value. If Q was assigned as an integer datatype, this statement could be written simply as `Q = lastQ/2,` since the assignment operator, =, will store the integer part of the right hand side by definition. In Java, the modulus operator, `%`, returns the remainder after performing integer division. That statement would be written as `R[i]=lastQ % 2`. Since we want to keep track of each digit of the answer, we can create an array, `R[]`, to hold each bit of the result. However, there is a *much* easier way to do this in Java which we will discuss shortly.

By the way, Equations (1) to (6) can easily be amended to handle *any* base. Instead of dividing by 2, you divide by the target base. Consider converting from decimal to hexadecimal notation (Base-16).

```
LastQ = 124,    I = 0

Iteration 0:    Q = 124/16 = 7,          R[0] = 12
Iteration 1:    Q = 7/16 = 0,            R[1] = 7
```

Result = 7C

A couple of observations are in order. First, if you replace each hexadecimal digit by its binary equivalent, one ends up with 7C = 01111100. Second, in printing out the result, as long as the target base is in the range of 2-10, the remainder will be in the range of [0,9], so each digit can be printed out directly. However when the base is 11 and above, the convention is to substitute letters of the alphabet starting at A. For the case of Base-16, the equivalents are as follows:

10	A
11	B
12	C
13	D
14	E
15	F

Converting Numbers to Characters

Consider the following String in Java

```
String LUT = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
```

Notice that it is ordered so that '0' corresponds to position 0, '9' to position 9, 'A' to position 10, 'B' to position 11 and so on. What we need is a function that "looks up" the character stored at the corresponding position.

In Java we do this as follows:

```
char Char10 = LUT.charAt(10);
```

In this case, the value of Char10 would be 'A' as wanted.

Storing the Intermediate Results

In the psuedocode example, the result at each interation was stored in an integer array. Since our result will be output as a string, we can simplify the program by converting the integer result at each iteration into its corresponding character value, and then concatenating that value to a string as follows:

```
String result = “”;                 // Blank string
while (lastQ > 0) {
   …
   result = result + LUT.charAT(lastQ % Base);
}
```

This gets you most of the way there. However, recall that the least significant digit is converted first, so the string is in *reverse* order. You will need to write a method to do this (a good exercise to familiarize yourself with Strings in Java).

Instructions

1. Write a Java program, `dec2base`, as shown at the beginning of these assignment notes. It should handle each of the cases shown and render the result in the target base. Your program *must* explicitly contain a method

   ```
   private String dec2B(int number, int base)
   ```

   that takes two integer arguments and returns a string containing the output, and a second method

   ```
   private String reverseString(String result)
   ```

   that takes a string as input and produces as output a string in reverse order.

   Your program should be set up to run in a loop, i.e., to successively read values from the user and print the result. One way of doing this, as outlined in the lecture notes, is to use Sentinal method that loops until the input value meets a specified condition. In this assignment, numbers are assumed to be positive or zero, so a negative value would be an appropriate signal to terminate the program as shown in the example below:

   ```
   while (true) {
     int number=readInt("Enter number to convert: ");
     if (number<0) break;
     <rest of loop code>
   }
   println("Program ended");
   ```

2. Run the following examples, making sure that your results are correct. Save these results to a file called dec2base.txt.

   ```
   Java base conversion demo:
   Enter number to be converted: 255
   Enter base to convert to: 2
   255 is represented in Base-2 as 11111111
   Enter number to be converted: 65535
   Enter base to convert to: 16
   65535 is represented in Base-16 as FFFF
   Enter number to be converted: 2147483647
   Enter base to convert to: 2
   2147483647 is represented in Base-2 as
   11111111111111111111111111111111
   Enter number to be converted: 33975
   Enter base to convert to: 5
   33975 is represented in Base-5 as 2041400
   Enter number to be converted: 760976
   Enter base to convert to: 9
   760976 is represented in Base-9 as 1378768
   Enter number to be converted: 67878903
   ```

```
Enter base to convert to: 10
67878903 is represented in Base-10 as 67878903
Enter number to be converted: 86
Enter base to convert to: 33
86 is represented in Base-33 as 2K
Enter number to be converted: 100100100
Enter base to convert to: 3
100100100 is represented in Base-3 as 20222100121112010
Enter number to be converted: 2147483647
Enter base to convert to: 35
2147483647 is represented in Base-35 as 15V22UM
Enter number to be converted: 17
Enter base to convert to: 8
17 is represented in Base-8 as 21
```

3. Make sure that your Java source file is properly documented and that it contains your name and student ID in the comments. Save this file as dec2base.java

4. Upload your files to myCourses as indicated.

Notes:

To receive full marks for this assignment, your program should replicate each of the examples shown above, including the text format.

About Coding Assignments

We encourage students to work together and exchange ideas. However, when it comes to finally sitting down to write your code, this must be done *independently*. Detecting software plagiarism is pretty much automated these days with systems such as MOSS.

https://www.quora.com/How-does-MOSS-Measure-Of-Software-Similarity-Stanford-detect-plagiarism

Please make sure your work is your own. If you are having trouble, the Faculty provides a free tutoring service to help you along. You can also contact the course instructor or the tutor during office hours. There are also numerous online resources – Google is your friend. The point isn't simply to get the assignment out of the way, but to actually learn something in doing.

fpf/January 16, 2018