Marcus Domingo

CSC205 section 1 Spring 2015
Homework 2

How to Submit:
Please submit your solutions (parts A and B separately) through Blackboard. Remember to put your name and homework number on all the documents that you submit as attachments.

Total possible points in this homework: 3 for Part B
(You receive 1 bonus point towards Part A if you get all Part A questions correct.)

Part A

1. You are given the 32-bit sequence 10000000 00000000 11111111 11111111 (or 0x8000FFFF). Answer the questions below.
   a. This code represents the integer -65535 in a sign-magnitude system. How to represent +65535 in the same system?
   b. This code represents the integer -2147418112 in a 1's complement system. How to represent +2147418112? In the same system?
   c. This code represents the integer -2147418113 in a 2's complement system. How to represent +2147418113 in the same system?

2. Perform the addition of the following numbers in 8-bit 2's complement representation and indicate whether overflow occurs.

   For example, 72+28:

	Carry				1	1			
	72		0	1	0	0	1	0	0	0
+	28		0	0	0	1	1	1	0	0
	100		0	1	1	0	0	1	0	0

No overflow

   a. -72+28:

	Carry			1	1	1			
	-72		1	0	1	1	1	0	0	0
+	28		0	0	0	1	1	1	0	0
	-44		1	1	0	1	0	1	0	0

Overflow? No

   b. 72+70:

	Carry		1							
	72		0	1	0	0	1	0	0	0
+	70		0	1	0	0	0	1	1	0
	142		1	0	0	0	1	1	1	0

Overflow? Yes Overflow

c. -72-70:

	Carry	1		1	1	1				
	-72		1	0	1	1	1	0	0	0
+	-70		1	0	1	1	1	0	1	0
	-142		0	1	1	1	0	0	1	0

Overflow?

Yes Overflow

3. In the IEEE-754 single precision floating point standard, what decimal number does the sequence 10111111 10000000 00000000 00000000 (or 0xBF800000) represent? You may leave it in the form $(+/-)2^L \times K_{10}$, where L and K are in decimal.

   Hint: You may use the following table to help you analyze the number.

Field	Bit values	Meaning, or decimal equivalent
Sign =	1	-1
Biased exponent value =	127	2^0
True exponent value =	0	2^0
Significand value =	0	1
The number is:	1.0×2^0	-1.0

4. Consider a floating point number excess-7 system with 4-bit exponent (where values 0000 and 1111 are reserved), 3-bit significand, with implied binary point and implied 1. Perform the addition of the two numbers represented respectively by 0 0100 010 and 0 1010 000. Show how the answer is represented in this system. Is the representation exact?

X = 0 | 0100 | 010 y = 0 | 1010 | 000

$X = 2^{-3} \times .01_2$ $y = 2^3 \times 1.00_2$

$X = 2^3 \times .00000001_2$

$Sum = 2^3 \times (.00000001 + 1.00)_2$

$= 2^3 \times 1.00000001_2$. 0 | 1010 | 000

Not Exact

Part B

1. [Total 3 pts] This question examines the limitations of the *int* and the *float* data types in Java, in which an integer is 32-bit and a float uses the IEEE 754 single precision format.
   a. [1pt] The following code segment computes the *exact* value of $C_R^N = \frac{N!}{R!\cdot(N-R)!}$ for the special case of $R = \lfloor N/2 \rfloor$ (meaning R is an integer that is half of N). Results are shown to the right of the code. Identify the type of error that occurs in the computation. Explain why it starts to occur at N=18 and not earlier or later.

Code segment 1 (in Java)	Output for N=1..30

public class Combination {
  private static final int defaultNMax = 30;
 	
  public Combination() {}
 	
  int compute(int N, int R) {
  int i;
  int numerator;
  int denominator;
 	
  numerator = 1;
  for (i=N-R+1; i<=N; i+=1)
  numerator = numerator*i;
 	
  denominator = 1;
  for (i=1; i<=R; i+=1)
  denominator = denominator*i;
 	
  return numerator / denominator;
  }
 	
  public static void main(String args[]) {
  Combination c = new Combination();
  int Nmax = defaultNMax;
 	
  for (int N=1; N<=Nmax; N+=1) {
  int result = c.compute(N, N/2);
  System.out.println("N=" + N + " C(" + N + "," + N/2 + ")=" + result);
  }
  }
}
	N=1 C(1,0)=1
	N=2 C(2,1)=2
	N=3 C(3,1)=3
	N=4 C(4,2)=6
	N=5 C(5,2)=10
	N=6 C(6,3)=20
	N=7 C(7,3)=35
	N=8 C(8,4)=70
	N=9 C(9,4)=126
	N=10 C(10,5)=252
	N=11 C(11,5)=462
	N=12 C(12,6)=924
	N=13 C(13,6)=1716
	N=14 C(14,7)=3432
	N=15 C(15,7)=6435
	N=16 C(16,8)=12870
	N=17 C(17,8)=24310
	N=18 C(18,9)=1276
	N=19 C(19,9)=-2308
	N=20 C(20,10)=117
	N=21 C(21,10)=9
	N=22 C(22,11)=19
	N=23 C(23,11)=1
	N=24 C(24,12)=3
	N=25 C(25,12)=0
	N=26 C(26,13)=0
	N=27 C(27,13)=0
	N=28 C(28,14)=0
	N=29 C(29,14)=-1
	N=30 C(30,15)=0

   b. [1 pt] In the following code segment, the programmer has converted the data type from *int* to *float*, and the results are shown to the right of the code.

      i. What is the first type of error that would occur? Explain.
      ii. At what value of N would you start to loose confidence on the result when using *float* to compute exact integers?

      (Hint: Consider what is the smallest positive integer that cannot be represented exactly using *float*.)

Code segment 2 (in Java)	Output for N=1..30

public class CombinationF {
  private static final int defaultNMax = 30;
 	
  public CombinationF() {}
 	
  float compute(float N, float R) {
  float i;
  float numerator;
  float denominator;
 	
  numerator = 1;
  for (i=N-R+1; i<=N; i+=1)
  numerator = numerator*i;
 	
  denominator = 1;
  for (i=1; i<=R; i+=1)
  denominator = denominator*i;
 	
  return numerator / denominator;
  }
 	
  public static void main(String args[]) {
  CombinationF c = new CombinationF();
  int Nmax = defaultNMax;
 	
  for (int N=1; N<=Nmax; N+=1) {
  float result = c.compute(N, N/2);
  System.out.println("N=" + N + " C(" +
N + "," + N/2 + ")=" + result);
  }
  }
}
	N=1 C(1,0)=1.0
	N=2 C(2,1)=2.0
	N=3 C(3,1)=3.0
	N=4 C(4,2)=6.0
	N=5 C(5,2)=10.0
	N=6 C(6,3)=20.0
	N=7 C(7,3)=35.0
	N=8 C(8,4)=70.0
	N=9 C(9,4)=126.0
	N=10 C(10,5)=252.0
	N=11 C(11,5)=462.0
	N=12 C(12,6)=924.0
	N=13 C(13,6)=1716.0
	N=14 C(14,7)=3432.0
	N=15 C(15,7)=6435.0
	N=16 C(16,8)=12870.0
	N=17 C(17,8)=24310.0
	N=18 C(18,9)=48620.0
	N=19 C(19,9)=92378.0
	N=20 C(20,10)=184756.0
	N=21 C(21,10)=352716.0
	N=22 C(22,11)=705432.0
	N=23 C(23,11)=1352078.0
	N=24 C(24,12)=2704156.0
	N=25 C(25,12)=5200300.5
	N=26 C(26,13)=1.0400601E7
	N=27 C(27,13)=2.00583E7
	N=28 C(28,14)=4.0116604E7
	N=29 C(29,14)=7.7558776E7
	N=30 C(30,15)=1.55117552E8

c. [1 pt] Suggest some possible ways that a programmer can protect the user from getting erroneous results when computing $C_R^N = \frac{N!}{R!\cdot(N-R)!}$.