Image Processing, Computer Graphics and Visualization using c++

Exercise 01

Negin Zarnegar

1.

variable	Final value	explanation
F1	13	F1 is evaluated to 12 at line 1 and then incremented by one where operator ++ is applied on it at line 3
F2	13	F2 is initialized to the same value as f1 where f1 itself has value of 12. Then its value is incremented by one at line 4 where operator ++ is applied to it at line 4
C1	68	Because In the case that the increase operator is used as a prefix (++f1) the value is increased **before** the result of the expression is evaluated and therefore the increased value is considered in the outer expression.[source: http://www.cplusplus.com/doc/tutorial/operators]
C2	67	in case that ++ operator is used as a suffix (`f2++`) the value stored in a is increased after being evaluated and therefore the value stored before the increase operation is evaluated in the outer expression

4.

variable	value	Explanation
Var1	Either 2,147,483,648 or -2,147,483,648	**Depending on the machine** (i.e. the word size of it). On a 32 bit machine, which the size of a word is 4 bytes and the signed values are defined in the form of 2's complement. The conversion of 2,147,483,648 into binary for a 32bit number means -2,147,483,648. In case the code is being compiled on a 64bit machine, there is no problem with an int value of 2,147,483,648.
Var2	Either -2,147,483,649 or 2,147,483,647	Again **depending on the word size**, in case of a 64bit machine, there is no problem having an int evaluated to -2,147,483,649 while on a 32bit machine, defining signed values by 2's complement of a 32bit variable, reducing the binary format (which is 1000..00) of -2,147,483,648 by 1, results in (011…11)the binary form of +2,147,483,647
b	4294967294	This is exactly the -2 in the format of 2's complement when considered as a normal binary number (unsigned)

5.

A floating point is decomposed to **sign**, **exponent** and **mantissa**

Sign: (could be) a single bit,

Mantissa: the binary form in normalized (always one 1,…..) form

Exponent: stored as 8-bit unsigned with a bias of 127 (i.e from -127 to 128)

In the format shown above, The approximate range of the float is from 1.0 * 2^-127 to 1.0*2^128. (The same range for negative values)