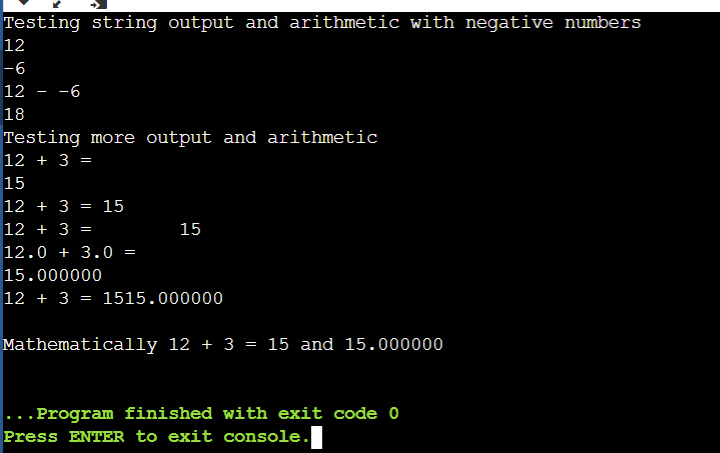
Kilibarda, Nebojsa 1001934192

1.a.i) The given line of output was: 12- -6

1.a.ii) The line produced was: 18

1.a.iii) The first line of code tells the program to print out two integers with assigned values 12 and -6, and to put a ‘-’ sign inbetween. The second line of code tells C to print out one decimal value, namely the result of performing subtraction between 12 and -6

1.b.i)



1.b.ii) The two numbers are printed out differently because on the first one %d is used (decimal form, for integers) and on the second one %f is used (floating point, which prints out decimal spaces and not just integers)

1.b.iii) Because the line ‘printf("%d", 12 + 3);’ did not end with an endline command (\n), the next line of code, ‘printf("%f\n", 12.0 + 3.0);’ printed right next to it.

1.b.iv) First statement has the \n in the line of code, which makes the result print in the next line in the output.

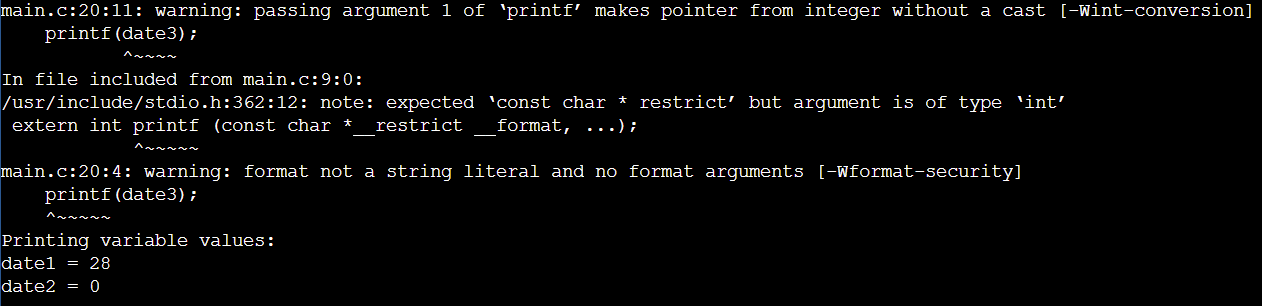
Second statement excludes the \n, so the output will continue on the same line

Third statement has \t instead of \n, which is the Tab character, causing the result to be a ‘tab’ away from the equal sign

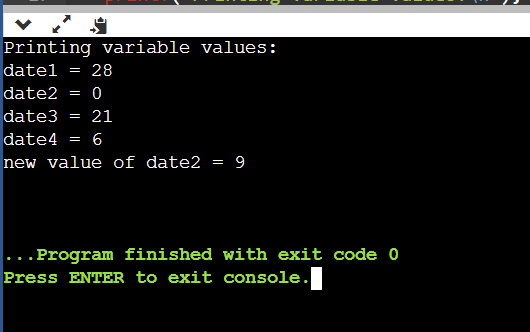
1.b.v) The first + sign is placed within “”. Because it's inside double quotes, it is printed out as part of the statement. In the other two places, it is used outside of double quotes, where it takes it’s normal form that denotes addition. The first of the two adds two integers, and the second one adds two floats

1.b.vi) Please see file NK4192Lab1Part1.c

2.a) This occurred because we didn’t tell C to print out an integer (“%d”), instead we only told it to print a statement and called on date3 integer, which didn’t have anywhere to print



2.b)



2.c)Output of the line is: Remainder of date1 & date2 = 1 :Remember the variables are int

2.d) The symbol ‘/’ between two integer numbers causes what’s called integer division, which doesn’t calculate the remainder of the division

The symbol ‘%’ between two integer numbers causes C to calculate the remainder of the division

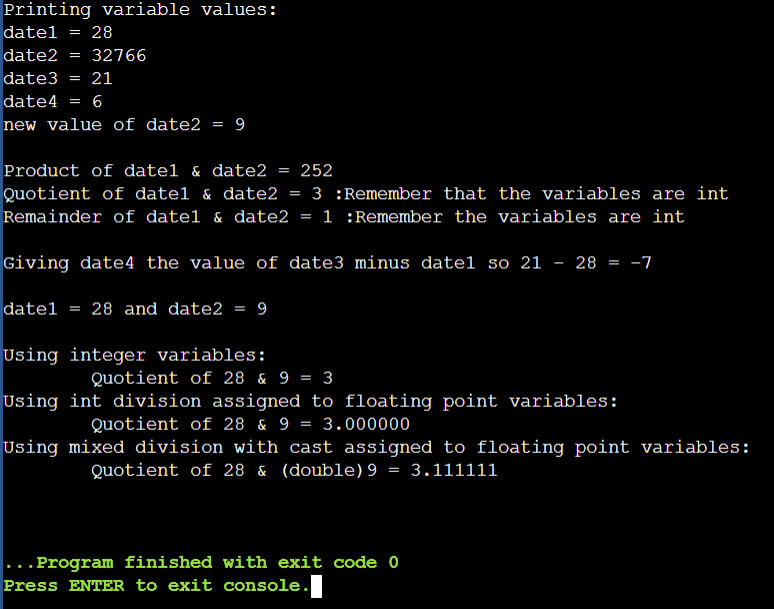
2.e) The = is an assignment operator, and it serves to assign a value to a variable in C

2.f) By casting date2 as a double, C balanced date1 into a double for this calculation, which gave us the result in the form of a double. This is why the output had the correct calculation past the decimal point, instead of just 00000s like in the previous line.

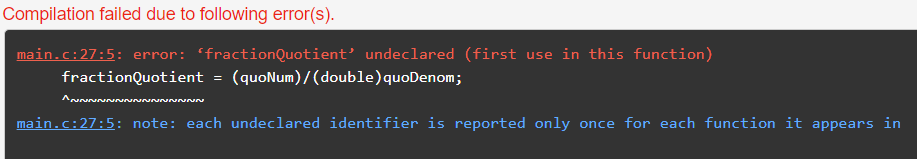
2.g) In the first calculation, integer division is performed (date1/date2) and the result is assigned to a double value, which causes C to print zeros after the decimal point. In the second calculation a (double) cast is assigned to one of the operands, which makes both of them act as (double) in the calculation, and provides the correct answer.

2.h) 5 variables are declared. The first four are integers, and they are: date1, date2, date3, date4; the fifth variable is a double and it is: date5.

2.i)



3.a)

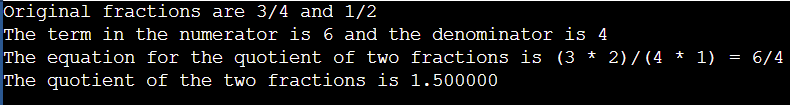


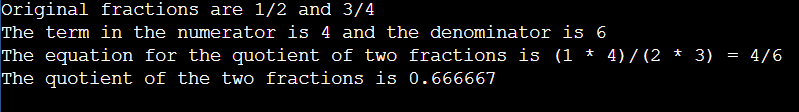
3.b) The line with the error is line number: 27

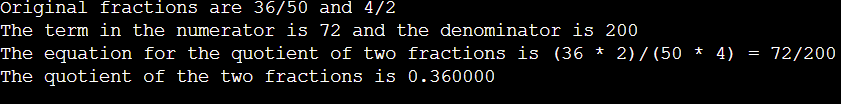
3.c) The error message from 3.a means that we attempted to use an undeclared value, or in other words a value that we still haven’t ‘told’ C about. The blue note says that no matter how many times an undeclared value appears within the function, it will only be reported once

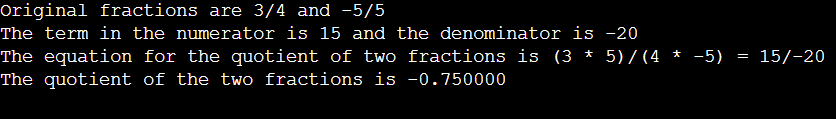
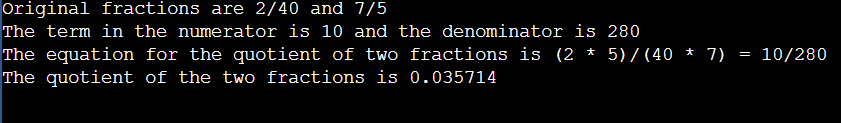
3.d) Please see file NK4192Lab1Part3.c

3.e) (First screenshot is with the original values)









3.f) The fractional answer is not in the simplest form because the fractions can be further reduced to:

3/2

2/3

9/25

1/28

-3/4