CS 31 Project 2 Report

1. One obstacle I faced was how to format all the “if”, “else if”, and “else” statements in a logical order that covers all the conditions spelled out in the project spec. Although I understood “if” and “else” statements conceptually, I struggled with applying that concept for some of the conditions, since placing “if” statements within other “if” statements sometimes got confusing. For example, it took me a while to figure out how to organize the code when writing the tax reduction part based on how many children the user had. (as seen below) I found it helpful to write out and organize the “if” and “else” statements on paper first before writing it in Visual C++ so that I could better visualize the program.

if (0 <= taxableIncome && taxableIncome < 120000)

{

if ((pretax - (200 \* children)) < 0)

tax = 0;

else

tax = (pretax - (200 \* children));

}

Another obstacle I had to overcome was finding logic errors in my code. It was difficult to recognize the details I missed such as including “&&” in some of my “if” statements which caused the logic errors. (as seen below) Since the logic errors were caused by the lines of code that dealt with numerical data, I first thought the way I formatted the calculations was wrong because the program gave the wrong numbers. I kept looking for my mistake in those lines of code, not realizing that those lines were right, and my mistake lied elsewhere. At the end, I decided to look over my code line by line which is when I found my error. At first glance, it is hard to find a small error within many lines of code, so I have realized while doing this project that writing code section by section and testing each section out before moving on could have helped mitigate such logic errors.

if (0 <= taxableIncome && taxableIncome <= 50000)

Empty string provided for name ( \_, n/a, n/a, n/a)

Taxable income is negative (Sam, -120000, n/a, n/a)

Empty string for occupation (Sam, 120000, \_, n/a)

Number of children is negative (Sam, 120000, bartender, -3)

Valid inputs (Sam, 110000, engineer, 0)

Taxable income value is under 50000 (Sam, 30000, bartender, 0)

Taxable income value is between 50000 and 120000 (Sam, 100000, bartender, 0)

Taxable income value is over 120000 (Same, 200000, bartender, 0)

Taxable income value is between 50000 and 120000 and occupation is engineer or scientist (Sam, 110000, engineer, 0)

Taxable income value is over 120000 and occupation is engineer or scientist (Sam, 200000, engineer, 0)

Tax reduction calculation for taxable income value under 50000 (Sam, 30000, engineer, 6)

Tax reduction calculation for taxable income value between 50000 and 120000 (Sam, 110000, engineer, 6)

Tax paid is a negative value (Sam, 25000, bartender, 10)

Income tax has value for cents (Sam, 29544.69, bartender, 0)

The first four test inputs could be used to test the error messages in the program. Each of the first four inputs correlate to a certain error message that should be seen when inputting either nothing (denoted “\_” above) for “name” and “occupation”, or a negative value for “taxable income” and “number of children”. The fifth test input could be used to test the whole program and see if every function works properly. The next three sets of inputs could be used to test the tax brackets (under 50000, from 50000 to 120000, and over 120000). The next two inputs could be used to test the occupation condition by itself with one case where the taxable income is between 50000 and 120000 and the second case where it is over 120000. The next two inputs could be used to test the tax reduction condition with one where the taxable income is under 50000 and the other where it is between 50000 and 120000. The next input listed could be used to test what the program does when the tax paid calculates to a negative value which in such case the program should tell the user to pay $0.00 rather than a negative value. The last test value I listed could be used to test the precision of the calculations in that the tax value produced should not have more than two values to the right of the decimal point. There are probably more tests one could perform, but these are some that could test the conditions given by the project spec.