# Task 1 Items C, D, and E

## Explain the Relationship between Flowchart and Pseudocode

**By David J Kelley**

(dkelle24@wgu.edu)

### Introduction

In this assignment, I will describe the relationship between the various documents from Items A and B. That means the logic behind the flowchart and pseudocode, as well as the alignment between them. Essentially, the flow chart is a visual representation of the ask of the assignment, and the pseudocode is the logical walk-through of the steps of what is in the flow chart. These are done to help ensure the underlying logic of the task and the application that would be built is correct. While this doesn't prevent all bugs or issues, it generally ensures that the application being written is on the right track for the given process, as documented by the flow chart and pseudocode.

In general, the requirements in the rubric are somewhat ambiguous, so I've tried to align with the level of complexity that I would use in my regular job if I were to use these tools.

### Task 1 Item A: Create a Flow Chart

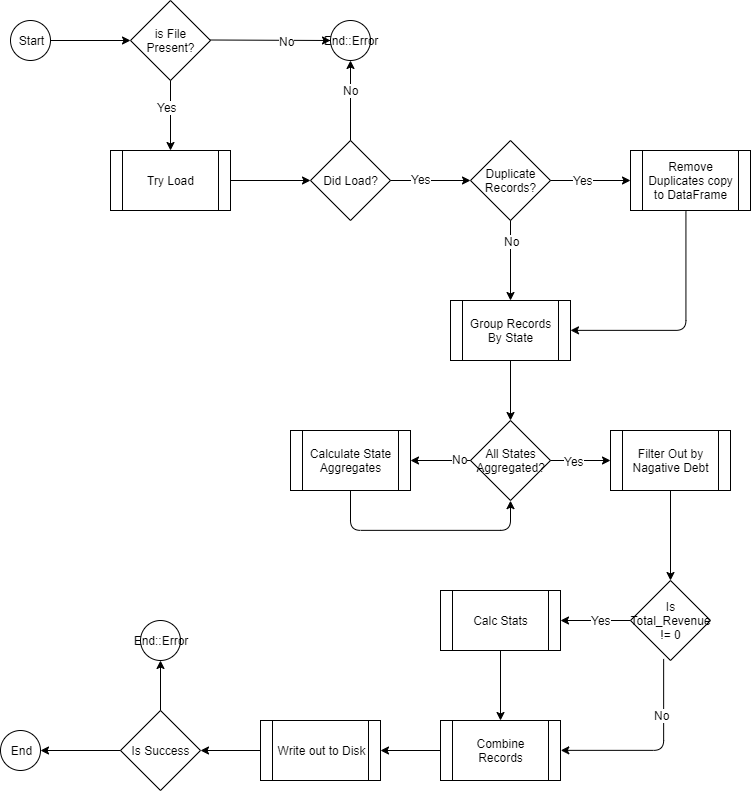
Refer to the accompanying document [D598 QKN1\_TASK1\_Item A - Flow Chart.drawio] [D598 QKN1\_TASK1\_Item A - Flow Chart.png]. Note that 'Draw.io' is not one of the listed tools for flowcharts, but it is equivalent to the free tool Lucid, which is mentioned.   
  


Figure 1A – The flow chart

This is a visual presentation of the assignment, available here [https://tasks.wgu.edu/student/012361758/course/33300017/task/4432/overview], which outlines a bulleted list of 6 items or tasks that need to be completed sequentially, with some implied logic as shown in the diagram.

Let us take a look at the following pseudocode.

### Task 1 Item B: Create Pseudocode

The following C-style pseudocode can be found in the accompanying file [D598 QKN1 Task 1 Item B - Pseudocode.txt].

// 1. Import the data file into a 'Data frame?'

DataTable data = ImportCsvToDataTable("path\_to\_file.csv")

// 2. Identify duplicate rows in the frame?

DataTable duplicateRows = data.AsEnumerable()

.GroupBy(row => string.Join("|", row.ItemArray))

.Where(g => g.Count() > 1)

.SelectMany(g => g)

// 3. Group all IDs by State and calculate floats

DataTable groupedStats = data.AsEnumerable()

.GroupBy(row => row["State"])

.Select(g => new {

State = g.Key,

MeanRevenue = g.Average(r => Convert.ToDouble(r["Revenue"])),

MedianRevenue = Median(g.Select(r => Convert.ToDouble(r["Revenue"]))),

MinRevenue = g.Min(r => Convert.ToDouble(r["Revenue"])),

MaxRevenue = g.Max(r => Convert.ToDouble(r["Revenue"])),

}).ToList()

// 4. Convert groupedStats to new DataTable

DataTable statsByState = ConvertToDataTable(groupedStats)

// 5. Filter businesses with negative debt-to-equity ratios

DataTable negativeDebtToEquity = data.AsEnumerable()

.Where(r => Convert.ToDouble(r["DebtEquityRatio"]) < 0)

.CopyToDataTable()

// 6. Create a new DataTable with the debt-to-income ratio

DataTable debtToIncome = data.Clone()

debtToIncome.Columns.Add("DebtToIncome", typeof(double))

// 7. Populate the debtToIncome object

foreach (DataRow row in data.Rows) {

double longTermDebt = Convert.ToDouble(row["LongTermDebt"])

double revenue = Convert.ToDouble(row["Revenue"])

double dti = revenue != 0 ? longTermDebt / revenue : 0

DataRow newRow = debtToIncome.NewRow()

newRow.ItemArray = row.ItemArray

newRow["DebtToIncome"] = dti

debtToIncome.Rows.Add(newRow)

}

// 7. Concatenate the debt-to-income ratio DataTable with the original one

DataTable finalResult = data.Copy() + debtToIncome.Copy()

// 8. Do something with the output

return finalResult;

Figure 2A –Pseudocode of logic assignment

Now let's walk through both of these.

### Task 1 Item C: Part 1: Flowchart Walk-through

Referring to the flow chart (see Figure 1A), I used the standard 'process' boxes and input/output boxes for flowcharts in Draw.io (the tool I was using). I also split some tasks and added implied logic. For example, in code, I might check if the file is present and attempt to load it in a single line of code. In contrast, the diagram breaks down that operation and illustrates the flow of what happens based on the condition.

Typically, I would classify this as an ETL process and would generally have it done by a script, such as Python. This means that some tasks, such as a failure, might be an error, and that is acceptable as a script. However, suppose I were to use a compiled process. In that case, I may need additional logic beyond this diagram for handling complex errors. Since I know this will be written in Python, I used an end state of type 'Error' in the flowchart. This could also be done in a compiled process that runs the ETL script and has its own error handling when the script errors out. This logic would not be in the context of the actual Python, so it would not be included in the diagram.

The rest of the flow in the diagram is more or less linear, but it tests for certain conditions and performs actions based on whether specific elements are present. For instance, if duplicate records are found, they are removed, and then the data is grouped by state. For each state, a loop is executed to calculate all the necessary aggregates. Then the flow of the logic checks for "Total\_Revenue" is not 0 and will then calculate additional stats and for those that didn't match it will pass them on and then merge both sets back to together and continue process and then writing to disk and checking to see if that was successful and if so ending typically or throwing an error and ending anyway.

Now, referring to the Pseudocode in Figure 2A, the logic appears more linear because much of the decision-making is performed together. To walk through the pseudocode logic, we can see that we are trying to load a data file (a \*.csv) into an object or some similar structure. Then, we look for duplicates and separate them (item labeled 2). Next, we perform the grouping operations and calculate aggregates (item labeled 3). Then we convert the states into the kind of object that allows for processing, and in item 5, we can filter, copy results, and test for the DebtEquityRatio. Then we can create another version of the core object that we will use to complete our work with the data. Then, in item 6, we will perform all the debt-to-income calculations. In item 7, we will merge the data left in our data object and add the debt-to-income records into a final object, which we can then return, write to disk, or perform any other desired action with.

### Task 1 Item C: Part 2: Pseudocode vs the Flowchart

Both items look at the logic from a slightly different angle. If we look at the item labeled "1." From the pseudocode and the flow chart, we check if the file is present by trying to load it. If it loads, we continue and look for duplicate records in the item labeled "2." Because our intent is to write a script, we are not handling cases where the file doesn't exist other than to error out. We are also dealing with cases where the file may exist but doesn't load, and in both cases, it errors out. In the flow chart, there are several boxes and decision points. Still, since it's a script, we don't really care if it errors out, at least not in the ETL process at the level we are looking at with both the flow chart and pseudocode. Similarly, the rest of the diagram and flow chart can be mapped in a similar manner, where the pseudo code will perform tests, aggregate, copy, and other operations together. In real code, we will need to declare values and code them out, but in most scripted languages used for ETL processes, it is not necessary.

All in all, the flowchart diagram shows the logic split out a bit more, where the pseudocode is closer to how the computer will actually run the program, and helps work out the kinks logically before moving to actual code.

### Task 1 Item D: References

In this assignment, I didn't use any references except those noted in the assignment itself.