NORTHWEST NAZARENE UNIVERSITY

Endowment Sorting Computer Program for Small Universities

THESIS
Submitted to the Department of Mathematics and Computer Science
in partial fulfillment of the requirements
for the degree of
BACHELOR OF ARTS

Hayden M. Crabb
2021

THESIS
Submitted to the Department of Mathematics and Computer Science
in partial fulfillment of the requirements
for the degree of
BACHELOR OF ARTS

Hayden M. Crabb
2021

Endowment Sorting Computer Program for Small Universities

Author: 
Hayden M. Crabb

Approved: 
Barry L. Myers, Ph.D., Department of Mathematics and Computer Science, Faculty Advisor

Approved: 
Donna M. Allen, Ph.D., Department of Communication Arts & Science, Second Reader

Approved: 
Barry L. Myers, Ph.D., Chair, Department of Mathematics & Computer Science

Abstract

Endowment Sorting Computer Program for Small Universities.

CRABB, HAYDEN (Department of Mathematics and Computer Science),
MYERS, DR. BARRY (Department of Mathematics and Computer Science).

Patrons to many universities donate large sums of money to the schools and the students through specific endowments. These donations are often what allow students to continue attending the university. Each donor has specific requirements that students must meet in order to receive their money. To find students who meet these requirements and satisfy the donor's desired specifications, financial aid employees must manually search through a comprehensive database of students to find matches. In order to make this process more efficient and find students for all the endowments, an endowment sorting computer program was developed. The tool developed through this project searches the databases of endowments and students in order to generate a report of who should receive what. Northwest Nazarene University is a smaller private university which utilized the tool as a test of its effectiveness. Through the work of this project the university was able to automate many parts of this process. During this test, the tool saved nearly a week of manual labor time and proved its effectiveness. Additionally, the program has been developed in such a way that it can continue to be effective even as requirements and endowments change, as well as potentially being used at other universities in the future.

Acknowledgements

I would like the many people that helped make this project a reality. First, Northwest Nazarene Universities past Financial Aid Director Ann Crabb for technical and conceptual help with the process of awarding endowments. Also, I would like to thank both Dr Barry Myers and Dr Dale Hamilton for their technical help with the programming process, and Dr. Donna Allen for her thoughtful guidance and suggestions.

Table of Contents

List of Figures

Background

Universities across the world rely on endowments to support the school and to provide financial support. “An endowment is an aggregation of assets invested by a college or university to support its educational and research mission in perpetuity” (American Council on Education, 2021). This means the universities receive donations from a variety of sources and utilize the combined funds as investments. The interest earned from this fund is a means of financial support for the university. Each donor generally has specific requirements on how their donation can be used for the university. They may specify that the money be used for student financial aid, teaching, new research, athletics or in other areas. Additionally, the university is legally bound to follow these specifications outlined by the donor. (American Council on Education, *Facts About College and University Endowments*). The university, therefore, must find a way to correctly and efficiently match the donor's money for its intended purpose. The focus of this research is on endowments that are designed to fund student financial aid. Many donors want their funds to enable specific types of students to attend college. These requirements can be very particular, forcing the money to go towards students with specific GPAs, hometowns or even more detailed requirements.

The Problem

The process of finding a student who matches each specific endowment’s requirements can be very difficult. This process is especially challenging when the

endowment has stricter parameters on who can receive the money. Many universities still utilize traditional methods of storing and finding this data. In the case of Northwest Nazarene University Excel spreadsheets are used to track each endowment's requirements and each student's information. This process is not ideal for several reasons.

First, it forces the university to manually input data from a variety of sources which could lead to human errors. It is also very time consuming for the Financial Aid department. These issues alone make it necessary to utilize computing processes in this financial aid process.

Second, financial aid officers must search through each spreadsheet manually to find students who qualify for each endowment. This involves identifying the requirements of an endowment, then laboriously searching through the Excel spreadsheets for a student that matches. This process can be very time consuming, especially as the size of the university grows. The complexity of this issue costs the university both time and money.

The Solution

In light of these particular challenges and the lack of other software solutions, a C++ Windows-based computer program was developed to tackle the problem. Originally, the program was designed as a console application that could be run by the developer and people with technical knowledge. This program was not particularly user friendly but it accomplished the task. The program took two Excel sheets as the input, one for the endowment data, and one for the matching students data. The program then ran analysis on the data and produced a list of students and the endowments they could receive. While

the data was accurate, and successfully solved the problem, there were several issues. The method that allowed the computer to interpret the Excel sheets was hardcoded into the program. Due to this, the Excel files could not be modified or the program would not recognize the data types. Columns could not be added, deleted, or modified without breaking the entire process. Additionally, the number of rows in each sheet was hardcoded, and the program would result in a buffer overflow error if rows were added.

In order to solve these problems, this particular research project was undertaken. To promote user accessibility, a GUI was implemented as a Win32 application. The GUI also allows the program results to be modified with settings that the user can manipulate. In order to accept variance in the Excel sheets, the program was created so that the user can select and edit the data in a specific "setup" screen. The setup screen also enables different universities to use the program with their own unique spreadsheet. The program then produces a results spreadsheet which makes recommendations on which students should receive which funds. There were a variety of development decisions that made this program effective. Each choice was intentional in order to produce specific results, these decisions and development processes are documented below.

Development

The development of the Endowment Sorter program was done utilizing C++ and Microsoft Visual Studios to create a Win32 application. There are three main parts of the program; The select files screen, the setup screen, and the results page. For instructions on how to use these different screens please see the documentation. This section will talk about how these screens function.

Program functionality

The select files screen utilizes the common item dialog function to create simple windows that can be utilized to select files to open. This program only accepts Excel files in comma separated values (CSV) format. These dialog boxes are common to Windows applications so users are familiar with the way the function. For this program the common item dialog was useful for selecting files, and was later utilized to save the resulting file as well.

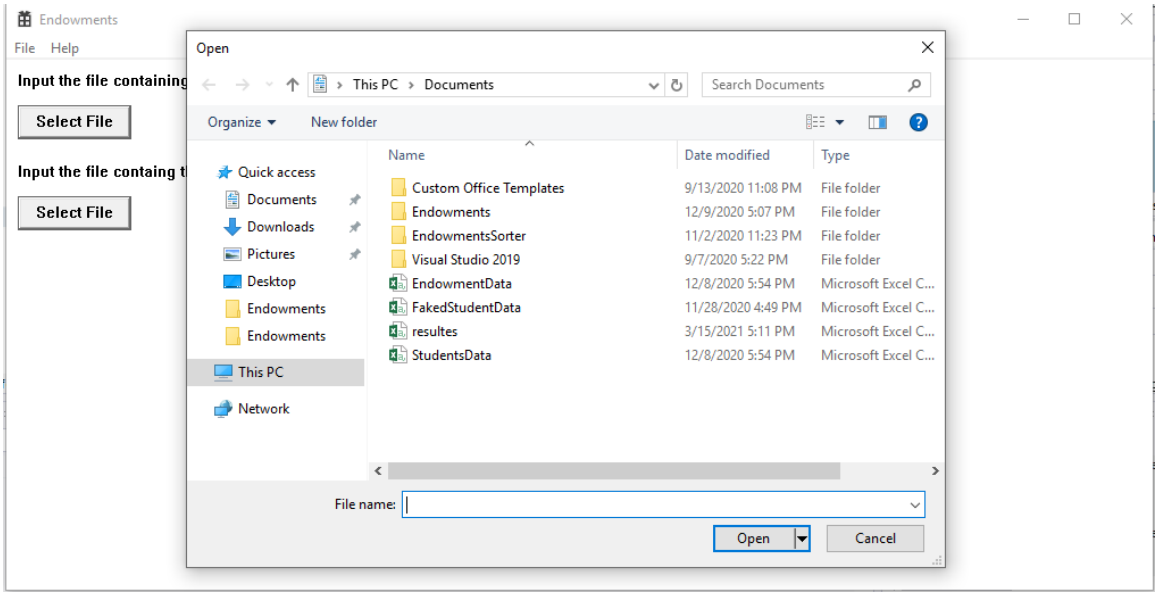


Figure 1. The Common Item Dialog Box.

Once both files have been selected, the first line of each file is read in and stored in the attributes doubly linked list (DLL). This process is diagrammed in Figure 2 below. Each attribute also corresponds to a particular type of data, the full list of data types includes: Information, Balance, Low Income Students, Greater Than or equal to, Max money to award, or List of Strings. For a full explanation of each of these data types, please see the Endowment sorter Documentation section Types of Data. The attributes DLL is a global

array that is used to determine how to handle each piece of data that is read in by determining the data type. Different types of data are treated differently.

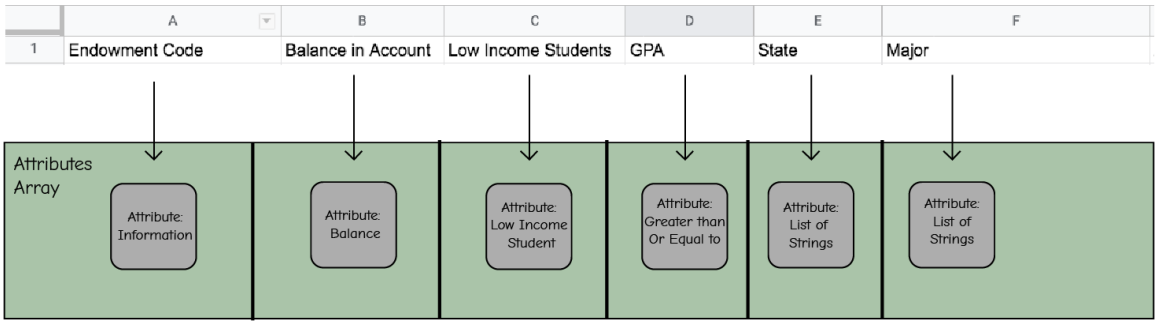


Figure 2. Process of how Excel columns become attributes

When the first line of the Excel sheet is first read in, the data is assumed to be the type "List of Strings" by default, but the user can change the type of data during the setup screen as shown in figure 3. A complete demonstration of how labeling data works can be found in the documentation under the Using the program section. Once the user has labeled each attribute with its corresponding type, the main execution portion of the program is run. First, the rest of each file is read in. The full code for how this section works can be found in the attached code in the inputEndowmentsData function and the inputStudentsData function. Each endowment is created as an instance of the endowment class. Within each instance is a DLL of requirements. The requirement class is a data structure designed to store each type of data. The requirements DLL parallel the attributes DLL, they are kept parallel so that the program may understand how to handle each requirement. Once the files have been inputted the comparisons and awarding begins. Before I examine the exact awarding details, in order to create context I will discuss the matching process that is utilized to award money to students.

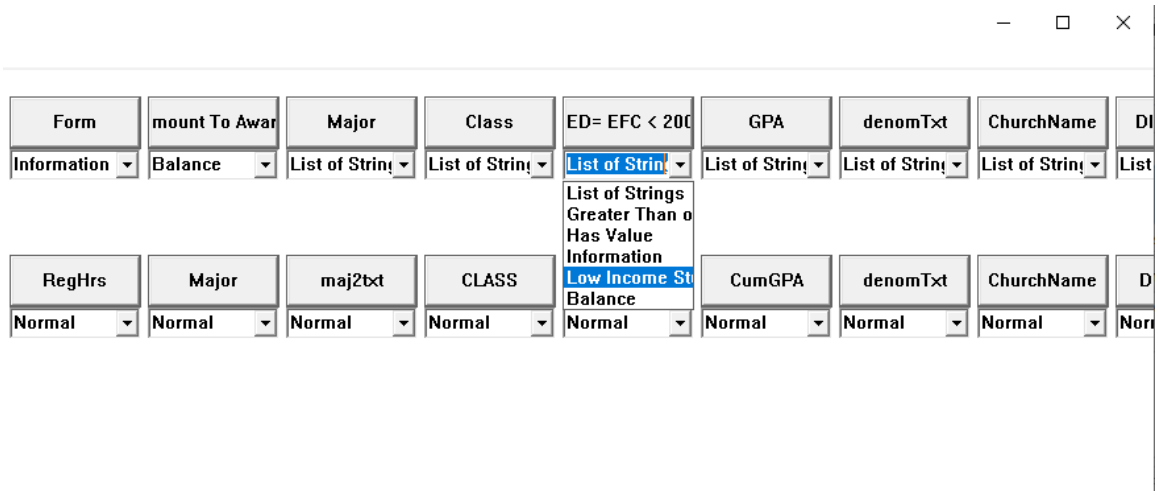


Figure 3. Selecting the type of data for each column

Matching Process

The ultimate goal of any financial aid office is to find as many students that can match with as many endowments as possible. This helps keep the donors satisfied because their money is going to students in need. However, since students are limited in how much money they can receive, there is a scarcity of students and many endowments that need to be awarded. It is important that a student who matches with a more strict endowment (an endowment that can only be given to some students based on its requirements) does not receive their money from a more general endowment that could be given to anyone. I will discuss an example in order to understand the concept fully.

Matching Process Example

Say endowment A is a stricter endowment designed to award money to low income students in ministry whereas endowment B is a more general endowment that could be awarded to nearly anyone. Student 1 qualifies for endowment A and B, whereas student 2 only matches with endowment B. Both students only need a small amount of

money to fulfill their needs. Say the program awards student 1 with money from endowment B, now student 1's financial needs are met. However, now there is no student that could potentially match with endowment A. This leaves endowment A with money in its account and Student 2 has not received any money at all. In order to fix this it is important that stricter endowments get their choice of students first. Endowment A gives to student 1, and endowment B gives to student 2. Everyone is satisfied and the largest possible amount of money is awarded.

In order to do this with code, the program runs through the doComparisons() function one time to find how many students qualify for each endowment. At this point in time it does not actually assign them the scholarships, but creates a list of all the students who can potentially receive the scholarship. This is done in order to determine which endowments are the easiest to find students for (stricter endowments), and those that are more challenging (general endowments). Based on these numbers the endowments are then sorted from strictest to most general so that the most strict endowments can have their pick of students first.

Producing Results

Once the strictest endowments have been identified, the program runs the doComparisons() function again. This time using the list of potential students to find students that match with each scholarship and awarding them money based on the endowment's balance, and the student's amount of needed aid. The information of each student is stored within each Endowment class instance.

Finally the program reaches the results page. The code produces a Comma Separated Values (.CSV) file that can be loaded back into Microsoft Excel for viewing.

This file is generated by looping through the list of endowments and displaying all the students who should receive money from them. The current resulting file layout is designed to meet the needs of Northwest Nazarene University. See figure 4 below for a reference of this structure. In this example, The AID column is a primary key which identifies endowments, and the Student ID column is a primary key that identifies each student. In future work other page formats could be implemented or the user would be given more control to adjust this format.

The program currently has limited error reporting capabilities. The fourth column in the resulting data spreadsheet represents a column the computer believes may have caused problems with that endowment. The current method for identifying errors is elementary, but still helpful. The Program simply identifies the requirement in each endowment that caused most students to not match and labels this column as the potential problem. If a user believes there are errors in the report, this column would be a good place to start looking.

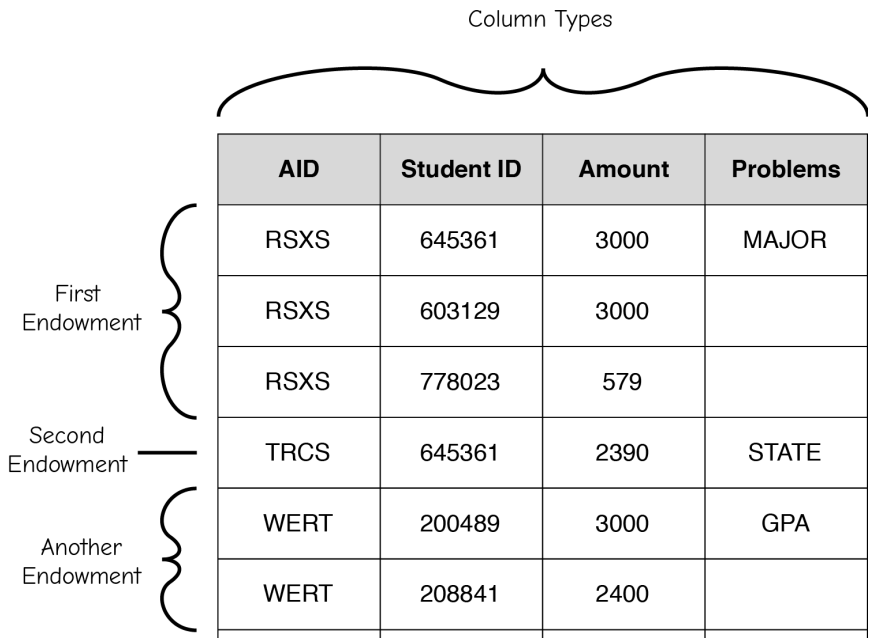


Figure 4. Current setup of the resulting data csv file

Programming Choices

A variety of intentional programming choices were made in order to make this program function the way it does. There are many ways to resolve each problem that was faced in this development, but particular decisions were made. The next section will discuss several techniques that were used and why those decisions were made.

Graphical User Interface (GUI) Development

There are a variety of ways that could have been utilized in order to produce a GUI for the application, but a traditional Win32 application was chosen. First, the two primary constraints to this decision were based on the need for this application to run on Windows operating systems, and the desire to write code in C++. The aim of the software

development project was to create a Windows application as this is a common operating system, and also the operating system of Northwest Nazarene University the subject of this research.

The second primary constraint was that the interface had to be created using C++. As mentioned previously, this research effort was to further the developments made in years prior on this project. The previous code written was in the language C++ and it was desirable to reuse sections of this code as much as possible.

The Microsoft Windows Developer Documentation denotes a variety of different ways to create desktop applications in the Windows environment including; Universal Windows Platform (UWP), WPF and Window Forms, Win32 and a variety of other third-party development options (Microsoft Windows Developer Docs, 2021). Other than Win32 the only other option that was seriously considered was Window Forms. Both are solid development platforms, but ultimately Win32 was chosen because of its extensive documentation and proven reliability. Both options could have achieved the purpose. Choosing the GUI development platform was the first main programming decision made.

Doubly Linked Lists

In this program there were many instances that required arrays. However, the size of those arrays was not given. For this project doubly linked lists (DLL) were chosen over dynamically allocated arrays because of their functionality. There are two main advantages to the DLL's and some disadvantages as well. First, a DLL has an indeterminate size. This was extraordinarily valuable for the purpose of this program because the program does not know how many students, or how many endowments it will have to work with. Rather than having to calculate how many there would be, the

linked list allows values to be pushed onto it as long as there is memory available on the computer.

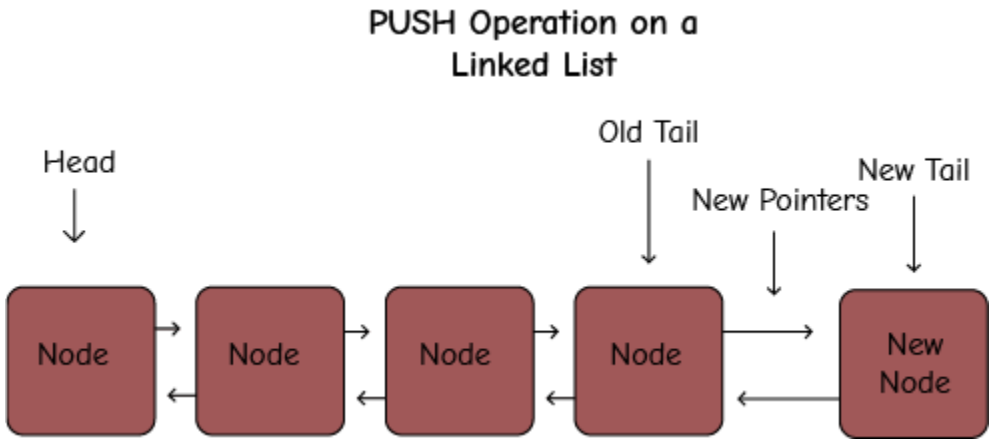


Figure 5. Linked list pushes an infinite amount of nodes onto the list

This allows there to be a nearly infinite number of endowments and students in each Excel sheet. For each new student or new endowment that the program has to consider, it can simply push new dynamically allocated data onto the list. This process is simple, clean and effective.

Second, the doubly linked list also allows for easy reordering because no data has to be copied, the pointers which link the data simply need to be switched. This is critical because it saves time and resources. This process saves the processor from having to copy data from place to place. A portion of the process is a sorting algorithm which utilizes a select sort to organize the endowments based on how many students they can match. Therefore, making the swapping process more efficient is a critical component.

The downside of doubly linked lists is the difficulty in implementing them utilizing code. Arrays are much simpler to implement in C++ as they are overall much simpler. Although challenging, the advantages outweigh the disadvantages and doubly

linked lists were utilized to store the list of endowments and the list of students. A DLL was also utilized for the lists of attributes for the endowments, and the attributes of the students. This was a second large development decision that had to be made.

A Note on Databases

It is worth mentioning why databases and SQL were not implemented on this project, even though they may seem like a very efficient solution. SQL is made to search through huge stores of data and pluck out the exact row desired. This is exactly what the Endowment Sorter program is intended to do, so why was SQL not implemented? The simple answer is that it would be more challenging to format the data for SQL queries than to utilize unconventional query techniques.

In an ideal environment both the endowment and student data would be located in a normalized database where queries could be run. This would make development incredibly simple. However, the data presented in this case was in the form of Excel files. Rather than creating a relational database to store the data, it was simplest to manipulate the data through classes in memory, and work with it there. As mentioned in the background, this program was created in a niche market where a centralized database system does not exist. So while creating a database would be a better solution if you are looking at this problem from the big picture perspective, this program was designed to sort Excel files, and it does that quite well. This is just another example of the programming decisions that had to be made.

Results

This research proved the effectiveness of this computer application. According to Northwest Nazarene University's Financial Aid Director, Ann Crabb the program successfully awarded 168 endowed and current scholarships totaling $721,896. This money was awarded to 190 unique students and saved the department 19-25 hours of manual labor time (A Crabb, personal communication, October 19, 2020). This data indicates the effectiveness of the program and how valuable it can be for financial aid offices. The larger the school is, the more time this program will save. With continued modifications and improvements the tool could become even more effective.

Challenges

The most complicated portion of this entire project is data purification. Once the files have been successfully inputted, the process is relatively simple. However, there are many challenges that need to be overcome when inputting the files. Primarily is the issue of human error when entering data.

A common issue is with Machine Readability: human entered data that makes sense to them does not make sense to a computer. For example the program failed to match a student with the major of *SCNDRY ED* with then endowment requiring students to have a *SECONDARY EDUCATION* major. To a human, these are obvious matches, but the computer does not see them as the same at all. A potential solution to this includes using a major code instead of text. An integer code would be machine readable and more efficient. This would be an easy, and ideal solution, but the program was forced to work

with the data given, and a major code was not provided. The solution that is likely to be implemented in the future is to attempt and find an algorithm that solves the closest string problem (CSP). Using A CSP algorithm would match the endowment with the student if their data string was within an acceptable distance from the target string. This is a desirable solution because it would also fix issues with typos in the data.

Future Work

There is a lot that can be done to improve this program and its results. These improvements could help make the program more usable and increase its efficiency. Some of the possible improvements include adding settings to allow adjustments, enhancing the look and feel of the program, and improving computing performance efficiency.

Adding Settings

Several areas could potentially be improved by adding settings. These settings could be accessed on the top bar of the main window, and would create an additional pop up window where the settings could be adjusted as shown in Figure 6. These settings would allow the user more control over what the program does. Potential settings include adjusting the low income student classification. (By default this is set to less than $20,000 of estimated family contribution.) The default minimum and maximum amount to award could become a setting, enabling the program to suggest different amounts to award to each student. Settings could also be used to determine how the output should look,

whether to output the results to a file or print them to the screen. These would all greatly improve the functionality of the program, but were outside the scope of this project.

Figure 6. Potential location for settings menu

Error Reporting Improvements.

The current mechanisms of error reporting could be greatly improved, and would therefore greatly increase the quality of the program. As mentioned in the challenges section there are many errors caused by human data that the computer does not interpret as acceptable. In these instances it would be helpful to indicate that some sort of error has occurred. When an endowment can not find any matching student, it would be helpful to indicate why. Currently, the program indicates the most likely problem column, but it could be improved greatly. There is an opportunity for machine learning techniques to be implemented to suggest which data is incorrect, but there are other potential solutions as well. One such method that has not yet been implemented would be that for each endowment, ensure that at least one student can qualify for every single column. If there is a column that is causing every single student to fail, it is likely an error.

It would also be incredibly valuable to report these errors to the user before they finish using the program so that they can fix them as the program runs. However the improvements are made, each addition will greatly enhance the performance of the program as a whole.

Design Improvements

Additional work could be done to improve the GUI of this program, and improve the overall user experience and usability. The current program was designed as a first version of the program to test its effectiveness at solving the problem. Very little effort was made towards making the program beautiful, or especially user friendly. The current screens of the program feature dull gray boxes and do not look especially appealing. While functional, they leave a lot to be desired from a graphic design perspective. The program currently lacks other key user experience devices such as loading bars.

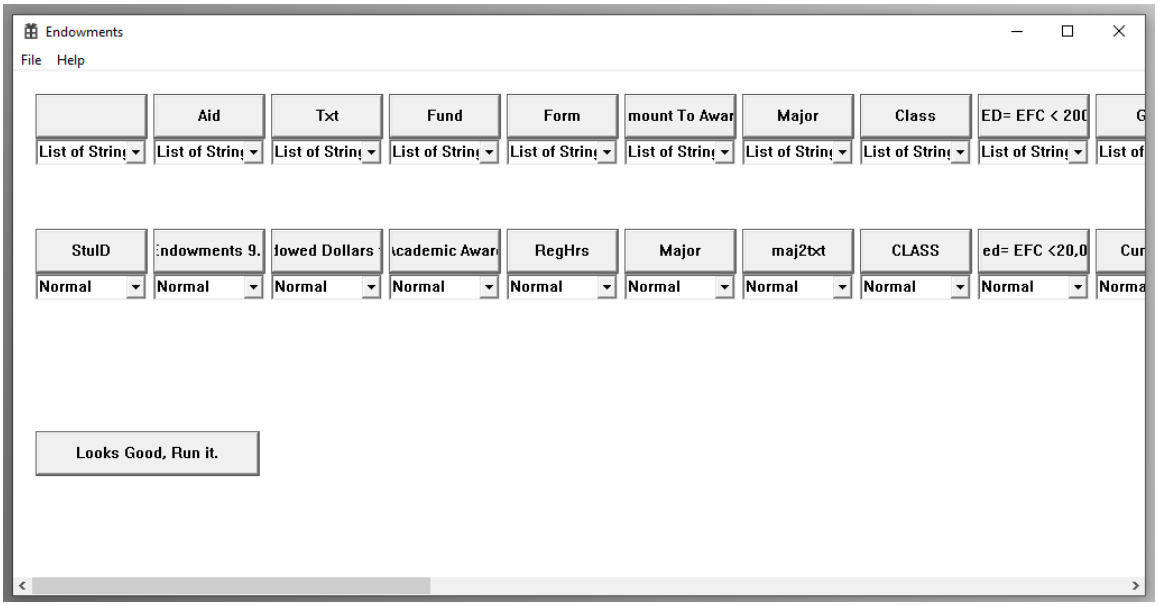


Figure 7. Current Setup Screen - In need of visual improvement

Conclusion

The Endowment Sorter program development was a successful research project that indicated the usefulness of this computer application in this scenario. This project

was successfully able to take the previous work and create a Windows application that enabled the user to interact with it. The improvements also enabled the program to accept Excel sheets ordered and structured in any manner. The program is capable of functioning not just on a specific set of requirements, but on any requirement that the user creates. If an endowment requires a student to enjoy mornings and coffee it can search for that, given the data exists. The possibilities created by these improvements are endless. In this particular research setting the program saved the Northwest Nazarene University Financial aid department 19-25 hours of manual labor, a significant feat for the size of the university. There are many opportunities for this program to be improved and modified to further increase the impact this project has on the financial aid industry. As modifications are continued to made, the possibility of successfully utilizing this tool at other universities is hopeful.

References

American Council on Education. (2021). *Understanding College and University Endowments.* https://www.acenet.edu/Documents/Understanding-College-and-University-Endowments.pdf

American Council on Education. (n.d.). *Facts About College and University Endowments* https://www.acenet.edu/Documents/Facts-About-College-and-University-Endowments.pdf

Microsoft Windows Developer Docs. (2021). *Choose your windows app platform.* https://docs.microsoft.com/en-us/windows/apps/desktop/choose-your-platform

Appendix A: Usage Documentation

Overview

The endowments sorter program is a windows computer application designed to help small universities to match their students to donors’ money in endowments. Many small universities keep track of both the varying requirements of these endowments and the matching student information in Excel spreadsheets. This tool is designed to process information in both spreadsheets and produce a resulting spreadsheet which identifies which students should receive which endowment’s money.

Who can use this program?

The endowments sorter program is intended to be used by small universities who don't have a larger financial aid software to support them. This program aids in the process of choosing which students should receive money from donor's endowments. If your financial aid office has to manually search for students who would qualify for each endowment, this program is for you. This program can be used when certain criteria needs to be met by each endowment.

What does it do?

The program will find students who match the requirements of endowments so that the money in those endowments can then be awarded. It does this by going through the following four steps. This is an overview of what the program will be doing, but if you are looking for a more detailed explanation of how to utilize the program, please see "using the program."

Gathering information:

The endowments sorter program reads in the requirements and information of each endowment from an Excel spreadsheet and compares this with a matching spreadsheet which contains all the student's information.

Setting up Data:

The second part of the program is the Setup data screen. At this point in the program the user indicates what type of data each column represents, and matches this data to the corresponding data in the students spreadsheet. The goal of this section is to tell the computer how to process the data it's received from the spreadsheets, and how it should compare the endowments to the students.

Data crunching:

After the user has labeled the data, it's time for the computer to compare the data and decide which students should receive money. It does this in 2 parts.

1. Identify which endowments are easiest and hardest to match students with.
2. Allocate the money from each fund to the students. Endowments that are harder to find students for will be allocated first in order to ensure their money is used. More on this later.

Producing results and identifying errors:

After the endowment's money has been allocated the program produces an Excel spreadsheets which identifies which students should, and can, receive money from each endowment. The program also suggests how much money should be given to each student.

The program attempts to identify endowments & students that have errors in their data and will notify the user to examine the data to determine whether an error has occurred or not.

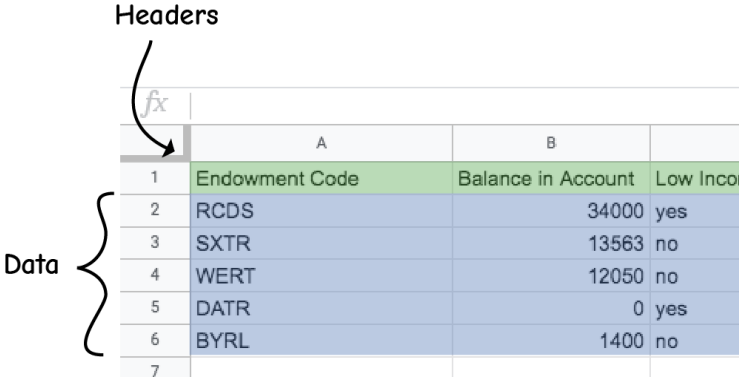
Formatting the Excel Sheets

This program utilizes two spreadsheets as input in order to do the computation: A spreadsheet of each Endowment and it's requirements, and a spreadsheet of each student and their corresponding information. The program assumes that the data in those spreadsheets is formatted in a particular way. The goal of this section is to inform you how to format your spreadsheets so the program can run through the data and match it correctly.

General Layout

The layout of the spreadsheet is essential to assure the program interprets the data correctly. *In this overview we'll mainly focus on the file containing data about the endowments, as this information is slightly more complicated.*
The program assumes that in each spreadsheet, each row represents a new endowment or a new student. Each column represents a piece of information about each student or endowment.

In each file, the first row is reserved for the headers; bits of text which indicate what each column contains.

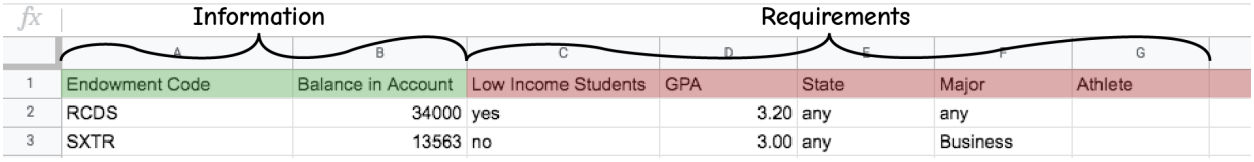


In the spreadsheets, each row is another student or endowment. (in this image endowments are shown). Add as many rows as you have endowments. Here only six endowments are shown, but the program

can handle an infinite number.

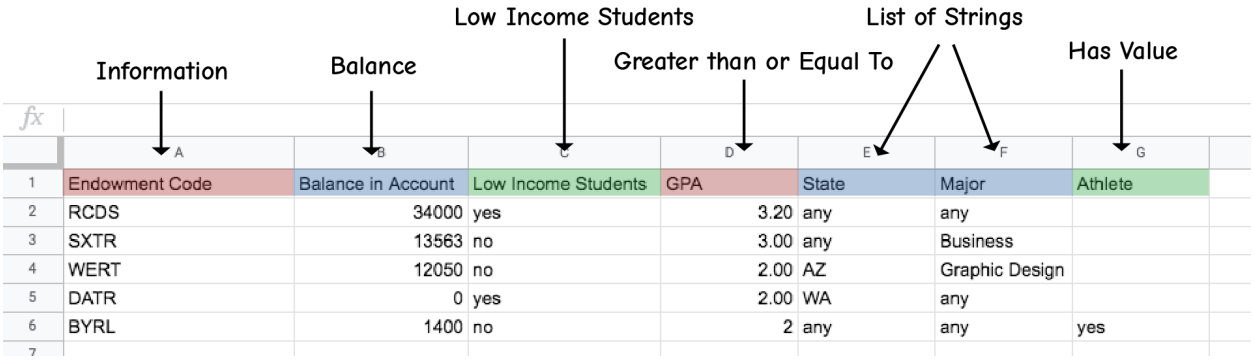
Types of Data:

The columns in the endowment spreadsheet represent either information or requirements.
Information is utilized to uniquely identify each endowment, and the requirements indicate what students need to have in order to qualify for the endowment's money.



	A	B	C	D	E	F	G
1	Endowment Code	Balance in Account	Low Income Students	GPA	State	Major	Athlete
2	RCDS	34000	yes	3.20	any	any	
3	SXTR	13563	no	3.00	any	Business	

There are six types of data that are supported by the endowment sorter program. Each type of data has a unique purpose.



	A	B	C	D	E	F	G
1	Endowment Code	Balance in Account	Low Income Students	GPA	State	Major	Athlete
2	RCDS	34000	yes	3.20	any	any	
3	SXTR	13563	no	3.00	any	Business	
4	WERT	12050	no	2.00	AZ	Graphic Design	
5	DATR	0	yes	2.00	WA	any	
6	BYRL	1400	no	2	any	any	yes

All six examples of different data types are shown here, let's break down what they mean:

Information

Any type of data.
Information is data that is important for a human to use, but not necessary for the computer to utilize. Any data marked as information will essentially be ignored in the matching and sorting processes. In the image above the code which identifies the endowments isn't used to match students, so it is labeled as information.

Balance

A number value.
The balance type is a subset of information. The balance isn't used to match students to the endowment, but it is utilized to determine how much money can be awarded. The computer will use the number in the balance column to allocate money from. If there is $10,000 in the balance column, the computer will award $10,000 dollars from that endowment. If there is only $300 in the column, the computer will only allocate $300 dollars. Endowments with a 0 in this column will be skipped completely.

Low Income Students

"Yes" or "No"
When marked with a "yes" value, the endowment requires a student who is considered low income. When marked with a "No" value, the endowment will accept any kind of student.
Students are considered low income when their estimated family contribution (EFC), taken from their FAFSA, is less than 20,000. This number can be adjusted in the settings.

	A	B	C
1	Student ID	Max Dollars to a	EFC
2	77321	10000	18199
3	74891	13900	0
4	75320	9900	13956
5	77901	15900	No Fafsa
6	76598	11000	22805

This is an example of a corresponding data sheet containing the student's information. In Column C we see the EFC values for each student. Students in rows 2,3 and 4 would be considered low income. Students in rows 5 and 6 would not be low income. If a student does not submit a FAFSA, such as the case for the student in row 5, it is assumed that they are not low income because they did not need to apply for federal student aid.

Greater than or equal to

Number value
A column marked as greater than or equal to will contain a numerical value. A student will need to have a corresponding value greater than or equal to the endowment's value in order to match with this endowment. This data type is frequently used with GPAs but it can be used in any situation.

List of Strings

A list of values separated by the pipe | character
This is the most common, and default, data type. The student value must be a word, or character matching one of the endowment's words or characters in order to match.

Secondary Education
Graphic Design | Business | Marketing | Digital Media
Any

An example of acceptable List of Strings values in the endowments spreadsheet.

In row 2, a Student with the degree "Marketing BA" would match. A student with a "Finance BA" degree would not match.

If the column is marked with the word "Any". The endowment will match with any student regardless of their major. The word "Any" is used whenever a particular endowment does not require a specific value for that column.

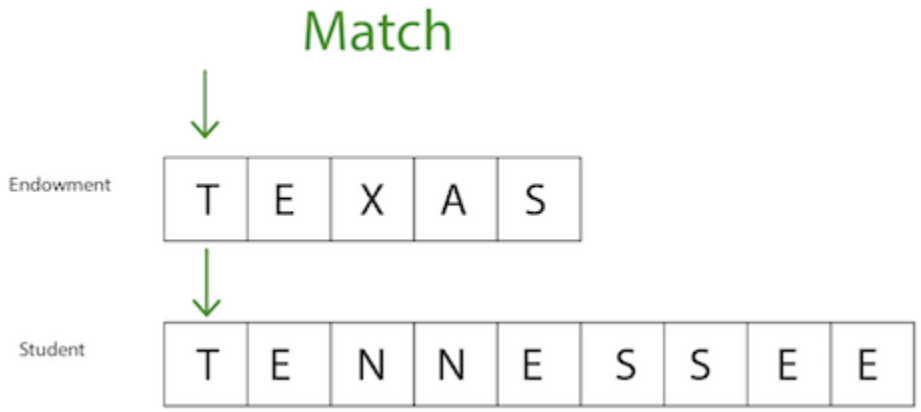
Purifying data

After the endowment sorter program has read in the two spreadsheet files, it formats the data so that a computer can understand it. As the user of the program, there are a couple things you should consider in order to make sure the program gets the best results possible. There are also some things the program does automatically that you don't have to worry about.

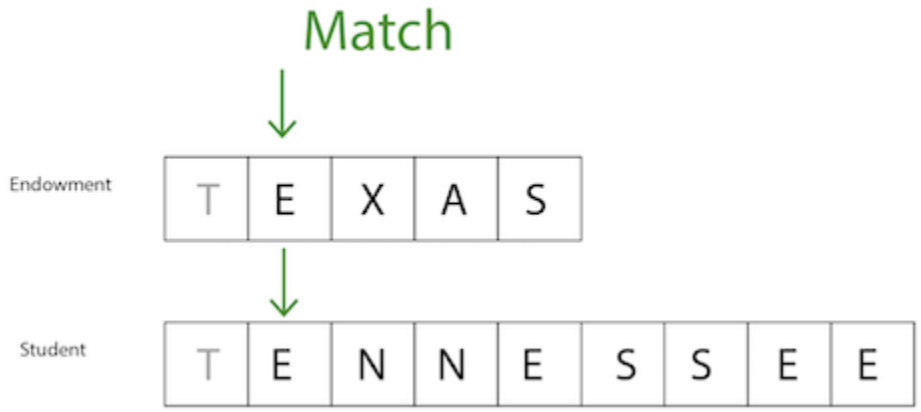
How the Computer understands matches.

When the computer is comparing two pieces of data to see if there is a match, it looks at each letter in the string in order to determine if the data is the same, and therefore matches. This usually occurs when comparing the list of strings data type. For example, let's say one of the columns on the endowment data is searching for students in a particular state. In the column marked state, the endowment has the value Texas. The computer finds a student with the value of Tennessee in its state column. The computer does not yet know if these values match, so it begins comparing the two values. It will begin by examining both words letter for letter.

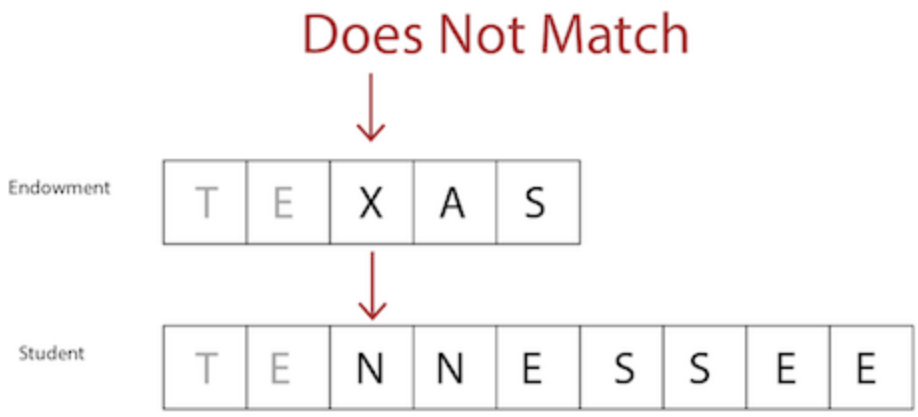
Step 1:



The first letters of both words match, so the computer proceeds to the next letter.
Step 2:



The second letters match as well, so the computer continues checking.
Step 3:



The letters in the 3rd column are not the same, therefore these words are not the same, and the columns don't match. As a result of this the computer will stop checking. It now knows that this student is not from the correct state, and will search for another student.

As a result of how the computer compares data, there are a couple things we can do as users of the program, to help the computer, and ensure that data matches with the data it's supposed to.

Ensure Data Cohesivity

It's very important that the data in both spreadsheets is designed to match with each other. Sometimes information that makes an obvious match to a human won't for a computer. For example, An endowment may require a student to be from the state "Idaho" but in the state column the student has marked that they are from "ID". To a human it's easy to see that then endowment requires the student to be from Idaho and the student is indeed from Idaho, but the computer will not see this as a match because it will not understand the abbreviation. If the data is abbreviated in one spreadsheet, it needs to be abbreviated in the other to match.

Watch out for typos.

The current ability of the program does not allow for there to be any difference in the data being compared. If there is a difference between the students data and the endowment data it's being compared against, it won't create a match. Due to this typos can cause endowments to not match with students and vice versa. For example, a endowment might require matching students to have a degree in "Economcs" (Note that Economics is misspelled). Students who have a degree in Economics wouldn't match

with this scholarship because of the typo in the data. It's important to proof-read through the data to ensure there are no spelling errors.

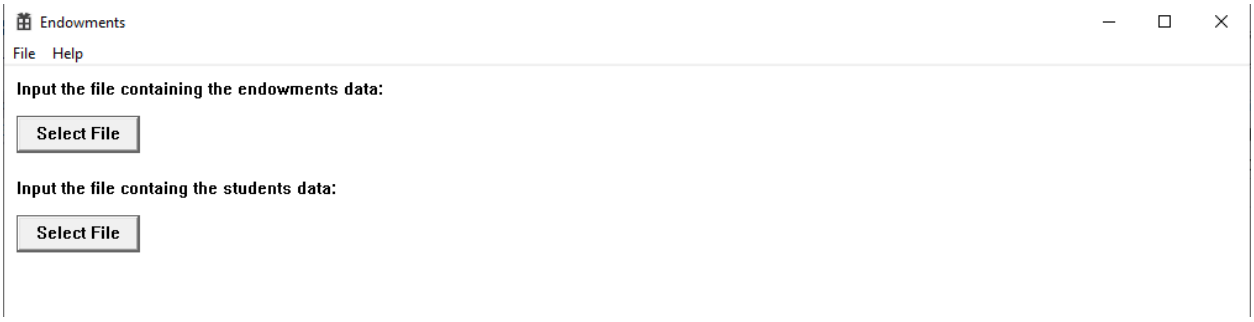
Don't worry about Capitalization

The computer automatically capitalizes all of the data it takes in. That way the word "Mission" Will match with the word "mission". In this program the data will match whether or not it's capitalized.

Using the Program

Selecting the Files Screen

The first screen shows two buttons which allow the user to select the endowment data, and the student data. Clicking each of these buttons will open a dialog box that will allow the user to navigate to, and select the corresponding piece of data. The screen looks like this:

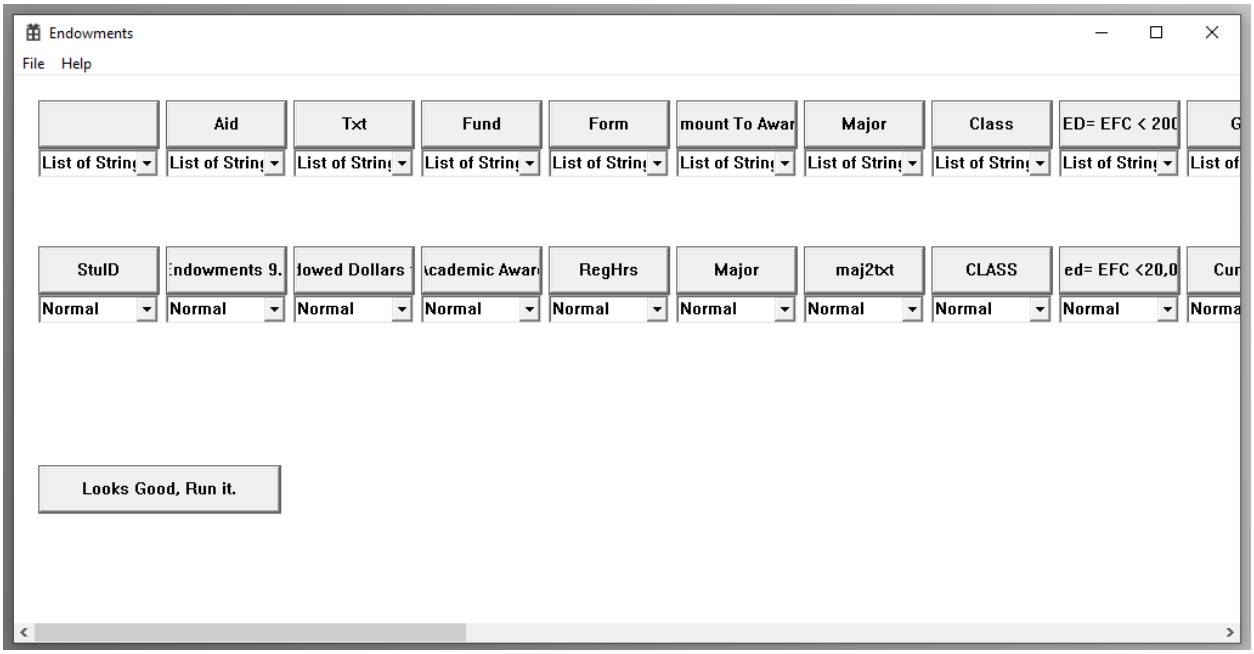


Make sure to select the endowment's data for the endowment button, and the student's data for the student button.

This program only accepts the .csv (Comma separated Values) file format.

The Setup Screen

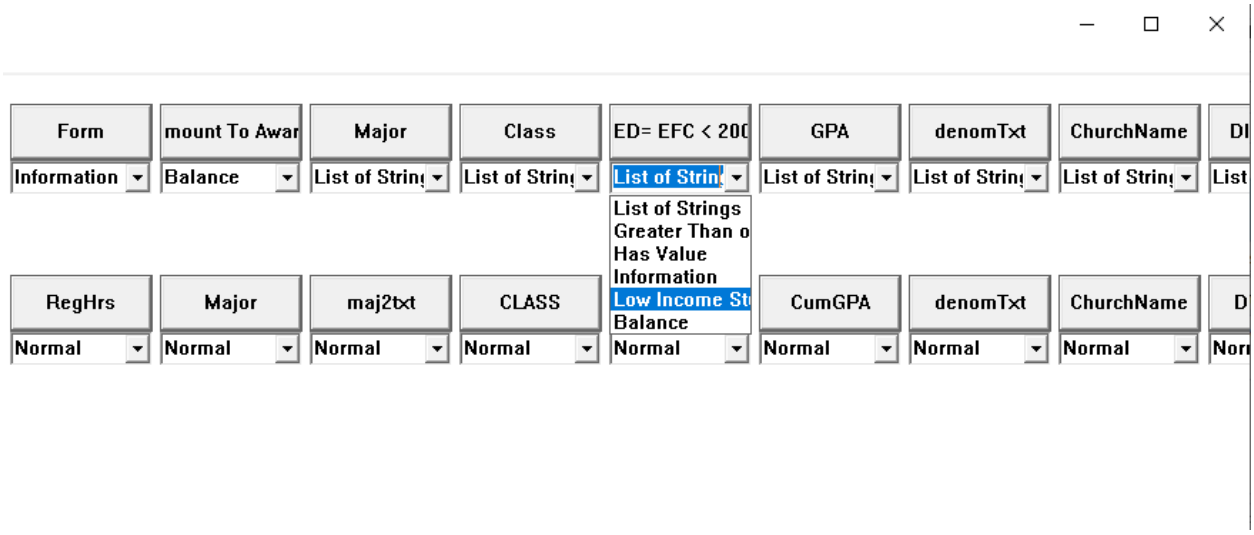
The setup screen allows the user to indicate what type of data exists on the files. The setup screen will look something like this:



The top row indicates the columns that are on the endowments file. The second row of grey boxes indicates the columns that were found on the students file. Underneath each box is a drop down which indicates the type of data that will be found in that column for each endowment. The word in each grey box is taken directly from column heading (the first row) on each spreadsheet.

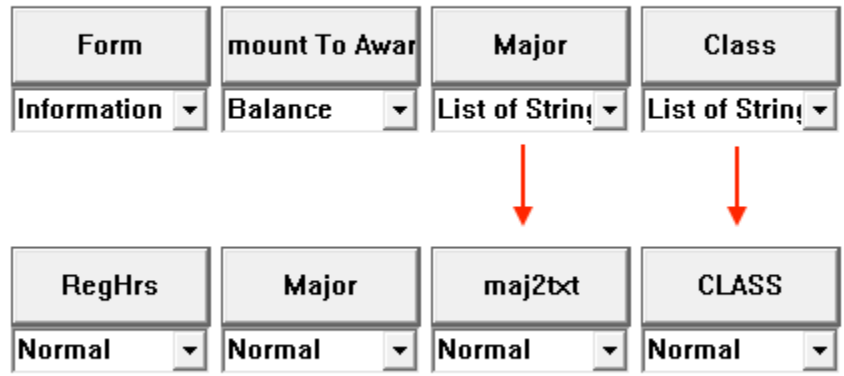
While on this screen the user needs to indicate the correct type of data in each column, and indicate the corresponding column it will match with on the student's data. Each column by default is a list of strings data type.

Select the correct data type for each column on the endowment spreadsheet by pressing the dropdown arrow and clicking the correct data format. Do this for each column.



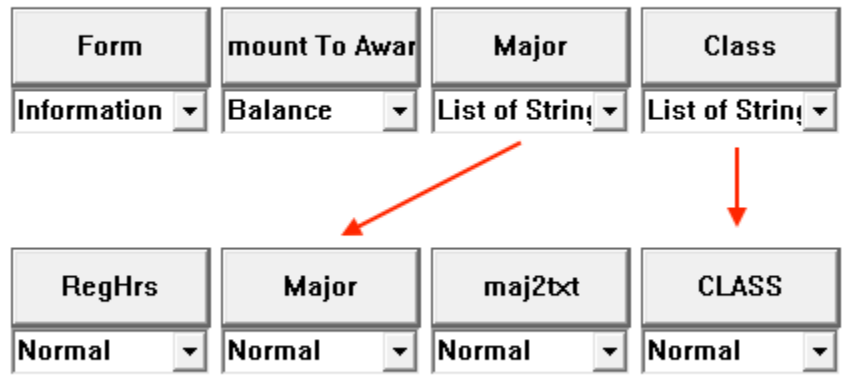
If the user incorrectly selects a data type for a column the program will produce errors. So make sure to spend some time on this screen in order to ensure the data is correctly labeled. See the data types section of this documentation for descriptions of all the supported data types.

Next, indicate which column will be compared with which. By default each requirement will be compared against the column in the student data directly below it. By default the program will compare data like so:

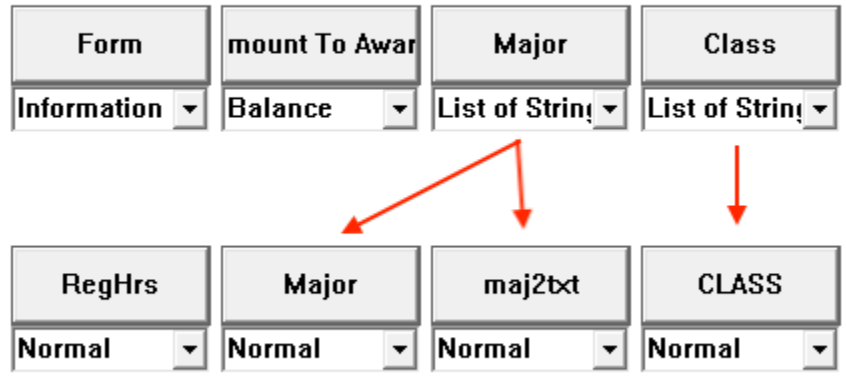


Notice that the Form, and Amount to award columns will not be compared with anything because these are Information data types. They are not requirements, only information about each endowment.

As shown above, the user will want to change which column the “Major” column is compared with. To do this, click the “Major” button, and then the corresponding column on the student’s data. This will result in the comparisons looking like this:



Now the required major on the endowments data is being compared to the column on the student's table that contains their major. However, students could potentially have two majors, so we will want to compare the desired major against both of the student's potential majors. To do this we would click the "Major" button again, and then click the "major2txt" column on the students data. Doing this will produce this:



Now the computer is correctly comparing each column. It will check both the "Major" and the "maj2txt" column on the students to see if there is a matching major.
Continue through all your data to make that each column is going to be compared to the correct corresponding data. It's easiest to line up the columns in the spreadsheets first, but they can be adjusted here as well.

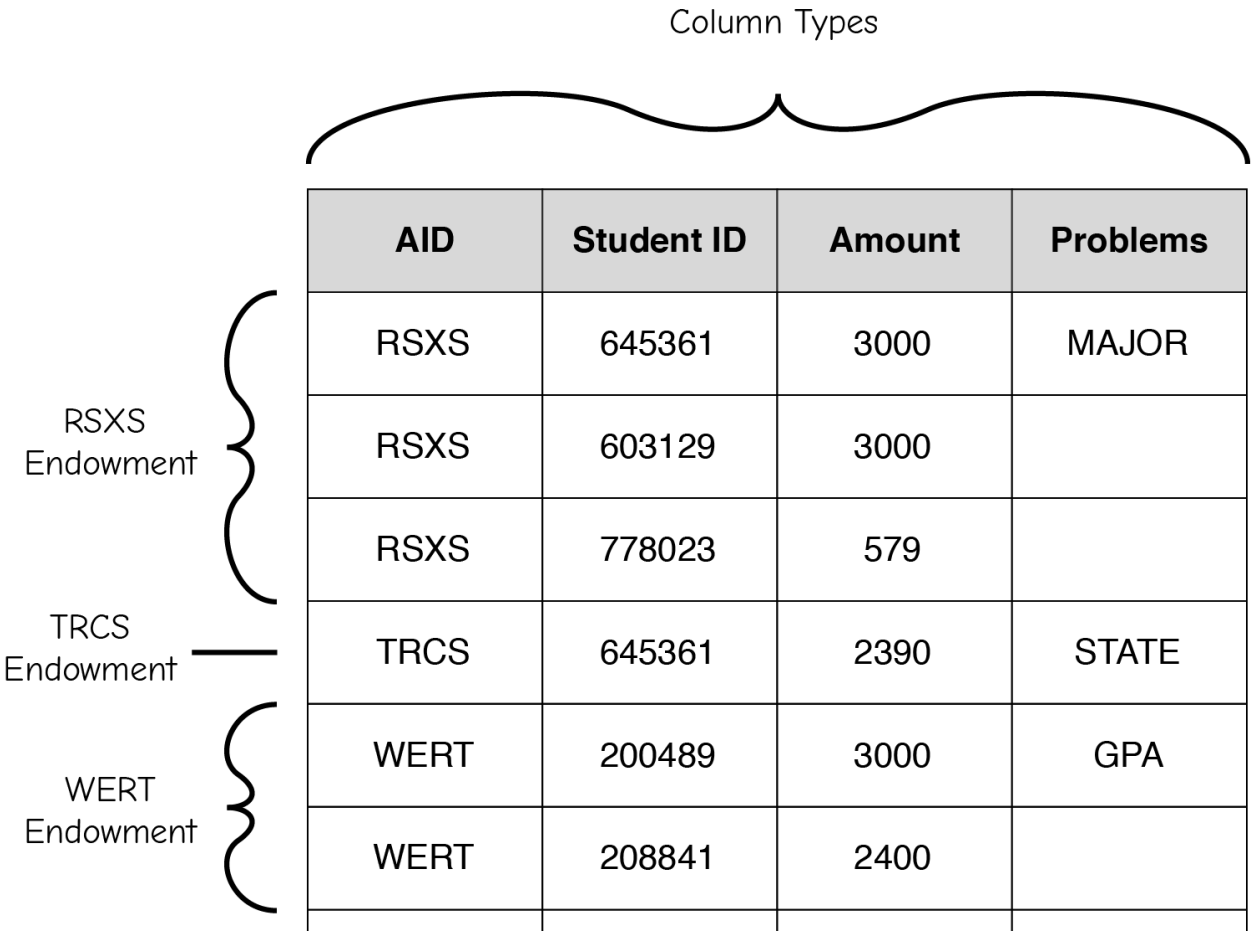
Once both of these tasks have been done, click the "Looks good, Run it!" button which will begin the comparison process.

Results Page.

After a bit of loading, depending on the size of the files, the computer will produce a resulting Excel file. A dialog box will appear asking you to choose a location to save the file. Choose somewhere you can locate the file, and click save. The program has run its course and made its suggestions, now it's time to examine the results.

Understanding Results

The program produces an Excel file with four columns and many rows. The overall purpose of the resulting file is to indicate which students should receive which endowments, and how much money they should get.
The file will look something like this:



AID	Student ID	Amount	Problems
RSXS	645361	3000	MAJOR
RSXS	603129	3000	
RSXS	778023	579	
TRCS	645361	2390	STATE
WERT	200489	3000	GPA
WERT	208841	2400	

The first column indicates the endowment that is being awarded. In the example shown above, each endowment is uniquely identified by a four letter code. This code was taken directly from the endowments spreadsheet that was utilized to create this resulting data page. So whatever uniquely identifies each endowment may look different from university to university.

The second column uniquely identifies each student. Once again this data comes from the student Excel sheet. In this case the university uses student ID numbers to identify each student and they are displayed here.

The third column indicates how much money the program suggests should be awarded to that student.

The final column indicates the title of a column that made it most challenging to find students for that endowment. In the example above The RSXS endowment was most limited by its requirement to find a student with a particular major. This column helps the user determine if there were errors matching students to this endowment. Just because there is a value in the problem column does not mean there was an error. This is just indicating the column that proved the most difficult to find students for.

Utilizing this resulting data file allows the financial aid department to quickly select students to receive money from each endowment.

Appendix B: User Interface

```
#define MAX_LOADSTRING 100

// Global Variables:
HINSTANCE hInst;
int screen = 0;                                 // the number of the screen we
 are on.
WCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
WCHAR szWindowClass[MAX_LOADSTRING];            // the main window class name
HWND activeWindows[200];                         // an array of all active win
dows.
int numberOfActiveWindows = 0;                  // an integer to keep track of
 active windows in the array.
int number_of_columns = 26;
bool adding_to_endowment = false;
int id_of_attribute_to_add = 0;

string end-=owments_finalFilepath;
string students_finalFilepath;

// Forward declarations of functions included in this code module:
ATOM                MyRegisterClass(HINSTANCE hInstance);
BOOL                InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
INT_PTR CALLBACK    About(HWND, UINT, WPARAM, LPARAM);


void OnPaint(HDC hdc)
{
    //this function paints a line from the endowment attribute to the student
attribute that it is compared against.
    if (screen == 1)
    {
         int  x_size = 105; // each endowment attribute will be located at 20
+ 105(id)
        //initialize drawing items
        Gdiplus::Graphics graphics(hdc);
        Gdiplus::Pen      pen(Gdiplus::Color(255, 0, 0, 0)); //black

        //loop through the attributes and draw a line to the correct Student a
ttribute.
        DLLNode<Attribute>* tempAttribute = nullptr;
        all_attributes->getNextNode(tempAttribute);
        while (tempAttribute != nullptr)
        {
            int type = tempAttribute->info.getType();
            if (type < 3) //types 3, 4 & 5 are information types and won't nee
d a line
            {
                //get the ids the attribute is associated to.
                int x_location = (x_size * tempAttribute->info.getId()) + 50;
```

```
                graphics.DrawLine(&pen, x_location, 0, x_location, 100);
            }
            all_attributes->getNextNode(tempAttribute);
        }

    }
}

int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
                      _In_opt_ HINSTANCE hPrevInstance,
                      _In_ LPWSTR    lpCmdLine,
                      _In_ int       nCmdShow)
{
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);

    // TODO: Place code here.
    Gdiplus::GdiplusStartupInput gdiplusStartupInput;
    ULONG_PTR           gdiplusToken;

    // Initialize GDI+.
    GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);

    // Initialize global strings
    LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
    LoadStringW(hInstance, IDC_ENDOWMENTS, szWindowClass, MAX_LOADSTRING);
    MyRegisterClass(hInstance);

    // Perform application initialization:
    if (!InitInstance (hInstance, nCmdShow))
    {
        MessageBox(NULL,
            _T("Call to RegisterClassEx failed!"),
            _T("Windows Desktop Guided Tour"),
            NULL);
        return FALSE;
    }

    HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC_ENDOW
MENTS));

    MSG msg;

    // Main message loop:
    while (GetMessage(&msg, nullptr, 0, 0))
    {
        if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }

    Gdiplus::GdiplusShutdown(gdiplusToken);
    return (int) msg.wParam;
```

```
}



//
//  FUNCTION: MyRegisterClass()
//
//  PURPOSE: Registers the window class.
//
ATOM MyRegisterClass(HINSTANCE the_hInstance)
{
    WNDCLASSEXW wcex;

    wcex.cbSize = sizeof(WNDCLASSEX);

    wcex.style          = CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc    = WndProc;
    wcex.cbClsExtra     = 0;
    wcex.cbWndExtra     = 0;
    wcex.hInstance      = the_hInstance;
    wcex.hIcon          = LoadIcon(the_hInstance, MAKEINTRESOURCE(IDI_ENDOWMEN
TS));
    wcex.hCursor        = LoadCursor(nullptr, IDC_ARROW);
    wcex.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
    wcex.lpszMenuName   = MAKEINTRESOURCEW(IDC_ENDOWMENTS);
    wcex.lpszClassName  = szWindowClass;
    wcex.hIconSm        = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_SMALL))
;

    return RegisterClassExW(&wcex);
}

//
//   FUNCTION: InitInstance(HINSTANCE, int)
//
//   PURPOSE: Saves instance handle and creates main window
//
//   COMMENTS:
//
//        In this function, we save the instance handle in a global variable a
nd
//        create and display the main program window.
//
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
   hInst = hInstance; // Store instance handle in our global variable

   HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
      CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, nullptr, nullptr, hInstance, nullptr
);

   if (!hWnd)
   {
       MessageBox(NULL,
           _T("Call to RegisterClassEx failed!"),
```

```
            _T("Windows Desktop Guided Tour"),
            NULL);
        return FALSE;
    }

    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);

    return TRUE;
}

void redrawScreen(HWND hWnd,bool newScreen)
{
    for (int i = numberOfActiveWindows - 1; i >= 0; i--)
    {
        DestroyWindow(activeWindows[i]);
    }
    numberOfActiveWindows = 0;
    if (newScreen)
    {
        screen++;
    }
    InvalidateRect(hWnd, NULL, TRUE);
}
LPCWSTR basicFileSave()
{
    LPCWSTR filepath = L"";
    HRESULT hr = CoInitializeEx(NULL, COINIT_APARTMENTTHREADED |
        COINIT_DISABLE_OLE1DDE);
    if (SUCCEEDED(hr))
    {
        IFileSaveDialog* pFileSave;

        // Create the FileOpenDialog object.
        hr = CoCreateInstance(CLSID_FileSaveDialog, NULL, CLSCTX_ALL, IID_IFil
eSaveDialog, reinterpret_cast<void**>(&pFileSave));

        if (SUCCEEDED(hr))
        {
            // Show the save dialog box.
            pFileSave->SetDefaultExtension(L"csv");
            hr = pFileSave->Show(NULL);

            // Get the file name from the dialog box.
            if (SUCCEEDED(hr))
            {
                IShellItem* pItem;
                hr = pFileSave->GetResult(&pItem);
                if (SUCCEEDED(hr))
                {
                    PWSTR pszFilePath;
                    hr = pItem-
>GetDisplayName(SIGDN_FILESYSPATH, &pszFilePath);
                    // Display the file name to the user.
                    if (SUCCEEDED(hr))
```

```
                    {
                        filepath = pszFilePath;
                        CoTaskMemFree(pszFilePath);
                    }
                    pItem->Release();
                }
            }
            pFileSave->Release();
        }
        CoUninitialize();
    }
    return filepath;
}
LPCWSTR basicFileOpen()
{
    LPCWSTR filepath = L"";
    HRESULT hr = CoInitializeEx(NULL, COINIT_APARTMENTTHREADED |
        COINIT_DISABLE_OLE1DDE);
    if (SUCCEEDED(hr))
    {
        IFileOpenDialog* pFileOpen;

        // Create the FileOpenDialog object.
        hr = CoCreateInstance(CLSID_FileOpenDialog, NULL, CLSCTX_ALL,IID_IFile
OpenDialog, reinterpret_cast<void**>(&pFileOpen));
        if (SUCCEEDED(hr))
        {
            // Show the Open dialog box.
            hr = pFileOpen->Show(NULL);

            // Get the file name from the dialog box.
            if (SUCCEEDED(hr))
            {
                IShellItem* pItem;
                hr = pFileOpen->GetResult(&pItem);
                if (SUCCEEDED(hr))
                {
                    PWSTR pszFilePath;
                    hr = pItem-
>GetDisplayName(SIGDN_FILESYSPATH, &pszFilePath);
                    // Display the file name to the user.
                    if (SUCCEEDED(hr))
                    {
                        filepath = pszFilePath;
                        CoTaskMemFree(pszFilePath);
                    }
                    pItem->Release();
                }
            }
            pFileOpen->Release();
        }
        CoUninitialize();
    }
    return filepath;
}
```

```
bool bothAreLoaded()
{
    return (IsWindowVisible(activeWindows[3]) && IsWindowVisible(activeWindows
[1]));
}
//
//  FUNCTION: WndProc(HWND, UINT, WPARAM, LPARAM)
//
//  PURPOSE: Processes messages for the main window.
//
//  WM_COMMAND  - process the application menu
//  WM_PAINT    - Paint the main window
//  WM_DESTROY  - post a quit message and return
//
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam
)
{

    SCROLLINFO si;
    int scroll_size = 25;
    static int xClient;
    static int scroll_x_pos;        // current horizontal scrolling position
    switch (message)
    {
    case WM_SIZE:
        {
            xClient = LOWORD(lParam);
        }
        break;
    case WM_HSCROLL:
        {
            if (screen == 1 || screen == 2)
            {
                si.cbSize = sizeof(si);
                si.fMask = SIF_ALL;

                // Save the position for comparison later on.
                GetScrollInfo(hWnd, SB_HORZ, &si);
                scroll_x_pos = si.nPos;
                switch (LOWORD(wParam))
                {
                    // User clicked the left arrow.
                case SB_LINELEFT:
                    si.nPos -= 1;
                    break;

                    // User clicked the right arrow.
                case SB_LINERIGHT:
                    si.nPos += 1;
                    break;

                    // User clicked the scroll bar shaft left of the scroll bo
x.
                case SB_PAGELEFT:
```

```
                    si.nPos -= si.nPage;
                    break;

                    // User clicked the scroll bar shaft right of the scroll b
ox.
                case SB_PAGERIGHT:
                    si.nPos += si.nPage;
                    break;

                    // User dragged the scroll box.
                case SB_THUMBTRACK:
                    si.nPos = si.nTrackPos;
                    break;

                default:
                    break;
                }

                // Set the position and then retrieve it.  Due to adjustments
                // by Windows it may not be the same as the value set.
                si.fMask = SIF_POS;
                SetScrollInfo(hWnd, SB_HORZ, &si, TRUE);
                GetScrollInfo(hWnd, SB_HORZ, &si);

                // If the position has changed, scroll the window.
                if (si.nPos != scroll_x_pos)
                {
                    ScrollWindow(hWnd, scroll_size * (scroll_x_pos - si.nPos),
 0, NULL, NULL);
                }
            }
        }
        break;
    case WM_COMMAND:
        {

            int wmId = LOWORD(wParam);
            int identification = HIWORD(wParam);

            if (identification == BN_CLICKED && lParam != 0)
            {
                if (wmId < number_of_columns) // One of the Endowment Attribut
es was clicked
                {
                    if (adding_to_endowment == false)
                        id_of_attribute_to_add = wmId;
                    adding_to_endowment = !adding_to_endowment;
                }
                if (wmId >= number_of_columns && wmId < 105)
                {
                    if (adding_to_endowment)
                    {
                        //add this wmId to this dude;
                        Attribute * temp_att = all_attributes-
>getNthNode(id_of_attribute_to_add);
```

```
                        temp_att->addTestingId(wmId - number_of_columns);
                        Attribute* student_temp_att = all_student_attributes-
>getNthNode(wmId - number_of_columns);
                        string output = student_temp_att->getTitle();
                        OutputDebugStringA(output.c_str());
                        adding_to_endowment = false;
                    }
                }
                if (wmId == 105) // endowments button selected
                {
                    LPCWSTR filepath = basicFileOpen(); //get filepath using c
ommon item dialog

                    if (filepath != L"")
                    {
                        wstring convertedFilepath = filepath;
                        string finalFilepath(convertedFilepath.begin(), conver
tedFilepath.end());
                        endowments_finalFilepath = finalFilepath;

                        int success = openFile(endowments_finalFilepath,true,t
rue, true);
                        //here is the list of processes that need to be comple
ted.
                        if (success == 0) //successfully entered attributes
                        {
                            //label the attributes as either info or attribute
                            ShowWindow(activeWindows[1], SW_SHOW);
                            ShowWindow(activeWindows[0], SW_HIDE);
                            if (bothAreLoaded())
                            {
                                redrawScreen(hWnd, true);
                            }
                            //labelAttributes(hWnd);
                            //once done run through all the endowments to crea
te them.

                        }

                    }
                }
                else if (wmId == 106) //students button selected.
                {
                    LPCWSTR filepath = basicFileOpen(); //get the filepath fro
m common item dialog

                    if (filepath != L"")
                    {
                        wstring convertedFilepath = filepath;
                        string finalFilepath(convertedFilepath.begin(), conver
tedFilepath.end());
                        students_finalFilepath = finalFilepath;

                        int success = openFile(students_finalFilepath, true, t
rue, false);
```

```
                        if (success == 0)
                        {
                            OutputDebugStringA("cool ");
                            ShowWindow(activeWindows[3], SW_SHOW); //the succe
ss message
                            ShowWindow(activeWindows[2], SW_HIDE); //the butto
n itself.
                            if (bothAreLoaded())
                            {
                                redrawScreen(hWnd, true);
                            }
                        }
                    }
                }
                else if (wmId == 107) // RUN IT BuTTON SELECTED
                {
                    //pop up a loading indicator?
                    //continue opening endowments file.
                    int success = openFile(endowments_finalFilepath, false, fa
lse, true);
                    //continue opening students file.
                    int student_success = openFile(students_finalFilepath, fal
se, false, false);
                    //do comparisions?
                    doComparisons(true);
                    OutputDebugStringA("\nHalfway There\n");
                    //sort the data based on potential students;
                    sortEndowmentsByPotentials();
                    OutputDebugStringA("\nSorted\n");
                    doComparisons(false);

                    LPCWSTR filepath = basicFileSave();
                    wstring convertedFilepath = filepath;
                    string finalFilepath(convertedFilepath.begin(), convertedF
ilepath.end());
                    outputToFile(finalFilepath);


                    OutputDebugStringA("Chaos baby");
                    // ask what data the want to see in the report?
                    // print report?
                    redrawScreen(hWnd, true);
                }
                else if (wmId == 108) // FINISH HIM BUTTON SELECTED.
                {
                    LRESULT endResult = WndProc(hWnd, UINT(WM_CLOSE), NULL, NU
LL);
                }
            }
            else if (identification == CBN_SELCHANGE)
            {
                //the user changed the dropdown on one of the selctions.
                //wmId is the attribute that we need to edit.
                int ItemIndex = SendMessage((HWND)lParam, (UINT)CB_GETCURSEL,
                    (WPARAM)0, (LPARAM)0);
```

```
                //update the attribute.
                Attribute* tempAttribute;
                if (wmId > number_of_columns - 1) //its a students array
                {
                    tempAttribute = all_student_attributes-
>getNthNode(wmId - number_of_columns);
                }
                else {
                    tempAttribute = all_attributes->getNthNode(wmId);
                }
                tempAttribute->setType(ItemIndex);
                //make the selector show the change has been made.
                TCHAR  ListItem[256];
                (TCHAR)SendMessage((HWND)lParam, (UINT)CB_GETLBTEXT,
                    (WPARAM)ItemIndex, (LPARAM)ListItem);
            }
            else {
                return DefWindowProc(hWnd, message, wParam, lParam);
            }
        }
        break;
    case WM_PAINT:
        {
            PAINTSTRUCT ps;
            HDC hdc = BeginPaint(hWnd, &ps);
            RECT rect;
            if (GetWindowRect(hWnd, &rect))
            {
                switch (screen)
                {
                case 0:
                {
                    if (numberOfActiveWindows == 0)
                    {
                        TCHAR endowments_greeting[] = _T("Input the file conta
ining the endowments data:");
                        TextOut(hdc, 10, 10, endowments_greeting, _tcslen(endo
wments_greeting));
                        activeWindows[numberOfActiveWindows++] = CreateWindow(
L"BUTTON", L"Select File", WS_TABSTOP | WS_VISIBLE | WS_CHILD | BS_DEFPUSHBUTT
ON, 10, 40, 100, 30, hWnd, (HMENU)105, (HINSTANCE)GetWindowLongPtr(hWnd, GWLP_
HINSTANCE), NULL);
                        activeWindows[numberOfActiveWindows++] = CreateWindow(
L"STATIC", L"Successfully Opened File", WS_TABSTOP | WS_CHILD, 10, 40, 200, 30
,hWnd , NULL, hInst,NULL);
                        TCHAR students_greeting[] = _T("Input the file contain
g the students data:");
                        TextOut(hdc, 10, 90, students_greeting, _tcslen(studen
ts_greeting));
                        activeWindows[numberOfActiveWindows++] = CreateWindow(
L"BUTTON", L"Select File", WS_TABSTOP | WS_VISIBLE | WS_CHILD | BS_DEFPUSHBUTT
ON, 10, 120, 100, 30, hWnd, (HMENU)106, (HINSTANCE)GetWindowLongPtr(hWnd, GWLP
_HINSTANCE), NULL);
```

```
                        activeWindows[numberOfActiveWindows++] = CreateWindow(
L"STATIC", L"Successfully Opened File", WS_TABSTOP | WS_CHILD, 10, 120, 200, 3
0, hWnd, NULL, hInst, NULL);

                    }
                    break;
                }
                case 1:
                {
                    if (numberOfActiveWindows == 0)
                    {
                        //create the scroll bar
                        int scroll_area = (((all_attributes-
>size() * 105) > (all_student_attributes->size() * 105) ? all_attributes-
>size() * 105 : all_student_attributes->size() * 105));
                        // Set the horizontal scrolling range and page size.
                        si.cbSize = sizeof(si);
                        si.fMask = SIF_RANGE | SIF_PAGE;
                        si.nMin = 0;
                        si.nMax = (scroll_area) / scroll_size;
                        si.nPage = xClient / scroll_size;
                        SetScrollInfo(hWnd, SB_HORZ, &si, TRUE);

                        //create list of options for each item
                        TCHAR type[6][25] = { _T("List of Strings"), _T("Great
er Than or Equal to"), _T("Has Value"), _T("Information"), _T("Low Income Stud
ent"), _T("Balance")};

                        OutputDebugStringA("Case 1 engaging");
                        int x_location = 20;
                        string temp = "";
                        wstring second_temp = L"";
                        LPCWSTR correctTitle = L"";

                        DLLNode<Attribute>* tempAttribute = nullptr;
                        all_attributes->getNextNode(tempAttribute);
                        while (tempAttribute != nullptr)
                        {
                            //create information label
                            int id = tempAttribute->info.getId();
                            temp = tempAttribute->info.getTitle();
                            second_temp = wstring(temp.begin(), temp.end());
                            correctTitle = second_temp.c_str();

                            activeWindows[numberOfActiveWindows++] = CreateWin
dow(L"BUTTON", correctTitle, WS_TABSTOP | WS_CHILD | WS_VISIBLE | BS_DEFPUSHBU
TTON, x_location, 20, 100, 40, hWnd , (HMENU)id, hInst, NULL);
                            activeWindows[numberOfActiveWindows++] = CreateWin
dow(WC_COMBOBOX, TEXT(""), CBS_DROPDOWN | CBS_HASSTRINGS | WS_CHILD | WS_BORDE
R | WS_VISIBLE, x_location, 60, 100, 150, hWnd, (HMENU)id, NULL, NULL);

                            //setup the combo_box
                            for (int i = 0; i < 6; i++)
                            {
```

```
                                SendMessage(activeWindows[numberOfActiveWindow
s - 1], (UINT)CB_ADDSTRING, (WPARAM)0, (LPARAM)type[i]);
                            }
                            SendMessage(activeWindows[numberOfActiveWindows -
1], CB_SETCURSEL, (WPARAM)0, (LPARAM)0);

                            //increment items for next round.
                            x_location += 105;
                            all_attributes->getNextNode(tempAttribute);
                        }

                        //while loop to loop through all students attributes a
nd compare them.
                        TCHAR type2[2][25] = { _T("Normal"), _T("Max Money To
Award")};
                        x_location = 20;
                        temp = "";
                        second_temp = L"";
                        correctTitle = L"";
                        tempAttribute = nullptr;
                        all_student_attributes->getNextNode(tempAttribute);
                        while (tempAttribute != nullptr)
                        {
                            //create information label
                            int id = tempAttribute->info.getId();
                            temp = tempAttribute->info.getTitle();
                            second_temp = wstring(temp.begin(), temp.end());
                            correctTitle = second_temp.c_str();

                            activeWindows[numberOfActiveWindows++] = CreateWin
dow(L"BUTTON", correctTitle, WS_TABSTOP | WS_CHILD | WS_VISIBLE | BS_DEFPUSHBU
TTON, x_location, 140, 100, 40, hWnd, (HMENU)id, hInst, NULL);
                            activeWindows[numberOfActiveWindows++] = CreateWin
dow(WC_COMBOBOX, TEXT(""), CBS_DROPDOWN | CBS_HASSTRINGS | WS_CHILD | WS_BORDE
R | WS_VISIBLE, x_location, 180, 100, 150, hWnd, (HMENU)id, NULL, NULL);

                            //setup the combo_box
                            for (int i = 0; i < 2; i++)
                            {
                                SendMessage(activeWindows[numberOfActiveWindow
s - 1], (UINT)CB_ADDSTRING, (WPARAM)0, (LPARAM)type2[i]);
                            }
                            SendMessage(activeWindows[numberOfActiveWindows -
1], CB_SETCURSEL, (WPARAM)0, (LPARAM)0);
                            //increment items for next round.
                            x_location += 105;
                            all_student_attributes-
>getNextNode(tempAttribute);
                        }
                        activeWindows[numberOfActiveWindows++] = CreateWindow(
L"BUTTON", L"Looks Good, Run it.", WS_TABSTOP | WS_CHILD | WS_VISIBLE | BS_DEF
PUSHBUTTON, 20, 320, 200, 40, hWnd, (HMENU)107, hInst, NULL);
                        //activeWindows[numberOfActiveWindows++] = CreateWindo
w(L"STATIC", NULL, WS_CHILD | WS_VISIBLE, 20, 80, scroll_area, 60, hWnd, NULL,
 hInst, NULL);
```

```
                        /*HWND drawingBoard = activeWindows[numberOfActiveWind
ows - 1];
                        PAINTSTRUCT new_ps;
                        HDC new_hdc = BeginPaint(drawingBoard, &new_ps);
                        OnPaint(new_hdc);
                        EndPaint(drawingBoard, &new_ps);*/

                    }
                    break;
                }
                case 2:
                {
                    if (numberOfActiveWindows == 0)
                    {
                        activeWindows[numberOfActiveWindows++] = CreateWindow(
L"STATIC", L"Outputed to file.", WS_CHILD | WS_VISIBLE , 20, 20, 200, 40, hWnd
, NULL, hInst, NULL);
                        activeWindows[numberOfActiveWindows++] = CreateWindow(
L"BUTTON", L"Quit", WS_CHILD | WS_TABSTOP | WS_VISIBLE | BS_DEFPUSHBUTTON, 20,
 80, 200, 40, hWnd, (HMENU)108, hInst, NULL);

                    }
                    break;
                }
                }
            }
            EndPaint(hWnd, &ps);
        }
        break;
    case WM_DESTROY:
        PostQuitMessage(0);
        break;
    default:
        return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}

// Message handler for about box.
INT_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
    UNREFERENCED_PARAMETER(lParam);
    switch (message)
    {
    case WM_INITDIALOG:
        return (INT_PTR)TRUE;

    case WM_COMMAND:
        if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
        {
            EndDialog(hDlg, LOWORD(wParam));
            return (INT_PTR)TRUE;
        }
        break;
    }
```

```
    return (INT_PTR)FALSE;
}
```

Appendix C: Data Analysis

```
// Endowments.cpp : Defines the entry point for the application.
//
#include "resource.h"
#include "framework.h"
#include <Commdlg.h>
#include <shobjidl.h>
#include <gdiplus.h> // necessary for line drawing only.
#include <string>
#pragma comment (lib,"Gdiplus.lib") //necessary for line drawing only.

#include <iostream>
#include <fstream>
#include <sstream>
#include <algorithm>
#include "Endowment.h"
#include "DoubleyLinkedList.h"
#include "Student.h"
#include "Attribute.h"
#include "potentialStudent.h"

using namespace std;

DoubleyLinkedList<Endowment>* all_endowments = nullptr;
DoubleyLinkedList<Student>* all_students = nullptr;

//the global list of all attributes
DoubleyLinkedList<Attribute>* all_attributes = nullptr;
DoubleyLinkedList<Attribute>* all_student_attributes = nullptr;

void outputToFile(string outputPath)
{

    ofstream outputfile;
    outputfile.open(outputPath, fstream::trunc);
    if (outputfile.is_open())
    {
        outputfile << "Aid,Student ID, Amount To Award, Problem Column" << end
l;
        DLLNode<Endowment> * tempEndowment = nullptr;
        all_endowments->getNextNode(tempEndowment);
        while (tempEndowment != nullptr)
        {
            bool firstTime = true;
            string aid = *(tempEndowment->info.information-
>getNthNode(1)); //cheating.
            if (tempEndowment->info.potentialStudents > 0)
            {
                DLLNode<potentialStudent>* studentInEndowment = nullptr;
                tempEndowment->info.studentsToReceiveAid-
>getNextNode(studentInEndowment);
                while (studentInEndowment != nullptr)
                {
```

```
                    if (firstTime)
                    {
                        outputfile << aid << "," << studentInEndowment-
>info.student->values[0] << "," << studentInEndowment-
>info.amount << "," << tempEndowment-
>info.getColumnThatCausedProblems() << endl;
                        firstTime = false;
                    }
                    else {
                        outputfile << aid << "," << studentInEndowment-
>info.student->values[0] << "," << studentInEndowment->info.amount << endl;
                    }
                    tempEndowment->info.studentsToReceiveAid-
>getNextNode(studentInEndowment);
                }
            }
            else {
                outputfile << aid << ", No Students Found" << ",," << tempEndo
wment->info.getColumnThatCausedProblems() << endl;
            }
            all_endowments->getNextNode(tempEndowment);
        }
    }
    else {
        OutputDebugStringA("Failure to open output file");
    }
}
void sortEndowmentsByPotentials() // this function makes the endowments that h
ave fewer potential students closer to the top so that they receive those stud
ents before they aren't avaliable.
{
    int outsideCounter = 0;
    int insideCounter = 0;
    DLLNode<Endowment>* tempEndowment = nullptr;
    all_endowments->getNextNode(tempEndowment);
    while (tempEndowment != nullptr)
    {
        //OutputDebugStringA(to_string(outsideCounter++).c_str());
        //OutputDebugStringA("\n");
        int lowestPotential = tempEndowment->info.potentialStudents;
        DLLNode<Endowment>* swapper = tempEndowment;
        DLLNode<Endowment>* secondTemp = tempEndowment;
        all_endowments->getNextNode(secondTemp);
        while (secondTemp != nullptr && lowestPotential != 0)
        {
           // OutputDebugStringA(to_string(insideCounter++).c_str());
            //OutputDebugStringA("\n");
            if (secondTemp->info.potentialStudents < lowestPotential)
            {
                lowestPotential = secondTemp->info.potentialStudents;
                swapper = secondTemp;
            }
            all_endowments->getNextNode(secondTemp);
        }
        if (tempEndowment != swapper)
```

```
        {
            all_endowments->node_swap(tempEndowment, swapper);
            tempEndowment = swapper;
        }
        all_endowments->getNextNode(tempEndowment);
    }

}
int howMuchMoneyToGiveStudent(int AmountInEndowment, int AmountNeededForStuden
t, int defaultAmount)
{
    //determine how much money should be subtracted from the endowment and giv
en to the student.
    if (AmountInEndowment > 0 && AmountNeededForStudent > 0)
    {
        if (AmountInEndowment > defaultAmount)
        {
            if (AmountNeededForStudent > defaultAmount)
            {
                return defaultAmount;
            }
            else {
                return AmountNeededForStudent;
            }
        }
        else
        {
            if (AmountInEndowment >= AmountNeededForStudent)
            {
                return AmountNeededForStudent;
            }
            else {
                return AmountInEndowment;
            }
        }
    }
    else
    {
        return 0;
    }
}
bool csvParser(string input, string* array, int arraySize)
{
    bool isEncapsulated = false;
    int previousSplit = 0;
    int arrayCounter = 0;
    int emptySlots = 0;
    for (size_t i = 0; i < input.length(); i++)
    {
        if (arrayCounter < arraySize)
        {
            if (input[i] == '"')
            {
                isEncapsulated = !isEncapsulated;
            }
```

```
            if (input[i] == ',' && isEncapsulated == false)
            {
                //split into array
                if (previousSplit != i)
                {
                    array[arrayCounter] = input.substr(previousSplit, i - prev
iousSplit);
                    arrayCounter++;
                    previousSplit = i + 1;
                }
                else
                {
                    array[arrayCounter] = "";
                    emptySlots++;
                    arrayCounter++;
                    previousSplit++;
                }
            }
            if ((i == input.length() - 1) && (input[i] != ','))
            {
                array[arrayCounter] = input.substr(previousSplit, i - previous
Split);
            }
        }
    }
    if (emptySlots == arrayCounter)
    {
        //we have an empty array, we have collected all the information from t
his sheet.
        return false; // error case.
    }
    return true;
}
string stripSpaces(string input)
{
    string newString = input;
    int count = 0;
    for (size_t i = 0; i < input.size(); i++)
    {
        if (input[i] != ' ' && input[i] != '"') // only copy over characters w
hich are not spaces.
        {
            newString[count++] = input[i]; // post fix increment
        }
    }
    newString.resize(count);
    return newString;
}
string toUpperCase(string input)
{
    string newString = input;
    for (size_t i = 0; i < input.length(); i++) {
        newString[i] = toupper(input[i]);
    }
    return newString;
```

```
}
string processString(string input)
{
    string newString = toUpperCase(input);
    newString = stripSpaces(newString);
    return newString;
}
bool is_number(const string& stringToTest)
{
    if (stringToTest == "")
    {
        return false;
    }
    for (int i = 0; i < stringToTest.length(); i++)
    {
        if (!(isdigit(stringToTest[i])) && stringToTest[i] != '.')
        {
            return false;
        }
    }
    return true;
}
double processDouble(string input, double default_value)
{
    string newString = stripSpaces(input);
    if (newString != "" && is_number(newString))
    {
        return stod(newString);
    }
    else {
        return default_value;
    }


}
void doComparisons(bool getPossible)
{
    //loop through all endowments.
        //loop through all attributues for each endowment.
            //if not info
                //test against all testing cases given the type.

    DLLNode<Endowment>* tempEndowment;
    tempEndowment = nullptr;
    all_endowments->getNextNode(tempEndowment);
    while (tempEndowment != nullptr)
    {
        if (tempEndowment->info.money_in_account > 0)
        {
            DLLNode<Student>* tempStudent;
            tempStudent = nullptr;
            all_students->getNextNode(tempStudent);
            while (tempStudent != nullptr)
            {
                bool Student_success = true;
```

```
                DLLNode<Attribute>* tempAttribute;
                DLLNode<Requirement>* requirementToTest;
                tempAttribute = nullptr;
                requirementToTest = nullptr;
                all_attributes->getNextNode(tempAttribute);
                while (tempAttribute != nullptr && Student_success == true)
                {
                    int type = tempAttribute->info.getType();
                    if (type != 3 && type != 5) // make sure this attrribute i
s not information.
                    {
                        tempEndowment->info.requirements-
>getNextNode(requirementToTest);

                        //was supposed to say: "satisfies" but I'm not fixing
it now.
                        bool statisfys_one_or_the_other = false; // this varia
ble will be true if any one of the possible columns satisfies this requirement
.
                        for (int i = 0; i < tempAttribute-
>info.number_of_testers; i++)
                        {
                            int position = tempAttribute->info.testing_ids[i];
                            if (type == 0) // list of strings
                            {
                                statisfys_one_or_the_other = requirementToTest
->info.containsString(tempStudent->info.getString(position));
                            }
                            else if (type == 1)
                            {
                                statisfys_one_or_the_other = requirementToTest
->info.isGreaterThan(tempStudent->info.getDouble(position));
                            }
                            else if (type == 2)
                            {
                                if (requirementToTest->info.valueRequired())
                                    statisfys_one_or_the_other = tempStudent-
>info.hasValue(position); // if it comes back true this column has a value.
                                else
                                    statisfys_one_or_the_other = true;
                            }
                            else if (type == 4) //EFC Column.
                            {
                                if (requirementToTest-
>info.valueRequired()) // will be true if this endowment needs a low income st
udent.
                                {
                                    string returnedString = tempStudent-
>info.getString(position);
                                    if (is_number(returnedString)) //this not
only prevents errors, but if the student has a "NO FAFSA" they will be conside
red not low income.
                                    {
```

```
                                        //right now we are considering anyone
who has a efc under 20,000 to be low income. 20,000 is the default low income
benchmark.
                                        statisfys_one_or_the_other = (stoi(ret
urnedString) < 20000 ? true : false); //in the future I want 20,000 to be a va
riable setting that the user can set.
                                    }
                                }
                                else {
                                    statisfys_one_or_the_other = true;
                                }
                            }
                            if (statisfys_one_or_the_other)
                            {
                                break;
                            }
                        }
                        if (statisfys_one_or_the_other == false && getPossible
)
                        {
                            requirementToTest->info.causedFailure();
                        }
                        Student_success = statisfys_one_or_the_other;
                    }
                    all_attributes->getNextNode(tempAttribute);
                }
                if (Student_success)
                {
                    if (getPossible)
                    {
                        tempEndowment->info.potentialStudents++;
                    }
                    else {
                        //seems to be working.
                        int amountToAward = howMuchMoneyToGiveStudent(tempEndo
wment->info.money_in_account, tempStudent-
>info.dollars_to_award, 3000); // default amount to award.
                        if (amountToAward > 199) // default minimum value to a
ward.
                        {
                            tempEndowment->info.money_in_account -
= amountToAward;
                            tempStudent->info.dollars_to_award -
= amountToAward;

                            //store this student and the amount we are going t
o give them in the endowment's list of students;
                            potentialStudent new_stu;
                            new_stu.student = &(tempStudent-
>info); //store a pointer to the student's data which already exists in the al
l_students array.
                            new_stu.amount = amountToAward;
                            tempEndowment->info.studentsToReceiveAid-
>append(new_stu);
                        }
```

```
                    }
                }
                all_students->getNextNode(tempStudent);
            }
        }
        all_endowments->getNextNode(tempEndowment);
    }
}
void inputStudentsData(ifstream& infile, bool firstline)
{
    int number_of_rows = 400; // this number has to be precise. I need to dete
rmine it.
    int number_of_columns = 26; //this number has to be precise. Must be deter
mined
    int number_of_endowment_cols = 26;
    int row_position = 0;
    string holder_variable;
    string all_cells_in_string[35];
    bool successful_split = true;
    if (firstline == false) // if we are not reading in the first line, then s
kip it.
    {
        getline(infile, holder_variable);
        holder_variable = "";
    }
    while (infile.good() && row_position < number_of_rows && successful_split)
    {
        getline(infile, holder_variable);
        successful_split = csvParser(holder_variable, all_cells_in_string, num
ber_of_columns);
        if (firstline)
        {
            //iterate over every attribute on this line
            for (int i = 0; i < number_of_columns; i++)
            {
                Attribute next_element;
                next_element.setId(i + number_of_endowment_cols);
                next_element.setTitle(all_cells_in_string[i]);
                all_student_attributes->append(next_element);
            }
            break;
        }
        else if (successful_split) {
            //create a new student to be added to the Students dll
            Student new_student;
            //iterate through the attributes array and add items here.
            int i = 0; // a variable to track where we are in the line as comp
ared to the attributes arrray.
            DLLNode<Attribute>* tempAttribute;
            tempAttribute = nullptr;
            all_student_attributes->getNextNode(tempAttribute);
            while (tempAttribute != nullptr)
            {
                if (tempAttribute-
>info.getType() == 0) // normal data, add it in.
```

```
                {
                    new_student.setValue(processString(all_cells_in_string[i])
);
                }
                else if (tempAttribute-
>info.getType() == 1) // max money to award
                {
                    if (is_number(all_cells_in_string[i]))
                    {
                        new_student.dollars_to_award = stoi(processString(all_
cells_in_string[i]));
                    }
                    new_student.setValue(processString(all_cells_in_string[i])
); // we still need to include this value in the values array otherwise the po
sition gets off by 1.
                }
                i++;
                all_student_attributes->getNextNode(tempAttribute);
            }
            all_students->append(new_student);
        }
    }
}
void inputEndowmentsData(ifstream& infile, bool firstline)
{
    int number_of_rows = 400; // this number has to be precise. I need to dete
rmine it.
    int number_of_columns = 26; //this number has to be precise. Must be deter
mined
    int row_position = 0;
    int successful_split = true;
    string holder_variable;
    string all_cells_in_string[26];
    if (firstline == false) // if we are not reading in the first line, then s
kip it.
    {
        getline(infile, holder_variable);
        holder_variable = "";
    }
    while (infile.good() && row_position < number_of_rows && successful_split)
    {
        getline(infile, holder_variable);
        successful_split = csvParser(holder_variable, all_cells_in_string, num
ber_of_columns);
        if (firstline)
        {
            //iterate over every attribute on this line
            for (int i = 0; i < number_of_columns; i++)
            {
                Attribute next_element;
                next_element.setId(i);
                next_element.setTitle(all_cells_in_string[i]);
                all_attributes->append(next_element);
            }
            break;
```

```
        }
        else if (successful_split) {
            //create a new endowment to be added to the endowments dll
            Endowment new_endowment;
            //iterate through the attributes array and add items here.
            int i = 0; // a variable to track where we are in the line as comp
ared to the attributes arrray.
            DLLNode<Attribute>* tempAttribute;
            tempAttribute = nullptr;
            all_attributes->getNextNode(tempAttribute);
            while (tempAttribute != nullptr)
            {
                //do the stuffs.
                if (tempAttribute->info.getType() == 0)
                {
                    //it's a list of strings.
                    Requirement next_requirement;
                    next_requirement.setTitle(tempAttribute->info.getTitle());
                    next_requirement.createListOfStrings(all_cells_in_string[i
]);
                    new_endowment.requirements->append(next_requirement);
                }
                else if (tempAttribute->info.getType() == 1)
                {
                    //its a greater than-
                    Requirement next_requirement;
                    next_requirement.setTitle(tempAttribute->info.getTitle());
                    next_requirement.setGreaterThanValue(processDouble(all_cel
ls_in_string[i], 2.00));
                    new_endowment.requirements->append(next_requirement);
                }
                else if (tempAttribute->info.getType() == 2)
                {

                    Requirement next_requirement;
                    string newString = processString(all_cells_in_string[i]);
                    if (newString != "" && newString != "ANY")
                    {
                        next_requirement.doesNeedValue(); // if this column is
 not blank, and does not say ANY, then this column does require the student to
 have a value.
                    }
                    new_endowment.requirements->append(next_requirement);

                }
                else if (tempAttribute->info.getType() == 3)
                {
                    //its information appended the string to the info array
                    new_endowment.information->append(all_cells_in_string[i]);
                }
                else if (tempAttribute->info.getType() == 4)
                {
                    //It's the efc column, if this column says 'yes' we will l
ook for a low income student, if no, we won't.
                    Requirement next_requirement;
```

```
                    string newString = processString(all_cells_in_string[i]);
                    if (newString == "YES")
                    {
                        next_requirement.doesNeedValue(); //This was origional
ly intended for has value columbs, but it can be used for this as well.
                    }
                    new_endowment.requirements->append(next_requirement);
                }
                else if (tempAttribute->info.getType() == 5)
                {
                    //this is the endowments balance. Set the balance off of t
his column.
                    string newString = processString(all_cells_in_string[i]);
                    if (is_number(newString))
                    {
                        new_endowment.money_in_account = stoi(newString);
                        if (new_endowment.money_in_account != 0)
                            new_endowment.had_money = true;
                    }
                }
                i++;
                all_attributes->getNextNode(tempAttribute);
            }
            all_endowments->append(new_endowment);
        }
    }

}

int openFile(string filepath, bool firstline, bool firsttime, bool reading_end
owments) {

    if (firsttime) {
        int num_of_columns = 26;
        //26 is the number of columns. I need to find that out here.
        if (all_attributes == nullptr)
            all_attributes = new DoubleyLinkedList<Attribute>[num_of_columns];
        if (reading_endowments)
        {
            //400 is the number of rows ,I need to find that out here first.
            all_endowments = new DoubleyLinkedList<Endowment>[400];
        }
        else
        {
            //create the students array.
            all_student_attributes = new DoubleyLinkedList<Attribute>[num_of_c
olumns];
            all_students = new DoubleyLinkedList<Student>[635];
        }
    }
    //input the endowments stuff.
    ifstream infile;

    infile.open(filepath);
    if (infile)
```

```
    {
        if (reading_endowments) {
            inputEndowmentsData(infile, firstline);
        }
        else {
            inputStudentsData(infile, firstline);
        }
        return 0; //successfully opened the first line of the endowments data.
    }
    else {
        return 1; //error couldn't open filepath.
    }
}

void clearMemory()
{
    delete all_attributes;
    delete all_endowments;
    delete all_students;
}
```

Appendix D: Attributes

```
#pragma once
#ifndef Attribute_h
#define Attribute_h
#include <string>

using namespace std;

class Attribute
{
public:
        Attribute();
        ~Attribute();
        string getTitle();
        int getId();
        int getType();

        void setId(int);
        void setTitle(string);
        void setType(int);
        void addTestingId(int);

        int testing_ids[10]; //an integer which shows which bit of student inf
o this attribute will be compared to.
        int number_of_testers;

private:
        int type;
        string title;
        //The name of this attribute. Ex. Estimated Need, Outside Activities,
etc.
        int id;         //shows the position in the DLL list of attributes
};

Attribute::Attribute()
{
        type = 0;
        title = "";
        id = NULL;
        number_of_testers = 0;
}
string Attribute::getTitle()
{
        return title;
}
int Attribute::getId()
{
        return id;
}
void Attribute::setTitle(string newTitle)
{
        title = newTitle;
}
```

```
void Attribute::setId(int newId)
{
	id = newId;
	if (number_of_testers == 0 && type != 3) // if a tester hasn't been se
t, and this isn't information, set a default testing id.
	{
		testing_ids[number_of_testers++] = id;
	}
}
void Attribute::setType(int newType)
{
	// 0 - list_of_strings
	// 1 - greater than or equal to.
	// 2 - has value;
	// 3 - INFORMATION
	// 4 - EFC < EFC_VALUE
	// 5 - Endowment_balance.
	if (newType >= 0 && newType <= 5)
	{
		type = newType;
	}
	else {
		type = 0;
	}
}
void Attribute::addTestingId(int newId)
{
	if (number_of_testers == 1)
	{

	if (testing_ids[0] == id) // check if the default tester is on.
		{

	testing_ids[0] = newId; // if it is on, replace it with what is assign
ed.
		}
		else {
			testing_ids[number_of_testers++] = newId;
		}
	}
	else {
		testing_ids[number_of_testers++] = newId;
	}

}
int Attribute::getType()
{
	return type;
}
Attribute::~Attribute()
{
	//decontstructer
	//nothing to do here.
}
```

```
#endif
```

Appendix E: Requirements

```
#pragma once
#ifndef REQUIREMENT_H
#define REQUIREMENT_H


class Requirement {
public:
        Requirement();
        ~Requirement();

    //getem
    int getFailures();
    std::string getTitle();

    //setters
    void createListOfStrings(std::string);
    void setGreaterThanValue(double);
    void setTitle(std::string);
    void causedFailure();
    void doesNeedValue();

    //checkem.
    bool isGreaterThan(double);
    bool containsString(std::string);
    bool valueRequired();
private:
        std::string title;
    bool needsValue;
    int causedFailures; // an integer which counts the number of times this at
tribute caused a student to not get a scholarship.
        double greater_than;     //the value the student needs to be greater t
han or equal to in order to match.
        //bool has_value; We don't need a variable, if the attribute is of typ
e 2, then we know what to do. If the cell has a value it matches with this end
owment, otherwise no. The requirements array does not need to be consulted.
        std::string * list_of_strings; // a list of strings which the student
needs to have one of in order ot match.
    int array_size;
};

#endif


#include <string>
#include <sstream>
#include <iostream>
#include "Requirement.h"

using namespace std;

int Requirement::getFailures()
```

```
{
    return causedFailures;
}
void Requirement::setTitle(string new_title)
{
    title = new_title;
}
void Requirement::causedFailure()
{
    causedFailures++;
}
int getArraySize(string input)
{
    int long numberOfOccurences = count(input.begin(), input.end(), '|') + 1;
    return static_cast<int>(numberOfOccurences);
}
string requirementStripSpaces(string input)
{
    string newString = input;
    int count = 0;
    for (size_t i = 0; i < input.size(); i++)
    {
        if (input[i] != ' ' && input[i] != '"') // only copy over characters w
hich are not spaces.
        {
            newString[count++] = input[i]; // post fix increment
        }
    }
    newString.resize(count);
    return newString;
}
string requirementToUpperCase(string input)
{
    string newString = input;
    for (size_t i = 0; i < input.length(); i++) {
        newString[i] = toupper(input[i]);
    }
    return newString;
}
string requirementProcessString(string input)
{
    string newString = requirementToUpperCase(input);
    newString = requirementStripSpaces(newString);
    return newString;
}
void splitByCharacters(string* array, string input, bool replaceBlanksWithAny)
{
    //Often times a colum has multiple options. Therefore the string in that c
olum will have a string with pipe characters, something like this: FR|SO|SR Me
aning Freshmen sophmores, or Seniors. We need to seperate these into an array.
 Arrays are passed by reference, so no need to return them.
    string cleanString = requirementProcessString(input);
    if (cleanString == "" && replaceBlanksWithAny)
    {
        cleanString = "ANY";
```

```
    }
    stringstream ssin(cleanString);
    string item;
    int i = 0;
    while (getline(ssin, item, '|'))
    {
        if (item != "")
        {
            array[i] = item;
            i++;
        }
    }
}

Requirement::Requirement()
{
    title = "";
    list_of_strings = nullptr;
    needsValue = false;
}
Requirement::~Requirement()
{
}
void Requirement::createListOfStrings(string input)
{
    array_size = getArraySize(input);
    list_of_strings = new string[array_size];
    splitByCharacters(list_of_strings, input, true);
}
bool Requirement::isGreaterThan(double test_value)
{
    return (greater_than <= test_value ? true : false);
}
void Requirement::setGreaterThanValue(double new_value)
{
    greater_than = new_value;
}
bool Requirement::containsString(string test_string)
{
    string acceptableAnswer = "ANY";
    if (*(list_of_strings + 0) == acceptableAnswer) // if we accept any answer
 then lets just send it.
    {
        return true;
    }
    //the function used to test if the values match for a student and a Endowm
ent.
    if (test_string != "")
    {
        for (int i = 0; i < array_size; i++)
        {
            if (test_string.find(*(list_of_strings + i)) != string::npos || (l
ist_of_strings + i)->find(test_string) != string::npos)
            {
                return true;
```

```
            }
        }
    }
    return false;
}
void Requirement::doesNeedValue()
{
    needsValue = true;
}
bool Requirement::valueRequired()
{
    return needsValue;
}
std::string Requirement::getTitle()
{
    return title;
}
```

Appendix F: Endowments And Students

```
#pragma once
#ifndef ENDOWMENT_H
#define ENDOWMENT_H

#include <string>
#include "DoubleyLinkedList.h"
#include "Requirement.h"
#include "potentialStudent.h"

class Endowment
{
public:
	inline Endowment();
	inline ~Endowment();

	//I want to use linked lists.
	DoubleyLinkedList<std::string>* information;
	DoubleyLinkedList<Requirement>* requirements;
	DoubleyLinkedList<potentialStudent>* studentsToReceiveAid;

	std::string getColumnThatCausedProblems();

	//potentially store the number of potential students in this array.
	int potentialStudents;
	int money_in_account;
	bool had_money;

};

Endowment::Endowment() {
	information = new DoubleyLinkedList<std::string>;
	requirements = new DoubleyLinkedList<Requirement>;
	studentsToReceiveAid = new DoubleyLinkedList<potentialStudent>;
	potentialStudents = 0;
	money_in_account = 0;
	had_money = false;
}

Endowment::~Endowment() {
}

std::string Endowment::getColumnThatCausedProblems()
{
	std::string problem = "";
	int highestError = 0;
	if (had_money == false)
	{
		return "No money in Account";
	}
	else {
		DLLNode<Requirement>* tempRequirement = nullptr;
		requirements->getNextNode(tempRequirement);
```

```
                while (tempRequirement != nullptr)
                {
                        if (tempRequirement-
>info.getFailures() > highestError)
                        {
                                problem = tempRequirement->info.getTitle();
                                highestError = tempRequirement-
>info.getFailures();
                        }
                        requirements->getNextNode(tempRequirement);
                }
        }
        return problem;
}

#endif
```

```
#pragma once
#ifndef STUDENT_H
#define STUDENT_H
#include <string>
#include "DoubleyLinkedList.h"

//Student will just store strings, we already know what to expect based on the
 endowments attributes array. We don't need to have that information in a sepe
rate array.

class Student {
public:
        Student();
        Student(int);
        ~Student();

        void setValue(std::string);

        std::string getString(int);
        double getDouble(int);
        bool hasValue(int);

        int dollars_to_award;
        std::string* values;
        int position_in_array;
};
#endif
```

```
#pragma once
#ifndef Student_h
#define Student_h
#include "Student.h"

void Student::setValue(std::string newString)
{
        values[position_in_array++] = newString;
```

```
}
std::string Student::getString(int position)
{
        std::string retrieved_Solution = values[position];
        return retrieved_Solution;
}
double Student::getDouble(int position)
{
        double retrieved_Solution = stod(values[position]);
        return retrieved_Solution;
}
bool Student::hasValue(int position)
{
        double retreived_Solution = (values[position] != "" ? true : false);
        return retreived_Solution;
}
Student::Student() {
        values = new std::string[35];
        dollars_to_award = 15000; // default value
        position_in_array = 0;
};
Student::Student(int columns)
{
        values = new std::string[columns];
        dollars_to_award = 15000; // default value
        position_in_array = 0;
}
Student::~Student() {
}
#endif
```