Code Purpose:

The purpose of this code was to model three types of distillation systems: Flash Distillation, Differential Distillation, and Multicomponent Fractional Distillation. The program accepts feed specifications, system conditions, and desired models from the user and then prints the results of the models to the screen.

Compiling and Testing Environment Specifications:

This program is written in C, following the C11 Standard Revision. The program was written, compiled, and tested in Xcode. Xcode is an integrated development environment developed by Apple. The program was compiled using the Apple LLVM 9.0 C/C++/Objective-C compiler for the i386 x86_64 architecture. Finally character encoding was done using UTF-16.

Model Details:

The Flash Distillation Model: calculated the boiling and dew point of the feed at equilibrium. The model also calculated the equilibrium compositions of the vapor and liquid. After calculating the equilibrium results for the system, the model calculated the vaporization temperature, liquid composition, and vapor composition using the Rachford-Rice Equation.

The Differential Distillation Model: was suppose to calculate the composition of the distillate product and the liquid remaining in the system using the Rayleigh Equation. This portion of the code has not been completed yet due to time constraints.

The Multicomponent Fractional Distillation Model: calculated distillate composition, bottoms composition, temperate at the top of the column, temperature at the bottom of the column, number of stages at total reflux, distribution of non-key components, the minimum reflux ratio at total reflux, the number of stages need at a given reflux (default reflux was 1.4X the minimum reflux), and the feed tray location for that reflux. This was modeled using The Fenske, Underwood, Gilliland and Kirkbride equations.

Data Files:

antoine_data.txt: This file contains the Antoine Constants: A, B, and C for over 5000 chemicals. It also contains the temperature range over which these constants are valid. The units for these constants are Celsius and mmHg.

azeotrope_data.txt: This file contains data of most known binary azeotropes. Included in the file are the normal boiling points of the two components, the boiling point of the azeotrope, and the azeotrope composition.

Model Assumptions:

All of these models require vapor-liquid equilibrium data for all the chemical components in the system, at the system's operating pressures and temperatures. Since the user input the chemical components of the feed, a wide range of feed chemical compositions were possible. As

such it was impractical to store vapor-liquid equilibrium data for all possible combinations of all chemicals. To solve this problem the antoine constants: A, B, and C for over 5,000 different chemicals were saved in *antoine_data.txt*. This file also included the temperature range, in Celsius, over which the constants were valid.

The Antoine Equation then used the antoine constants to calculate the saturated pressure data for the system. With the saturated pressure data Raoult's Law was used to calculate the vapor-liquid equilibrium data for the system. Using these equations the number of possible mixtures that could be modeled increased dramatically, while also reducing the amount of data that had to be stored in advance. However, these equations assume the mixture is an ideal mixture. In numerous real world mixtures there is a slight deviation from ideal behavior. This decreasing in accuracy was deemed acceptable in exchange for the increased modeling options.

In mixtures that contain azeotropes, the mixture's variation from ideal conditions can be significant at some pressures and temperatures. The stored azeotrope data was used to check the feed mixture for azeotropes. If an azeotrope was detected a warning was printed out with the model results. The warning informed the user of the azeotrope components and that the model results would likely deviate significantly from real world conditions.

Validation:

This program was numerically validated for accuracy and precision. The models in this program were also validated for real world applicability for non-azeotropic mixtures.

Numerical Precision: The precision of this program was validated by running the program repeatedly, ensure identical results were being computed for identical inputs. Also, calculations were performed using double-precision floating-point format numbers with significant figures being reported in such a way as to eliminate round off and truncation errors.

Numerical Accuracy: The accuracy of this program was tested by using the antoine data provided by the program to calculate the normal pure boiling points of the mixture's components. These results were then compared to literature values to ensure the antoine data and antoine calculations were producing accurate results. Accuracy was also validated again during model validation.

Model Validation: The models were validated for real world applicability when the system contained a non-azeotropic mixture by repeating the calculations in Excel using real vapor-liquid equilibrium data that was not obtained from the antoine equation. This ensured the assumption that non-azeotropic mixtures could be treated like ideal mixtures was valid. Finally Dr. Eric Maase of The University of Massachusetts Lowell repeated the calculations using real vapor-liquid equilibrium data. This served to validate the model's applicability to the real world again and ensure the accuracy of the recalculated results done in Excel.

Validation Results Comparison:

Liquid Equilibrium	Program	Excel	Dr. Maase
Dew Temp (Celsius)	99.9	99.97	100
1-Butanol (Mass Fraction)	0.69	0.687	0.687
1-Propanol (Mass Fraction)	0.14	0.135	0.136
Ethanol (Mass Fraction)	0.09	0.089	0.089
Methanol (Mass Fraction)	0.09	0.087	0.088

Vapor Equilibrium	Program	Excel	Maase
Boiling Temp (Celsius)	82.79	82.95	83
1-Butanol (Mass Fraction)	0.09	0.087	0.086
1-Propanol (Mass Fraction)	0.08	0.084	0.084
Ethanol (Mass Fraction)	0.24	0.240	0.241
Methanol (Mass Fraction)	0.59	0.587	0.589

Vaporization	Program Liquid	Excel Liquid	Program Vapor	Excel Vapor
Vaporization Temperature (Celsius)	89.37	88.8		
1-Butanol (Mass Fraction)	0.48	0.486	0.16	0.145
1-Propanol (Mass Fraction)	0.17	0.165	0.12	0.126
Ethanol (Mass Fraction)	0.17	0.162	0.25	0.256
Methanol (Mass Fraction)	0.19	0.187	0.47	0.469

Multicomponent Fractional	Program	Excel	Maase
Temperature Top (Celsius)	65.17	65.78	65.5

Temperature Bottom (Celsius)	95.76	94.34	94.3
Minimum Reflux (Rmin)	2.27	2.19	2.20
Minimum Number Of Stage (Nmin)	**11.41**	**9.15**	**9.21**
Number Of Stage For Reflux = 3.0	17.39	16.82	16.2
Feed Tray Number	8.28	8.88	8.6

There was a significant variation in Minimum Number Of Stage (Nmin) results given how closely the other values aligned. This variations ended up being the result of a programing error in:

File: fractional.c
Function: void fractional_user(void)

This error could not be corrected due to time constraints and the suboptimal structure of this program. However, the minimum number of stages for a system serves no real world purpose because it is only applicable during total reflux; a condition that results in a column unable to produce product. So while Nmin values are incorrect the real world applications for this program are largely unaffected.

Current Program Flaws And Future Improvements:

C is an early procedural programming language with manual memory management. Additionally, the C Standard Library provides limited built in functions compared to more modern languages such as PHP and Python. As such, programs written in C tend to be much long and the programmer is significantly more likely to make memory management errors.

The chance of errors can be reduced by practicing C Coding Conventions that create code that is well indented, spaced, and commented. Additionally, by creating a modular code where individual tasks are split into functions and groups of functions that serve the same purpose are placed in separate source files, code becomes easier to maintain.

After reading Clean Code: A Handbook of Agile Software Craftsmanship, by Robert C. Martin I learned all functions should only perform one action and generally be no longer than ten lines. I also learned that global variables should be avoided whenever possible. Functions in this program perform too many actions and numerous global variables were used making the code difficult to debug and modify later. Future versions of this code will have smaller functions and no global variables.