**Arnav Chandan**

**T00630177**

**COMP 3141**

**Final Project Documentation**

**Phase 1:**

To move forward with this phase of the final project, I have decided to build 3 C++ classes and 2 header files.

BasicCalculator.h //Header file for basic calculator functions

BasicCalculator.cpp //C++ file for basic calculator functions

ScientificCalculator.cpp //C++ file for scientific calculator functions

ScientificCalculator.h //Header file for scientific calculator functions

Main.cpp //Driver file displaying the menu.

**Below is a list of functions / constructors included in each file:**

Basic Calculator:

* Addition
* Subtraction
* Multiplication
* Division
* Square Root
* Square
* Constructor: BasicCalculator()
* Destructor: ~BasicCalculator()
* Accessors: getNum1(), getNum2
* Mutators: setNum1, setNum2

Scientific Calculator:

* Sin
* Sin Inverse
* Cos
* Cos Inverse
* Tan
* Tan Inverse
* Log
* Log with Base 10
* Exponent
* Constructor: ScientificCalculator()
* Destructor: ~ScientificCalculator()
* Accessors: getX(), getY()
* Mutators: setx(), setY()

**OOPS Principles Used:**

BasicCalculator.cpp:

* Encapsulation: The class `BasicCalculator` encapsulates the data members and member functions within its scope. The data members are private, and their access is controlled through public member functions (accessors and mutators).
* Abstraction: The class provides an abstraction by hiding the internal implementation details of the calculator operations. The user interacts with the calculator through the provided member functions without needing to know the underlying logic.
* Modularity: The code follows the modular approach by encapsulating related data and functions within a class. This allows for better organization, reusability, and maintainability of code.

ScientificCalculator.cpp:

* Encapsulation: The class `ScientificCalculator` encapsulates the data member and member functions within its scope. The data member is private, and its access is controlled through public member functions (accessor and mutator).
* Abstraction: The class provides an abstraction by hiding the internal implementation details of the scientific calculator operations. Users interact with the calculator through the provided member functions without needing to know the underlying logic.
* Modularity: The code follows a modular approach by encapsulating related data and functions within a class.