main.py

```
from Project02.implicit_equation import ImplicitEquation
from Project02.parametric_equation import ParametricEquation

FILE = "CS2300P2Windows"



def main():
    equations = []
    with open(FILE) as file:
        # Read two equations from the file, switching based on the type
        for i in range(2):
            if get_next_char(file) == 'i':
                equations.append(
                    ImplicitEquation(
                        get_float(file), get_float(file), get_float(file),
                        [get_point(file), get_point(file), get_point(file)]
                    )
                )
            else:
                equations.append(
                    ParametricEquation(get_point(file), get_point(file),
                                       [get_point(file), get_point(file), get_point(file)]
                                       )
                )

    # Iterate over the gathered data and print the summary
    for count, equation in enumerate(equations):
```

```
                print(f"Equation {count + 1}")
        print("----------")
        equation.print_summary()
        print("\n\n")


# The three functions below make the reading and parsing of the files easier
def get_point(file) -> (float, float):
    return get_float(file), get_float(file)


def get_float(file) -> float:
    return float(get_next_char(file))


def get_next_char(file, prev_chars=""):
    """
    Recursive function to read in each set of characters.
    If no character or whitespace is read, return the read characters or keep reading (if no characters are read)
    :param file: file
    :param prev_chars: str
    :return: str
    """
    char = file.read(1)

    if not len(char) == 0 and char not in [None, " ", "\n"]:
        return get_next_char(file, prev_chars + char)
return prev_chars if prev_chars != "" else get_next_char(file)



if __name__ == '__main__':
    main()
```

implicit_equation.py

```
from math import sqrt


class ImplicitEquation:
    def __init__(self, a, b, c, points):
        self.a = a
        self.b = b
        self.c = c
        self.points = points

    def print_distance_from_points(self):
        """
        This function iterates over the points supplied. For each point it calculates the distance away
        and prints the stats about it
        """
        norm = sqrt(pow(self.a, 2) + pow(self.b, 2))

        for point in self.points:
            dist = ((point[0] * self.a) + (point[1] * self.b) + self.c) / norm

            print(f"Distance from point [{point[0]: .1f}, {point[1]: .1f}] to the line is {dist: .1f}."
                  f"{' The point is on the line.' if dist == 0 else ''}")


    def print_implicit_form(self):
        """
        This is the easiest one to "compute". All the numbers have already been supplied
        """
```

```
          print(f"Implicit Form: {self.a: .1f}a + {self.b: .1f}b + {self.c: .1f} = 0")

def print_parameter_form(self):
    """
    Convert implicit to parametric and print!
    """
    point1 = (0.0, (self.c * -1) / self.b) if abs(self.b) > abs(self.a) else ((self.c * -1) / self.a, 0.0)


    print(f"Parameter form: l(t) = [{point1[0]: .1f}, {point1[1]: .1f}] + t[{self.b: .1f}, {self.a * -1: .1f}]")

def print_point_normal_form(self):
    """
    Calculates the point normal form and prints it out
    """
    print(
        f"Point Normal Form: "
        f"{self.a / abs(self.c): .1f}a + {self.b / abs(self.c): .1f}b + {self.c / abs(self.c): .1f} = 0"
    )

def print_summary(self):
    """
    Makes my main function cleaner.
    """
    self.print_implicit_form()
    self.print_parameter_form()
    self.print_point_normal_form()
    self.print_distance_from_points()
```

parametric_equation.py

```
from Project02.implicit_equation import ImplicitEquation


class ParametricEquation(ImplicitEquation):
    """
    This class is needed because multiple constructors are not allowed in python ://
    All this does is convert the given parametric equation to implicit form, then pass
    it up the chain to the implicit class
    """
    def __init__(self, p: (float, float), v: (float, float), points):
        self.p = p
        self.v = v

        a = self.v[1] * -1, self.v[0]
        c = (-1 * a[0] * self.p[0]) - (a[1] * self.p[1])
        super(ParametricEquation, self).__init__(a[0], a[1], c, points)

    def print_parameter_form(self):
        """
        Override the other parameter form method so that we get the original from the file rather
        than a different equation for the same line
        """
        print(f"Parameter form: l(t) = [{self.p[0]: .1f}, {self.p[1]: .1f}] + t[{self.v[0]: .1f}, {self.v[1]: .1f}]")
```