Lab A: Caesar cipher

Caesar cipher is a simple method of encryption. In a Caesar cipher each letter in a word is replaced by a letter some specified number of positions down the alphabet.
e.x: "hello" shifted 1 letter right becomes "ifmmp"
"hello" shifted 2 letters right becomes "jgnnq"

(a) In main, create an array called alphabet which contains all 26 letters of the English alphabet.

Write a function called letter_index that takes an array of letters, the size of the array and a letter. The letter_index function should return the index of the letter in the array.

letter_index has the following signature:

```c
int letter_index(char * alphabet, int alphabet_size, char letter)
```

(b) Write a function encrypt that takes an array of alphabet letters, an array of letters in a message, and the number of places to RIGHT shift the message. encrypt should shift the letters in the message and output the new message.
e.x: "hello" shifted 2 letters with encrypt should output "jgnnq"

encrypt has the following signature:

```c
void encrypt(char * alphabet, int alphabet_size, char * arr, int size, int shift)
```

(c) Write a function decrypt. decrypt is the same as encrypt except decrypt shifts LEFT.
e.x: "jgnnq" shifted 2 letters with decrypt should output "hello"

decrypt has the following signature:

```c
void decrypt(char * alphabet, int alphabet_size, char * arr, int size, int shift)
```
Solution:

```c
int letter_index(char * alphabet, int alphabet_size, char letter) {
    for (int i=0; i<alphabet_size; i=i+1)
        if (alphabet[i]==letter)
            return i;
}

void encrypt(char * alphabet, int alphabet_size, char * arr, int size, int shift) {
    for (int i=0; i<size; i=i+1) {
        int index=letter_index(alphabet, alphabet_size, arr[i]);
        index=(index+shift)%alphabet_size;
        arr[i]=alphabet[index];
    }
}

void decrypt(char * alphabet, int alphabet_size, char * arr, int size, int shift) {
    for (int i=0; i<size; i=i+1) {
        int index=letter_index(alphabet, alphabet_size, arr[i]);
        index=(index-shift+alphabet_size)%alphabet_size; // avoid negative,
        arr[i]=alphabet[index]; // assume shift<alphabet_size
    }
}

int main() {
    char a[26]={'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z'};
    char m[]="hello";
    encrypt(alphabet, 26, m, 5, 2);
    cout<<m<<'\n';
    decrypt(alphabet, 26, m, 5, 2);
    cout<<m<<'\n';
}
```
Lab B: Point

Point is a class which we define below. An object of class Point or an instance of class Point represents a point on an x-y plane. A Point object has a x-coordinate: x\_coord and a y-coordinate: y\_coord

class Point {
    double x\_coord;
    double y\_coord;
    public:
        Constructors:
        Point() { x\_coord = 0; y\_coord = 0; }
        Point(double x, double y) { ... } // (a)
        double x() {return x\_coord;}
        double y() {return y\_coord;}
        void translate(double x, double y) { ... } // (b)
    };

(a) Complete the incomplete constructor.

(b) Complete the translate function.
To translate a point is to move it a specified amount along the x-axis and/or the y-axis.
E.x: The point(3,5) translated (-1, 2) becomes (3-1, 5+2) = (2, 7)

(c) Write a function, that takes a line segment represented as two points, and returns a point which is the middle of the segment.

(d) Write a function, that takes two points, and returns the distance between the points.
Given two points \((x_1, y_1)\) and \((x_2, y_2)\), the distance between these points is given by the formula:

\[
d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}
\]
Solution:

class Point {
    double x_coord;
    double y_coord;

public:
    Constructors Point() {
        x_coord = 0;
        y_coord = 0;
    }

    Point(double x, double y) {
        x_coord=x;
        y_coord=y;
    }

    double x() {
        return x_coord;
    }

    double y() {
        return y_coord;
    }

    void translate(double x, double y) {
        x_coord=x_coord+x;
        y_coord=y_coord+y;
    }
};

Point mid(Point a, Point b) {
    return Point((a.x()+b.x())/2, (a.y()+b.y())/2);
}

double square(double a) {
    return a*a;
}

double distance(Point a, Point b) {
    return sqrt(square(a.x()-b.x())+square(a.y()-b.y()));
}