Objective:

- Object initialization.
- Writing wrapper methods and basic functionalities for different ADTs.
- And a bit of logic to keep your brains working ☺

Challenge-A: *Car* (4)

Write a class named 'Car' that has the following member variables:

- *yearModel*: An int that holds the car's year model.
- *make*: A char array that holds the make of the car. E.g., Toyota etc.
- *speed*: An int that holds the car's current speed.
- *maxSpeed*: An int that holds the car's maximum possible speed.

In addition, the class should have the following member functions.

- An appropriate constructor to initialize the *yearModel*, *make*, *maxSpeed*, and *speed* as per user provided values.
- *Accessor Methods*: Appropriate accessor functions to get the values stored in an object's *yearModel*, *make*, *maxSpeed*, and *speed* member variables.
- *accelerate*: The accelerate function should add 5 to the speed member variable each time it is called.
- *brake*: The brake function should subtract 5 from the speed member variable each time it is called.

Demonstrate the class in a program that creates a Car object, and then calls the accelerate function five times. After each call to the accelerate function, get the current speed of the car and display it. Then, call the brake function five times. After each call to the brake function, get the current speed of the car and display it.

Note: while applying brake, the current speed can't be less than zero and while accelerate, current car speed can't go above max speed. Max speed should be greater than or equal to 100 and less than or equal to 380.

Challenge-B: *Unit Conversion* (7(1,2,2,2))

For this task, you'll write a C++ Object Oriented program (based on what we learned so far) to do *unit conversion* -- That is, your program should be able to answer the common question (and many more like it): *How many meters in 10 miles?* (Answer: 10 miles = 16,093.44 m)

What units do you need to support?

We would like you to support units for *weights* as follows:

Unit categories	*Units you must support*
distance	**m** (meters), **cm** (centimeters), **mm** (millimeters), **km** (kilometers), **in** (inches), **ft** (feet), **yd** (yards), **mi** (miles)

You should be able to convert any amount of a unit distance to any other unit distance.

Where to find the conversion constants

Following table may help you for conversion.

Distance	
1 m	100 cm
1 cm	1 cm
1 mm	0.1 cm
1 km	100000 cm
1 in	2.54 cm
1 ft	30.48 cm
1 yd	91.44 cm
1 mi	160934 cm

Hints

How should you store all of the multiplying factors between units? There's the hard way and the easy way to think about it.

→ The hard way

The hard way is to build an internal two-dimensional table of the conversion of every unit to every other unit. It would be really painful. Imagine how many translators the United Nations would need if they had 1,000 people speaking different languages and they used this model...a million people!

→ The easy way

The easy way is to find a common, base unit (e.g., meters for distances) that is used to convert from the base unit to that unit. To perform conversion between two units (neither of which are the base unit), you simply convert from your SOURCE_UNIT to the BASE_UNIT, then from that to the DESTINATION_UNIT. Using this analogy, you'd only need to hire 1,000 translators at the UN -- as long as each could translate to and from their language to a common language.

On the basis of scenario explained above, following class design should be preferred. Your job is to provide definition for the member functions in it.

DistanceUnitConversion Class

```
enum DistanceUnitSymbols {M, CM, MM, KM, IN, FT, YD, MI};
class DistanceUnitsConversion
{
public:
    DistanceUnitsConversion();
        //initialize 'baseUnit' and 'baseUnitConversionTable'

    float convert(DistanceUnitSymbols source, DistanceUnitSymbols target, float dist);
        //it is the main function which client will use i.e., it receives the source and target units and
        //distance value and returns the converted distance unit

private:
    float convertToBaseUnit(DistanceUnitSymbols symb, float dist);
        //It's a utility function which will help the 'convert' function to achieve its objective. It
        converts the received distance unit to base unit

    float convertFromBaseUnit(DistanceUnitSymbols symb, float dist);
        //It's a utility function which will help the 'convert' function to achieve its objective. It
        converts the received base distance unit to target unit.

    DistanceUnitSymbols baseUnit;       //represents the base unit
    float baseUnitConversionTable[8];   //contains the standard conversion of each unit against base unit.

};
```

Sample Run

```
int main()
{
    DistanceUnitsConversion duc;
    duc.initializeUnitsTable();
    cout<<duc.convert(KM, M, 3); // prints 3000
    cout<<'\n';
    return 0;
}
```

speed = 150

MaxSpeed = 300

= 250

Commenting your code is like cleaning your bathroom - you never want to do it, but it really does create a more pleasant experience for you and your guests.