It is time to learn about the power of polymorphism with class inheritance. To that end, you are being tasked with implementing a multi-robot system. Obviously, a system this complex must be completed in multiple parts or milestones.

The first milestone is a baseline robot class along with one extension to a tracked robot. The baseline robot represents one of many and so must have a guaranteed unique name. We are using tracked robots because they are easier to "steer" than a robot which moves more like a car—using Ackermann steering.

This assignment is worth a total of 10 points:

- Up to 6 points for successful compilation and correctness,
- Up to 3 points for correctly following the Google style guide, and
- Up to 1 point for correct documentation.

Directory

You will notice that I have provided you with the following directory structure:

- **bin**: To contain executable binary files.
- **build**: To hold the intermediary .o object files.
- **include**: All .h header files will be located here.
- **lib**: A directory of useful classes for your project, provided by me. Normally this directory would hold true library archives which would get compiled into linkable libraries. This is outside of the scope of this class. I have provided you a makefile for this directory and you may make all the .o files you need by running "make library" from either the root directory `03hw/`, or the lib directory `03hw/lib/`.
- **src**: All .cc source files will be located here.
- **test**: Directory to hold the unit test source files which you will, hopefully, write to test your classes before they are submitted.

MobileRobot:

Instances of the MobileRobot class represent something that can move, without describing how it moves. At this point, the only representation we care about is that it has a max speed stored as a double precision floating point value, and a unique id stored as a C++ string object.

- **Constructor(s)**: All mobile robots must know their max speed; therefore, MobileRobot shall have a default constructor which assumes speed is 0.0 units/time and a constructor which accepts speed as a double precision floating point value. The constructor must assert that speed is non-negative.
- **Destructor**: After a robot falls out of scope, its id shall no longer by non-unique.
- **Accessor**: Because different mobile robots may have different max speeds, create an accessor named **speed** which returns the mobile robot instance's speed as a floating point value. This method *should* promise the compiler it does not change any members' value.

- **Accessor**: Mobile robots need some means by which to communicate if they are to coordinate or take direction from a controller. This means they must each have a unique name. Create a method called **id** which returns an instance's unique name as a string from the C++ string library. The method *should* return the name as an r-val and promise the compiler it will not change members' values.
- **Mutator**: A user may wish to assign a name to a robot. The method **id** shall accept a string from the C++ string library and return true if the string represents a unique name, false otherwise. If the name is unique, the mutator shall assign the string to the robot instance's id member variable. Do not forget to set the robot's previous name as non-unique.
- **IsUnique**: A good user might want to check whether a name is unique for attempting to assign it to a robot instance. Provide users of your class with a **static** method named **IsUnique** which accepts a string from the C++ string library and returns true if the string has not yet been assigned to a robot.

Tracked:

With the Tracked class, we extend the functionality of the MobileRobot class by providing a means of translation between locations. We may now represent our location as a point in two-space. We shall also be able to query the robot about whether it can move from its current location to a desired location in a single step.

- **Constructor(s)**: Tracked robots need to know their location and shall make no assumptions about such. There is no default constructor for a Tracked Robot. There are two ways to provide location to a Tracked instance,

  1. one uses a Point class instance from the `lib/include/point.h` library and
  2. the other uses a pair of double precision floating points.

  In either form, an additional double may be provided to set some initial speed. If no initial speed is provided, it is 0.0.
- **Destructor**: After a robot falls out of scope, its id shall no longer by non-unique.
- **Accessor**: Tracked robots shall be instantiated where they start and shall only move under their own power, this means there shall be an accessor with id **location** which returns a **Point** class instance from the `csce240::two_dim` namespace, but not mutator to set the location.
- **CanTranslateTo**: Before a robot is assigned a location to which it should translate, it should be established whether the robot can do so in a single step. The Tracked class shall have a method named **CanTranslateTo** which accepts a **Point** class instance from the `csce240::two_dim` namespace and returns true when that point is no further than **speed** distance from its location. You might look at the Vector class in the `csce240::two_dim` namespace from `lib/include/vector.h` lib

Submission

: To get credit, you must upload a zipped (not tarred, gzipped, or 7z) archive containing your submission files in the the directory tree provided. Given the difficulty I have seen with so many student unable to successfully create and archive files, you will archive **ONLY** the directory structure which I provided you.

Late submissions will be handled as per syllabus. I am attempting to fit four more assignments into this semester so that struggling students have a chance to bring up their grade; consequently, there will be no extensions.

The End.