Design Decisions

To create horses with different strategies I used an abstract base class Horse with three derived classes EarlySprintStrategy, SteadyRunStrategy, and SlowStartStrategy. Using an abstract base class allowed me to use a virtual function move() which was inherited in the derived classes of my superclass. This allowed me to define all information that was common to any Horse object in the Horse class and use my derived classes to specify different uses of the move() function that varied based on strategy type. This means that strategy type was set in these derived classes and they simply held a constructor and a definition of move(). Each move() function defines the strategy of its respective class. The base class holds the name, maxSpeed, and distance (travelled) of each horse. It also holds a constructor, get/set methods for its private variables, and virtual function move(). I also used a Race class to manage running the race with 5 user specified horses. The Race class has one data object raceHorses that is an array of 5 Horse objects (the fixed length array was used as there should always be exactly 5 horses). It contains methods: enrollHorse(), which takes the user input data for one Horse and enters the Horse object into the raceHorses array, resetStrategy(), which resets the run strategy of a previously created Horse object, runRace(), which simulates the horse race by updating the horse distances in a loop until one passes 10 miles and keeping track of who’s in the lead, printDistances(), which prints the distance travelled of each horse, and announceWinner(), which takes the winning horse’s name as input and prints it to the user. In the main class to create a race I simply had to create a Race object, use enrollHorses() to enroll 5 horses in the race, and then use runRace() to find the winner of the race.