CS3210 – 003
HW3 – Functional Programming Languages - Lisp
Due: 6 pm Friday March 16 before Spring Break

The purpose of HW3 is for you to gain experience with writing and testing relatively simple procedures in LISP. For each problem below, include your code (with identification of the problem number being solved), as well as comments and explanations of your code, **and** demonstrate your code's functionality against a set of test cases. Create and include your own additional, meaningful test cases to ensure that your code works not only on typical inputs, but also on "boundary" or difficult cases. Get in the habit of writing and running these test cases after **every** procedure you write – no matter how trivial the procedure may seem to you.

There are utilities and skeleton code for each of the problems below in file "baseball.lisp"

Problem 1: Some simple physics

We are going to begin by modeling how far a baseball can travel – the same physics will hold for both hitting a ball and throwing a ball. We are going to simplify things by assuming that baseballs don't spin as they move (clearly false but it makes life much easier). This means we can treat the movement of a baseball as if it were restricted to a two-dimensional plane. What happens when a baseball is hit? For the moment, we'll model a baseball as a particle that moves along a single dimension with some initial position *u*, some initial velocity *v*, and some initial acceleration *a*, as shown below:

The equation for the position of the baseball at time *t*, given *a*, *v*, and *u* is $ut = ½\ at^2 + v\ t + u$.

For those of you not math majors, this is a first order differential equation in time. This equation can be applied to either the horizontal (*x*) component of baseball motion, or the vertical (*y*) component of baseball motion.

Write a procedure that takes as input values for `a`, `v`, `u`, and `tm` and returns as output the position of the baseball at time `tm`.

```
(defun position (a v u tm)
    YOUR-CODE-HERE))
```

Test your position code for at least the following cases:

code	answer
`(position 0 0 0 0)`	0
`(position 0 0 20 0)`	20
`(position 0 5 10 10)`	60
`(position 2 2 2 2)`	?
`(position 5 5 5 5)`	?

The test file `basebot.scm` will have these tests, and other test cases for other procedures, which you should run (you can add/show your output values). In addition, you should add some test cases of your own to these to cover other boundary and typical conditions.

Problem 2: Basic Math

One of the goals is to determine how far a baseball will travel in the air, if it is hit with some initial velocity at some initial angle with respect to the ground. To do this, we will need to know when the baseball hits the ground, and for that we'll want to find when the *y* coordinate of the baseball's position reaches zero. This can be discovered by finding the roots of the *y* position equation, and selecting the one that is larger (later in time). The proper tool for this is the quadratic formula. Given the coefficients of the quadratic equation $ax^2 + bx + c = 0$, write a function to find one of the roots (`root1`) and another function to find the other root (`root2`)

```
 (defun root1 (a b c)
      YOUR-CODE-HERE))

(defun root2 (a b c)
      YOUR-CODE-HERE))
```

Notice that depending on how you wrote your functions, for some test cases you get an error. For example, try `(root1 5 3 6)`. If you get an error, which is likely if you wrote your code the straightforward way, figure out how to change it so that your function returns a false value in those cases where there is not a valid solution.

Problem 3: Flight Time

Given an initial upward velocity (in meters per second, or m/s) and initial elevation or height (in meters, or m), write a procedure that computes how long the baseball will be in flight. Remember that gravity is a downward acceleration of $9.8 m/s^2$ To solve this you'll need the root of a quadratic equation. Try using root1 first, then root2. Only one of these solutions makes sense. Which one? And why? Use this to create a correct version of the procedure below.

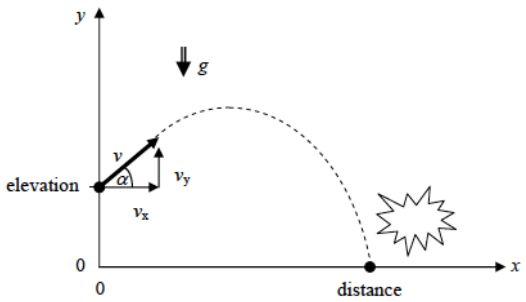
```
(defun time-to-impact (vertical-velocity elevation)
      YOUR-CODE-HERE))
```

In some cases, we may want to know how long it takes for the ball to drop to a particular height, other than 0. Using your previous procedure as a template, write a procedure that computes the time for the ball to reach a given target elevation.

```
(defun time-to-height  (vertical-velocity elevation target-elevation)
      YOUR-CODE-HERE))
```

Problem 4: Flight Distance

Suppose the baseball is hit with some velocity *v*, at a starting angle *alpha* relative to the horizontal (in degrees), and from an initial elevation (in meters). Compute the distance in the horizontal direction the baseball will travel by the time it lands. Remember that some of the velocity vector goes into the *x* direction, and some into the *y*, as shown below:



Motion of a baseball in two dimensions, acting under gravitational acceleration *g*.

Checking the Common Lisp documentation, you will find procedures `sin` and `cos`. To use these (which require angles in radians rather than degrees), you may also find the procedure `degree2radian` useful. It is given below:

```
(defun degree2radian (deg)
```

```
    (/ (* deg pi) 180)))
```

Write a procedure `travel-distance-simple` that returns the lateral distance the baseball thrown with given velocity, angle, and initial elevation will travel before hitting the ground.

```
(defun travel-distance-simple (elevation velocity angle)

   YOUR-CODE-HERE))
```

Note that everything in metric units (distances in meters, weight in kilograms).

There are conversion routines in baseball.lisp for metric/english.

Using your code, determine the time to impact of a ball hit at a height of 1 meter (right down the middle of the plate) with an initial velocity of 45 m/sec (about 100 mph – about what a really good professional player can do – without steroids), at angles of 0, 45 and 90 degrees (be sure to use radian units or degree units depending on how you provide input to *sin* and *cos*, but remember that those procedures expect arguments in units of radians).

How far does the baseball travel in each case? Provide answers both in meters and feet. Notice the distance traveled in feet for a ball hit at a 45 degree angle, with this bat speed. Seems incredible – right?

Problem 5: What's the best angle to hit?

Before we figure out why professional players don't normally hit 700 foot home runs, let's first see if we can find out the optimal angle at which to launch a baseball, in order to have it travel the furthest. Write a procedure `find-best-angle` that takes as arguments an initial elevation and an initial velocity, and which finds the best angle at which to hit the baseball to optimize distance traveled. You will probably want to write a recursive procedure that tries different angles between 0 and pi/2 radians, sampled every 0.01 radians or between 0 and 90 degrees, sampled every 1 degree (depending on whether your code works in radians or degrees – either way be sure that you provide the right kind of unit to your trigonometric functions).

```
(defun find-best-angle (velocity elevation)

  YOUR-CODE-HERE)
```

Use this for same sample values of elevation and velocity. What conclusion can you reach about the optimal angle of hitting?

To turn in:

a) Lisp source code in plain text format. If more convenient, create a separate file for each of the problems above.
b) Your input test file (s)
c) Your output file (s)
d) Short discussion including:
    a. how much time it took,
    b. the 2 or 3 most difficult aspects of doing this project
    c. Any additional comments