EEN118 LAB THREE

In this lab, you will be performing a simulation of a physical system, shooting a projectile from a cannon and working out where it will land. Although this is not a very complicated physical system, it does introduce the fundamental principles of computer simulation, which is a very important field. As the last step, you will turn the simulation into a little interactive game about blowing things up.

1. *First Flight.*

If a projectile is fired or thrown upwards at a known velocity, its entire trajectory can be accurately computed. You probably remember the physics formula

$$h = v t - ½ g t^2$$

(h is the projectile's height at any given time, v is the initial upwards velocity, t is the time elapsed since it was fired or thrown, and g is the "acceleration due to gravity", an approximate constant anywhere on the surface of the earth). If you measure h in feet, t in seconds, and v in feet per second, g is about 32.174.

Write a program that allows the user to enter the initial velocity for a cannonball, then computes and prints the height above ground level of the ball for the first 20 seconds. Experimentation is easy, but velocities of between 100 and 200 feet per second give reasonable trajectories.

There are functions in the library called `read_int`(), `read_double`(), `read_float`(), `read_string`(), etc. When called, they wait until the user has typed something of the appropriate type, then return as their result that value. A program could rudely ask someone their age like this:

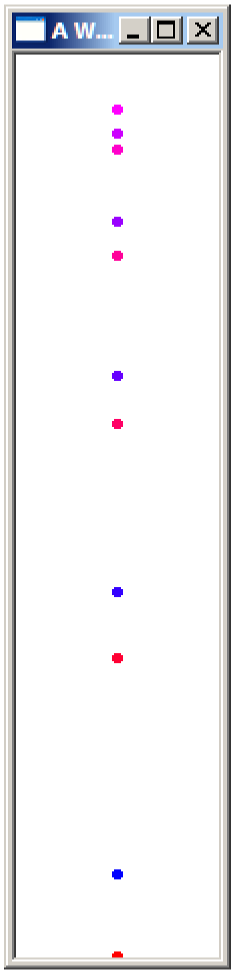
```
print("How old are you? ");
const int age = read_int();
```

The system only notices input typed when the black "dos shell" window is active.

2. *Visual Representation.*

Now make your program plot those altitudes as vertical dots in a reasonably sized window, one dot for each second of flight. *If* you make the colour of the dots gradually change as time progresses, the results are much easier to interpret.

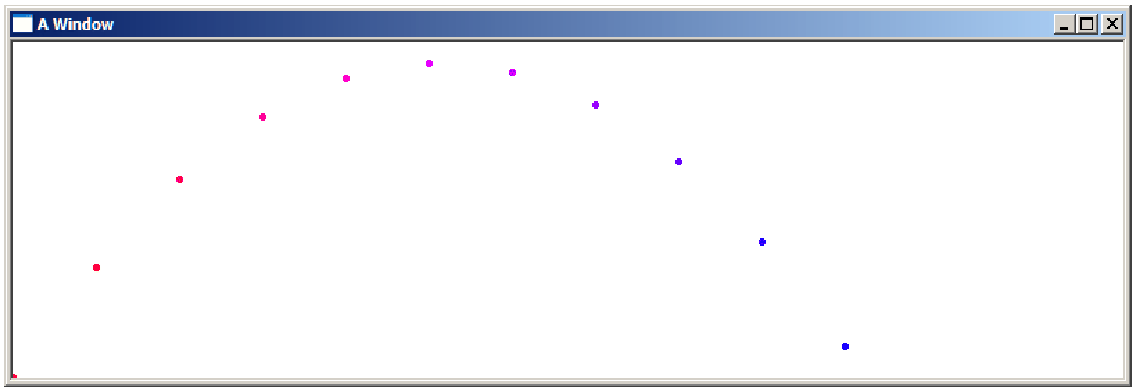
If you use reasonable velocities, you will be able to have a scale of one pixel per foot, which is quite convenient.



3. *Making the Plot More Readable.*

Even with the dots gradually changing colour as time increases, that vertical picture doesn't give a very good representation of a trajectory. Using height as the vertical axis at one pixel per foot works out quite nicely. Letting time provide the horizontal axis also makes a lot of sense. You'll need to scale the time axis somehow, as one pixel per second is useless. Even if the cannonball is fired upwards at the speed of sound, it only spends about 63 seconds in flight.

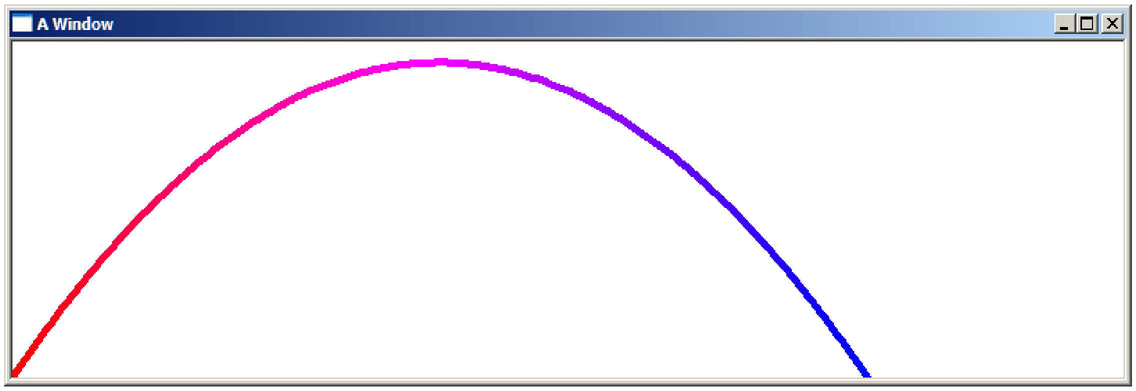
Make it happen.



4. *A Nice Arc.*

It seems absolutely trivial to convert the program to draw lines between the points. It is almost trivial, but there is one little problem that you'll have to deal with. The problem will become obvious when you try it. Try it now.

Once you have got the points joined by lines, you notice that one point per second isn't enough. See what you can do to produce a nice smooth accurate trajectory.



5. *Take the Battle to the Enemy.*

Sensible artillerymen do not fire their cannons straight up. The always set a cannon at an angle so that the cannonball will travel horizontally as well as vertically. If the cannon is set at angle α (0° = vertical, 90° = horizontal), the position equations are slightly altered:

$$h = v\,t\cos(\alpha) - \tfrac{1}{2}\,g\,t^{2}$$
$$d = v\,t\sin(\alpha)$$

Now, *d* represents the horizontal distance travelled. Remember that computers use radians instead of degrees; there are π radians to 180°.

Modify your program so that it now takes two inputs from the user before plotting the path. The first is still the initial velocity, the second should be the angle that the cannon is set at. Be kind to your user: accept the input angle in degrees, and have the program make the conversion.

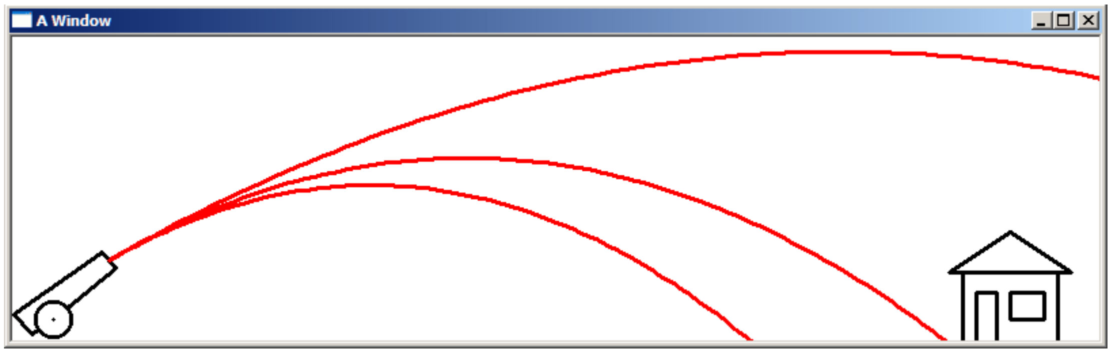
And of course, the horizontal axis should now represent the horizontal distance travelled, not just elapsed time. If the user enters an angle of zero (vertical) the trajectory should appear as a straight line up and down.

6. *The Game.*

Draw a simple representation of a cannon set at the right angle at one edge of the window, and a simple representation of the Enemy HQ at the other. If the cannonball hits the ground inside Enemy HQ, the user wins. Perhaps you could draw a little congratulatory explosion. Don't get obsessed by graphical perfection. It is only a game.

7. *Make it Interesting.*

Give the user a few chances to pick the right velocity and angle, three to five tries seems reasonable. Make the HQ appear at a random position each time a new game is started, so the user can't just learn the right numbers. Perhaps even make it work in *real time*, so that a flight of five seconds really takes five seconds.



8. *OPTIONAL, For Extra Credit.*

This was a popular game on Macintosh computers in the mid-1980s. Sell your version of it to Nintendo and become a millionaire. My cut is only 15%.