Lecture 23

Project B

More classes, constructors

Admin

- Sent an e-mail on Monday
  - Big thing was the reading is supposed to be 6.2
  - Section 6.3, 8.1 and 8.2 will be covered on Friday & Monday.
    - If this stuff is confusing you, please read ahead of time.
      - It will help, and should only take an hour to 90 minutes.

Project B

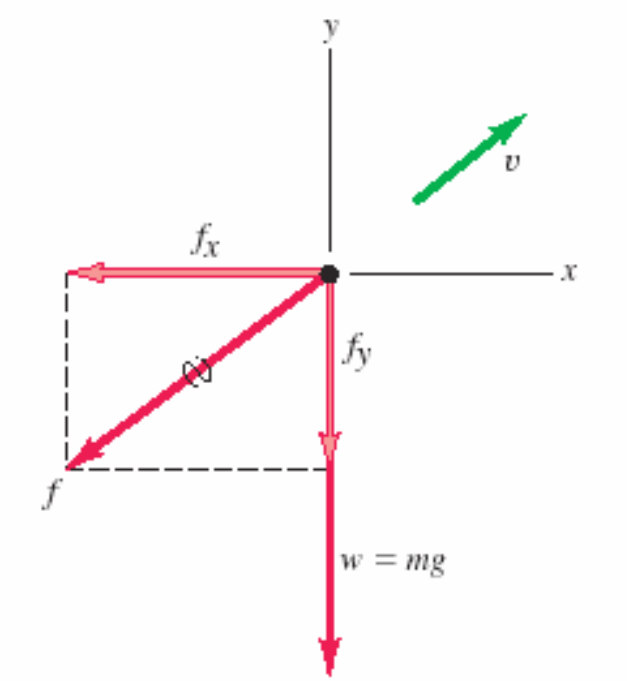
- Project B consists of two short(ish) projects you will write from scratch.
  - First one is a physics simulation
  - Second one is an encoding/decoding problem.

Cannons for Junkyard Wars

- The problem:
  - You are working on a project for Junkyard Wars and have been asked to figure how to shoot a ball as far as possible from a cannon with an initial velocity of 500 m/s (about 1.5 times the speed of sound). Further, as the initial velocity can't be changed, they'd like you to figure out how to shoot targets at different ranges by only changing the angle of fire from the cannon.
- Why a program?
  - Air resistance is dependent upon velocity.
  - Velocity is constantly changing
  - Hard to solve mathematically.

Physics

- The forces acting on the ball in the “y” direction (up and down) are:
  - 9.8kg*m/s2 in the downward direction (due to gravity)
  - D*vy2 in the opposite direction of travel air resistance.
- In the “x” direction the only force acting on the ball is air resistance
  - (so D*vx2) in the opposite direction of travel.
- Thus:
  - F_y=-9.8kg*m/s^2 ± D*v_y^2 and
  - F_x= ± D*v_x^2
- Where the ± is determined by the direction of the flight and D is a constant.



Modeling

- The way we model this is actually quite simple
  - We keep track of the x and y positions and velocities
  - We chose a small timestep, say 0.01 seconds.
    - We update the positions and velocities after one timestep.
      - New position is equal to old position + old velocity * timestep.
      - New velocity is equal to old velocity + acceleration * timestep.

Forces

- The ball we are modeling is 1kg, so the *value* of the forces and the acceleration caused are the same…
  - The force from drag is D*v^2, where D is a constant with units of kg/m
    - Force is in the opposite direction of velocity
  - The force from gravity is 9.8 m/s^2 (downward).

Time 0

- Initially, the ball is traveling at 500m/s
  - Say we fire it at a 45 degree angle and start (arbitrarily) at location (0,0).
  - This provides us with an initial x velocity of 500m/s*cos(45 degrees) and an initial y velocity of 500m/s * sin(45 degrees).
    - **v_x=353.55 m/s**
    - **v_y=353.55 m/s**
    - **p_x=0**
    - **p_y=0**

Time 0.01

- Say our time step is 0.01 seconds.
  - p_x will change by 0.01s*v_x or 3.53 meters.
  - The value of p_y will change by the same amount.
- The velocity in the x direction will drop by D*v^2 times the timestep
  - D for this problem is 0.00126 kg/m
    - see handout for where that number came from, but for the entire problem you can just use that value of D.
  - So the change in v_x is
    - (0.00126 kg/m*(353.5 m/s)^2 * 0.01s)/1kg
    - Which is 1.575m/s.

Time 0.01 (continued)

- For the y velocity there is also a gravitational component that produces an acceleration of 9.8m/s^2.
  - Recall that v=a*t+v_0.
  - In our case that means the y velocity will be further reduced by 9.8m/s^2*0.01s or .098 m/s.
    - v_x=351.98 m/s
    - v_y=351.88 m/s
    - p_x=3.53
    - p_y=3.53

Now

- You just keep doing the same thing until the ball comes back to the ground
  - P_y<= 0

Part II

- Caesar cipher
  - Pretty simple notion.
  - Encode letters my moving them over a constant amount.
  - The traditional cipher was
    - A=D, B=E, …X=A, Y=B,Z=C
    - So “BOB” would be “ERE”

Classes

```
struct Cmpx
{
        double real;
        double img;
        Cmpx add(Cmpx x);
};
```

```
Cmpx Cmpx::add(Cmpx x)
{
        Cmpx tmp;
        tmp.real=x.real+real;
        tmp.img=x.img+img;
        return(tmp);
};
```

```
main()
{
        Cmpx a,b,c;
        a.real=1;
        b.real=3;
        a.img=5;
        a.img=7;

        c=b.add(a);

        cout << "("<< c.real << " + "<< c.img <<"i)\n";
}
```

```
struct Cmpx
{
        double real;
        double img;
        Cmpx();
        Cmpx(double r, double i);
        Cmpx add(Cmpx x);
};

Cmpx::Cmpx()
{
        real=0;
        img=0;
        cout << "Here I am!\n";
}
```

```
main()
{
        Cmpx a, b;
        cout << "X\n";
        Cmpx c;
        cout << "Y\n";
        a.real=1;
        b.real=3;
        a.img=5;
        b.img=7;

        c=b.add(a);

        cout << "("<< c.real << " + "<< c.img <<"i)\n";
}
```

```
struct Cmpx
{
        double real;
        double img;
        Cmpx();
        Cmpx(double r, double i);
        Cmpx add(Cmpx x);
};

Cmpx::Cmpx()
{
        real=0;
        img=0;
        cout << "Here I am!\n";
}

Cmpx::Cmpx(double r, double i)
{
        real=r;
        img=i;
        cout << "Here now!\n";
}
```

```
main()
{
        Cmpx a(1,5);
        Cmpx b(1,5);
        cout << "X\n";
        Cmpx c;
        cout << "Y\n";

        c=b.add(a);

        cout << "("<< c.real << " + "<< c.img <<"i)\n";
}
```

```
x
```