• Only 4 weeks left in the semester, 4 lectures left including this one

• Format of final – discussion next week
Reminder

• HW08 due Thursday at noon
• HW09 will be assigned today/tomorrow
• Grading of HWs almost caught up!
• Solutions all posted
• Register for piazza!
  – Only 60 out of 70 so far
  – All questions about remaining homeworks and final exam will only be answered via piazza

• My office hours:
  – 3:30-4:30pm Tuesdays in Room 022-C (our computer lab)

• TA office hours
  – In Room 022-C
    • Mondays 7-9:30pm
    • Tuesdays 4:30-6:30pm
    • Wednesdays 7-9:30pm
Review and Outline

• Last time:
  – Gaussian statistics
  – Uncertainties: statistical and systematic

• Today:
  – A note on the gravity problem
  – Other probability distributions
  – Interpreting uncertainties
Example: Structs and Functions
Example: The Gravity Problem

- **HW07, part 3:**
  - Make a function that reads in a data file containing info on some number of masses and stores information in an array of type ‘body’
  - Make a second function that calculates the center of mass of this collection of masses
  - Make a third function that calculates the three-vector of the gravitational force acting each of the masses

- **HW08, part 2:**
  - Complete the simulation for the motion of the masses
  - Calculate the new positions, velocities, and forces for each body for all Delta_t time steps
Example: The Gravity Problem

The Problem:
We want to read in the position vectors, initial velocities, and masses of a bunch of objects. Then, using this data, we want to calculate the gravitational force on each object, due to the others. (For the first part, we'll ignore the initial velocities.)

Here's one of our objects. It has mass “m”, and it's located at position X. The calculated force on it is F.
Example: The Gravity Problem

Adding the Forces:

To find the total force on one mass, we just add the force vectors due to each of the other forces.

\[ \vec{F} = \vec{F}_b + \vec{F}_c + \vec{F}_d \]
Example: The Gravity Problem

Finding Distance and Direction:

We'll need to know the distance and direction to each other object.

This is the vector from a to b:

\[ \vec{r} = \vec{X_b} - \vec{X_a} \]

The magnitude of this vector gives us the distance:

\[ r = |\vec{r}| \]

Once we know these, we can make a unit vector pointing from a to b:

\[ \vec{u} = \frac{\vec{r}}{r} \]
Example: The Gravity Problem

Calculating a Single Force:

Newton tells us that the magnitude of the gravitational force between two objects is:

\[ F = G \frac{m_a m_b}{r^2} \]

The force will point toward the other object, so the force vector will just be:

\[ \vec{F} = F \hat{u} \]
Example: The Gravity Problem

**Data Structure:**

To solve this problem programmatically, we'll first need a data structure to store information about each body:

```c
typedef struct{
    double s_vec[3];  // space(position) vector
    double v_vec[3];  // velocity vector
    double f_vec[3];  // force vector
    double mass;
} body;

const int MAX_BODIES = 100;
body bodies[MAX_BODIES];  // array of bodies
```
Example: The Gravity Problem

Reading Data from a File:

```c
int read_data(char* file, body *bodies){
    int num=0; // number of entries read from file
    int status;
    FILE *file_p = fopen(file,"r");

    while(num<MAX_BODIES) {
        status=fscanf(file_p,"%lf %lf %lf %lf %lf %lf %lf",,
            &bodies[num].s_vec[0],
            &bodies[num].s_vec[1],
            &bodies[num].s_vec[2],
            &bodies[num].v_vec[0],
            &bodies[num].v_vec[1],
            &bodies[num].v_vec[2],
            &bodies[num].mass);
        if (status==EOF) break;
        num++;
    }
    return num;
}
```
Example: The Gravity Problem

Some Useful Functions:

```c
// Find distance between two points:
double distance(double *svec1, double *svec2){
    double dist2=0;
    int i;
    for (i=0; i<3; i++)
        dist2 += (svec1[i]-svec2[i]) * (svec1[i]-svec2[i]);
    return sqrt(dist2);
}

// Find difference of two vectors:
void vsub(double *v1, double *v2, double *v1m2){
    int i;
    for (i=0; i<3; i++)
        v1m2[i] = v1[i]-v2[i];
}
```
Example: The Gravity Problem

Calculating the Forces:

```c
void forces(body *bodies, int nbodies){
    double dist, force;
    double dirvec[3];
    const double G = 6.67e-11;

    for(int i=0; i<nbodies; i++){
        bodies[i].f_vec[0]=0;
        bodies[i].f_vec[1]=0;
        bodies[i].f_vec[2]=0;
        for(int j=0; j<nbodies; j++){
            if ( i!=j ) {
                dist = distance(bodies[i].s_vec,bodies[j].s_vec);
                vsub(bodies[j].s_vec,bodies[i].s_vec,dirvec);
                dirvec[0] /= dist;
                dirvec[1] /= dist;
                dirvec[2] /= dist;
                force = G*bodies[i].mass*bodies[j].mass/(dist*dist);
                for(int k=0; k<3; k++) {
                    bodies[i].f_vec[k] += force*dirvec[k];
                }
            }
        }
    }
}
```
Example: The Gravity Problem

Trajectories:

\[ \vec{F} = m \vec{a} \]
\[ \vec{a} = \frac{\vec{F}}{m} \]

\[ \Delta \vec{V} = \vec{a} \Delta t \]

\[ \vec{V}_{\text{new}} = \vec{V}_{\text{old}} + \Delta \vec{V} \]
Example: The Gravity Problem

Calculating Trajectories:

Here's one simple way to approximate the motion of the objects. Here, we assume a constant velocity during each time step:

```c
void evolve(body *bodies, int nbodies, double delta_t) {
    for (int i=0; i<nbodies; i++) {
        for (int j=0; j<3; j++) {
            double acceleration_j = bodies[i].f_vec[j] / bodies[i].mass;
            bodies[i].s_vec[j] += bodies[i].v_vec[j]*delta_t;
            x_{j new} = x_j + v_j \Delta t
            v_{j new} = v_j + a_j \Delta t
            bodies[i].v_vec[j] += acceleration_j * delta_t;
        }
    }
}
```
A Better Approximation:

Here's a better approximation. In this version, we only assume a constant acceleration during each time step:

```c
void evolve(body *bodies, int nbodies, double delta_t){
    for (int i=0; i<nbodies; i++) {
        for (int j=0; j<3; j++) {
            double acceleration_j =
                bodies[i].f_vec[j] / bodies[i].mass;
            x_j^new = x_j + v_j \Delta t + \frac{1}{2} a_j \Delta t^2
            bodies[i].s_vec[j] +=
                (bodies[i].v_vec[j]*delta_t+0.5*acceleration_j*delta_t*delta_t);
            v_j^new = v_j + a_j \Delta t
            bodies[i].v_vec[j] += acceleration_j * delta_t;
        }
    }
}
```
Debugging
Errors: Two Types

**Compile-time versus Run-time Bugs:**

The bugs that afflict our programs can generally be divided into two categories:

- **Compile-time** bugs are caught by the compiler, which will warn us about them. This kind of bug includes all of the various syntax errors we've talked about already: variable type mismatches, missing semicolons, and any typo that isn't valid C code.

- **Run-time** bugs occur when our program is all perfectly good C code, but it doesn't do what we want it to do. It may produce strange results, or crash in some way. These bugs are due to mistakes in our program's design, not typos.
Compile-time Bugs

Here's an excerpt from the error messages observed when compiling a complicated piece of code.

This looks bad, but the first error gives us the solution:

```
src/MemoryMap.cpp:26: parse error before ...
```

Looking around line 26, the programmer found that line 25 was missing its **semicolon**. Often one simple fix will clear up many errors.

(And often that simple fix is a semicolon!)

**Rule of thumb** When you get a large number of error messages from the compiler, just look at the **first one**. Errors cascade, so one bad line will corrupt many following lines.
Top seven compiler errors in my experience:

7. Use of variable outside scope of conditional

6. Passing wrong arguments to functions

5. Mismatched () on conditionals

4. Mis-cast variables

3. Missing `include` statement for header file

2. Missing semi-colon

1. General typo
Run-time Errors

- Two types:
  - innocuous kind which just foul up your data in a subtle fashion
    - int i=25;
      printf ("%f \n",i);
    Wrong format specifier
    prints 0 instead of 25
  - catastrophic kind: cause program to cease running immediately
    - aka a “crash!”
    - output form a crash is stored typically in a core dump, a file of human-readable instructions the computer was following when the crash happened – like the black box of an airplane
Run-time Errors: Examples

**Segmentation faults:**
Your program has tried to access memory that is not allocated to it. That is, it's trying to manipulate data in memory locations it has no privilege to access. This is the OS limiting your access to resources.

Examples:
1) int i;
   scanf("%d", i); // should have used &i

2) FILE *outfile;
   // outfile = fopen("my_file.txt","w");
   fprintf(outfile, "hello\n"); // bad things happen if you try
   // to access an unopened file!

**Divide by zero:**
Example:
   int i = 1;
   float f = 3.14/(i-1);

Divide by zero generates a “floating exception” error for integer arithmetic, but with floating-point arithmetic your program will continue to run, using the special values “inf” or “NaN” (“Not a Number”) as the result of the division. This is almost certainly not what you want.
Finding Errors: Tips

• Follow good coding practices, which we are learning
• Read compiler messages – avoid warnings and address errors according to line number
• Make incremental changes to code and compile frequently to keep track of changes
• For run-time errors, insert many printf() statements around the wonky bits to see how your variables are being manipulated. Remove these later for performance sake.
• If all else fails, back out all intricate code, recompile successfully the simple remnants and add back in small piece by piece, until it breaks again
• Last advice: use google, etc., to find if others have encountered your error!
Problem Solving
Problem Solving: Using Logic

- Computer programs are a powerful tool for solving problems!
- But like any problem-solving venture, it is best to be methodical and logical when setting out to find a solution

- Simple steps:
  1. Define and understand the problem
  2. Outline a solution
  3. Design an algorithm
  4. Convert the algorithm into a program, compile and execute
  5. Check the results

(These somewhat differ from what was presented in Reddy but the elements are largely the same)

- These simple steps can be used in any setting, using any language.
- In fact they can be adjusted for use in any setting (even non-programming)
Problem Solving and Logic: Example

• Say you have some data, like these values: 0, 4, 3, 2, 5

• I want you to use a computer program to take this data and report the values in order, high to low
1. Define the problem:
   – Take this data and order it numerically, high to low
Problem Solving and Logic: Example

1. Define the problem:
   – Take this data and order it numerically, high to low

2. Outline a solution: use human language
   – I need to access this data and put it in a format that a program can understand
   – I need to report back the data in numbered order
3. Write an algorithm: use “pseudocode”
   – Input these numbers into a file

   numbers.dat

   0, 4, 3, 2, 5
3. **Write an algorithm: use “pseudocode”**
   - Input these numbers into a file
   - Read them in into a list and store them
   - an array is best here, for convenience
     • call this the “input array”
3. Write an algorithm: use “pseudocode”
   - Input these numbers into a file
   - Read them in into a list and store them
   - an array is best here, for convenience
     • call this the “input array”
   - Create another array, same size, called “output array”
3. **Write an algorithm: use “pseudocode”**

- Input these numbers into a file
- Read them in into a list and store them
- An array is best here, for convenience
  - Call this the “input array”
- Create another array, same size, called “output array”
- Read in first value from input array, put in first position of output array

```
0,4,3,2,5
```

```
[0, 4, 3, 2, 5]
```

```
[ , , , , ]
```

```
[0, , , , ]
```

```
numbers.dat
```

```
[0,4,3,2,5]
```

```
input array
```

```
output array
```
3. Write an algorithm: use “pseudocode”
   - Input these numbers into a file
   - Read them in into a list and store them
   - an array is best here, for convenience
     • call this the “input array”
   - Create another array, same size, called “output array”
   - Read in first value form input array, put in first position of output array
   - Read in next value from input array; find value in output array that is smaller than this new value, push all subsequent values down and insert this one

```
numbers.dat
0,4,3,2,5
[0,4,3,2,5]
[0, , , , ]
[4,0, , , ]
```

input array
output array
3. Write an algorithm: use “pseudocode”
   – Input these numbers into a file
   – Read them in into a list and store them
   – An array is best here, for convenience
     • Call this the “input array”
   – Create another array, same size, called “output array”
   – Read in first value from input array, put in first position of output array
   – Read in next value from input array; find value in output array that is smaller than this new value, push all subsequent values down and insert this one
   – Repeat until all values exhausted

```
0, 4, 3, 2, 5
```

```
[0, 4, 3, 2, 5]
```

```
[0, , , , ]
```

```
[4,0, , , ]
```

```
[4,3,0, , ]
```

```
[5,4,3,2,0]
```
3. Write an algorithm: use “pseudocode”
   - Input these numbers into a file
   - Read them into a list and store them
   - An array is best here, for convenience
     • Call this the “input array”
   - Create another array, same size, called “output array”
   - Read in first value from input array, put in first position of output array
   - Read in next value from input array; find value in output array that is smaller than this new value, push all subsequent values down and insert this one
   - Repeat until all values exhausted
   - Print to screen: ordered: 5, 4, 3, 2, 0

```
numbers.dat

0, 4, 3, 2, 5

input array
[0, 4, 3, 2, 5]

output array
[ , , , , ]

[0, , , , ]

[4,0, , , ]

[4,3,0, , ]

[5,4,3,2,0]
```

```
4. Write the code...omitted for now.

5. Check results: do they look sensible?
   - this is the most important – and most often overlooked – part of the process
   - “sanity check” – use a simple case in complicated settings

ordered: 5,4,3,2,0
Some Notes on Style:
Good Coding Style
Some Guidelines for Good Coding Style

• There are many opinions on this issue, many choices of convention one can adopt

• Ideas presented here are widely accepted as “best practices”

• These are summarized on the class wiki at the style guide link

• You should start using these conventions

• Why?
  – make your code look more professional
  – easier for me, TA and grader to assess
  – easier for others to examine if it is ever distributed
  – easier for you to debug
Some Guidelines for Good Coding Style

• **Variable names:**
  – start variable names with *a lower case letter* and initialize properly:
    ```c
    int area = 0;
    ```
  – use *variable case* for multi-word names:
    ```c
    int xLength = 0;
    ```
  – make sure the name is *descriptive but short* esp. for variables of extended scope. Also do not abbreviate:
    - Ex: use `beginTime` and `endTime`
    - instead of `begT` and `endT`
    - instead of `x` and `y`
    - instead of `timeOfBeginning` and `timeOfEnding`

• **Function names:**
  – as for variables, use mixed case and avoid abbreviations:
    ```c
    void calculateDeltaT(int start, int end, *deltaT)
    ```
Some Guidelines for Good Coding Style

• Constant variable names:
  – define these with descriptive names that are all CAPS
    `const double TWO_PI = 6.283185;`

• Do not use pre-processor definitions:
  do not use `#define`
  – these are potentially dangerous if used improperly – so just avoid using them in this class and in life
Some Guidelines for Good Coding Style

• Variables of limited scope:
  – These can be short but take care:
    ```
    for (int i=0; i<100; i++) pages[i]=0;
    – i,j,k are commonly used iterators in for loops
    ```

• use prefix “n” for counters:
  ```
  int nPages = 0;
  int nAttempts = 0;
  ```

• Do not distinguish variables just through their capitalization:
  – Do not do this:
    ```
    int itemCounter = 0;
    int ItemCounter = 0;
    ```
Some Guidelines for Good Coding Style

• Indentation of long lines:
  – Avoid code that has very long individual lines. Have each line have less than 80 characters – you will get a feel for this
  – If you have an exceptionally long statement, you can take it to a new line and indent it properly for easy debugging:

```c
printf("Acceleration of %f Kg body due to force of %f Newtons is %f m/s^2:\n", mass, force, force/mass);
makeGraph(xValues, yValues, xRMS, yRMS, outputArray);
```
Some Guidelines for Good Coding Style

• Guidelines for for(...) loops and if() statements and related:
  – do not make the conditional expression unnecessarily complicated
  – indent the body of the loop/conditional/switch to easily see what is going on:

    ```
    const int NITEMS=10;        // define data size
    int items[NITEMS];          // array is NITMES in size

    for (int i=0; i<NITEMS; i++){
      // loop over NITEMS
      items[i] = myFunction(i);
      ...
    }
    ```
Some Guidelines for Good Coding Style

• Location of `#include` statements:
  – always put these at the beginning of your piece of code

• Location of `function prototypes`:
  – always put these near the beginning of your code, after the `#include` statements but before definition of `main(…)`

• Definition of functions:
  – always put these after the definition of `main(…)`
  – indent the body of functions

• The `main(…)` program:
  – indent the body of `main(…)`
More Probability Distributions
The Bernoulli Distribution:
Success or Failure?
The Bernoulli Distribution:

- Only two possible outcomes (true or false, success or failure).
- The probability, $p$, of one possible outcome is known.

$$P(\text{heads}) = p$$
$$P(\text{tails}) = (1-p)$$
The Bernoulli Distribution

- Only two possible outcomes (true or false, success or failure).
- The probability, $p$, of one possible outcome is known.

$$P(\text{heads}) = p$$
$$P(\text{tails}) = (1-p)$$

Questions:
The Bernoulli Distribution

- Only two possible outcomes (true or false, success or failure).
- The probability, \( p \), of one possible outcome is known.

\[
\begin{align*}
P(\text{heads}) &= p \\
P(\text{tails}) &= (1-p)
\end{align*}
\]

Questions:

What is the \( p \) for tossing heads in one trial?
The Bernoulli Distribution

Questions:

What is the $p$ for tossing heads in one trial?

Say I have thrown ten heads in a row – what is the probability of doing that?
The Bernoulli Distribution

**The Bernoulli Distribution:**

- Only two possible outcomes (true or false, success or failure).
- The probability, \( p \), of one possible outcome is known.

\[
P(\text{heads}) = p \\
P(\text{tails}) = (1-p)
\]

**Questions:**

What is the \( p \) for tossing heads in one trial?

Say I have thrown ten heads in a row – what is the probability of doing that?

After ten heads, what is probability that next one is heads too?
The Bernoulli Distribution

The outcomes need not have the same probability! For example:

Outcome 1: Roll a 6.

\[ P(6) = p = \frac{1}{6} \approx 0.17 \]

Outcome 2: Roll something else.

\[ P(tails) = (1-p) = \frac{5}{6} \approx 0.83 \]
The Binomial Distribution: How Many Successes in Many Attempts?
What if we flip a coin many times, or if we flip many coins? How many instances of a given outcome (say, heads) should we expect?

The **Binomial distribution** gives the probability of observing a certain number of true results, $x$, after doing $n$ tests.

$p$ is the probability of success on each test.

It's used when we are interested in a number of TRUE/FALSE experiments.

Consider $n$ coins tosses:
TRUE = HEADS,
FALSE = TAILS

What is the probability of getting $x$ HEADs-up if you flip the coin $n$ times?
Understanding the Binomial Distribution:

- $x =$ number of successes (e.g., how many heads?)
- $n =$ number of trials (e.g., how many tosses?)
- $p =$ probability of success in a single trial.

$P(x; n, p) =$ Probability of seeing $x$ successes after $n$ trials, given probability $p$ of success.

$$P(x; n, p) = \frac{n!}{x!(n - x)!} p^x (1 - p)^{n-x}$$

Number of possible ways to arrange the $x$ successes and $(n-x)$ failures.

Probability of getting $x$ successes and $(n-x)$ failures.
Consider a single fair 6-sided die
Define “success” as rolling a 5
Roll the dice 10 times – what is probability of rolling *three* 5’s?

- n=10, p=1/6, x=3
- prob(5) = 1/6 hence prob(three 5’s) = (1/6)^3
- prob(else) = 1/6 hence prob(seven non-5’s) = (5/6)^7
- But there are many ways to get three 5’s in 10 rolls:
  - 5-5-5-3-2-6-1-2-4-1
  - 5-5-5-3-3-3-2-2-4-6
  - ...
- The combination term “n choose x” calculates how many such combinations exist. Here “n choose x” = 10 choose 3 = 10! / (3! (10-3)! = 120
The Binomial Distribution

Binomial Distribution for One Fair Coin Flip:

Here's the Binomial distribution with
n = 1 (one coin flip)
p = 0.5, 50% chance of heads on a flip
1 − p = 0.5, 50% chance of tails on a flip

One toss, 50% heads/tails

\[
P(x; n, p) = \frac{n!}{x!(n - x)!} p^x (1 - p)^{n-x}
\]

x = 0: getting zero heads
x = 1: getting one heads
The Binomial Distribution

Binomial Distribution for One Biased Coin Flip:

Here's the Binomial distribution with
\( n = 1 \) (one coin flip)
\( p = 0.3 \), 30% chance of heads on a flip
\( 1 - p = 0.7 \), 70% chance of tails on a flip

One toss, 30% heads/70% tails

\[ P(x; n, p) = \frac{n!}{x!(n-x)!} p^x (1 - p)^{n-x} \]

\( x = \text{Number of heads} \)
The Binomial Distribution

Binomial Distribution for 20 Fair Coin Flips:

20 tosses of a fair coin

\[ P(x; n, p) = \frac{n!}{x!(n-x)!} p^x (1 - p)^{n-x} \]

\( x = \text{Number of heads} \)

Notice that the mean value for \( x \) is 10. Could we have predicted this?
The Binomial Distribution

Mean and Variance for the Binomial Distribution:

Given the binomial distribution:

\[ P(x; n, p) = \frac{n!}{x!(n-x)!} p^x (1 - p)^{n-x} \]

We can calculate the mean value (\( \mu \)) of \( x \):

\[ \mu = np \]

and the variance (\( \sigma^2 \)) of \( x \):

\[ \sigma^2 = np(1 - p) \]
The Binomial Distribution

The Effect of Varying $p$:

20 tosses, vary $p$ from 5% to 50%

$p = 5\%$

Very lopsided coin.

$p = 30\%$

$p = 50\%$

Fair coin.

$x = \text{Number of heads}$
A Special Case of the Binomial Distribution

The Poisson Distribution:
How Many Successes in MANY MANY MANY Attempts if $p$ is REALLY SMALL?
The Poisson Distribution

An interesting special case of the binomial distribution is the one in which:

- The number of trials, $n$, approaches infinity,
- The probability of success, $p$, approaches zero,
- The mean number of successes, $\mu = np$, remains fixed.

As these limits are approached, the binomial distribution can be approximated by the following (much simpler) expression:

$$P(x; \mu) = \frac{\mu^x}{x!} e^{-\mu}$$

This is called the Poisson Distribution, and it is valid when $p$ is small, $n$ is large and $\mu$ is some intermediate value.
The Poisson Distribution

How Good is the Poisson Approximation?

Binomial distribution, $n=20$, $p=0.05$

Poisson distribution, $\mu=1$

$P(x; 20, p)$

$n = 20$
$p = 0.05$
$\mu = np = 1$

$x = \text{Number of heads}$
The Poisson Distribution

• Rule of thumb:
  – The Poisson distribution is **fairly good** approximation of the Binomial distribution if
    • \( n > 20 \)
    • \( p < 0.05 \)
  – The Poisson distribution is an **excellent** approximation of the Binomial distribution if
    • \( n \geq 100 \)
    • \( np \leq 10 \)

  – So with just a moderate number of trials and reasonably small probability of success, the Poisson is good.
Example: The Poisson Distribution

Radioactive Decay:

Consider a sample of a radioactive material like Uranium.

We point a detector at the sample and count the number of radioactive decays that happen during a five-minute period.

- \( n \) is large, on the order of Avogadro's Number.
- \( p \) is small (it's unlikely that a particular nucleus will decay while we're looking at it).
- \( \mu = np \), the average number of decays in five minutes, is still an appreciable number, since \( n \) is so large.

The results of several 5-minute observations would be Poisson distributed.
Example: The Poisson Distribution

http://www.engineerguy.com/videos/video-lines.htm

https://www.youtube.com/watch?v=F5Ri_HhziI0
The Poisson Distribution

Mean and Variance of the Poisson Distribution:

Given the Poisson distribution:

\[ P(x; \mu) = \frac{\mu^x}{x!} e^{-\mu} \]

We can derive the very simple expression for the variance of \( x \):

\[ \sigma^2 = \mu \]

So the standard deviation is:

\[ \sigma = \sqrt{\mu} \]
The Poisson Distribution and Histograms

Consider the following:

We fill a histogram with a large number of entries, \( n \).

The probability, \( p \), that any given entry will land in a particular bin is small.

This implies that we can use Poisson statistics to describe the variations in the number of counts in a given histogram bin.

If the count in a given bin is \( m \), then the best estimate of the uncertainty in the bin count is \( \sigma = \sqrt{m} \).
The Poisson Distribution and Histograms

Histogram with Error Bars: