Introduction to Computational Physics

Meeting 7: Numerical Ingegration

Today:
- Computer Calculus.
- Test your knowledge!

Welcome back!

Remember that you can try out these programs using your account on Galileo. For instructions on how to use Galileo, see:

http://galileo.phys.virginia.edu/compfac/courses/comp_intro/connecting.html

Today we'll see one way to write a program that computes the values of definite integrals, and we'll pause for a while to look at what we've learned so far.

Let's get started!

Last Week's Practice Problem:

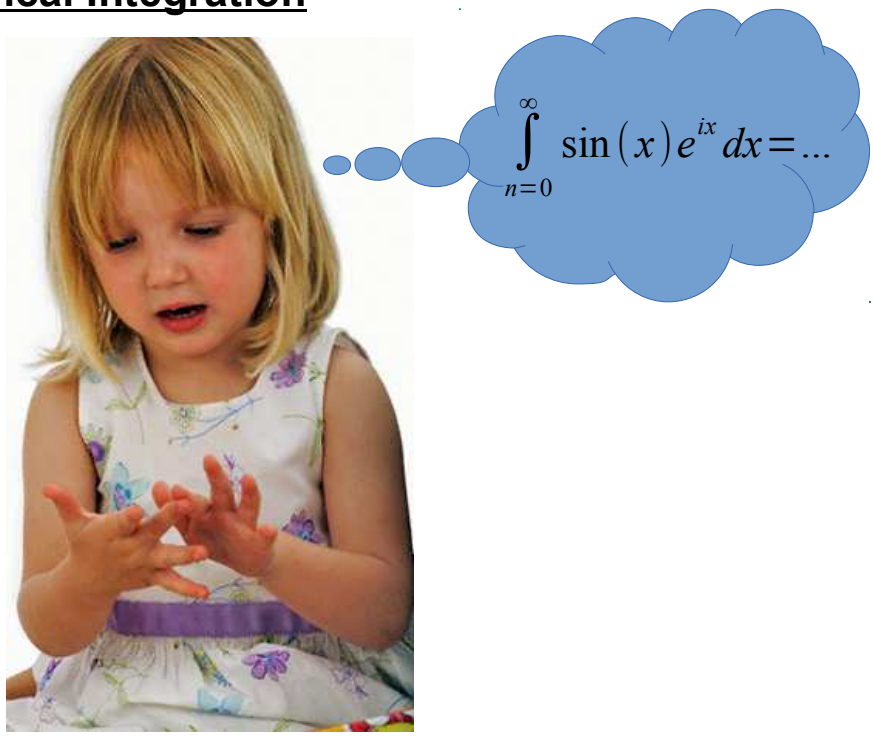
"Can you modify the 'distance' function so that it can calculate distances in higher dimensions? Hint: add another argument that specifies the number of dimensions."

```
double distance ( int n, double a[], double b[] ) {
  double d=0;
  int i;
  for ( i=0; i<n; i++ ) {
    d += pow( b[i]-a[i], 2 );
  }
  d = sqrt(d);

  return(d);
}
```

So, we just need to make two minor changes to our function. With these changes, we can now use this function to comput distances in Euclidean space for any number of dimensions!

Numerical Integration

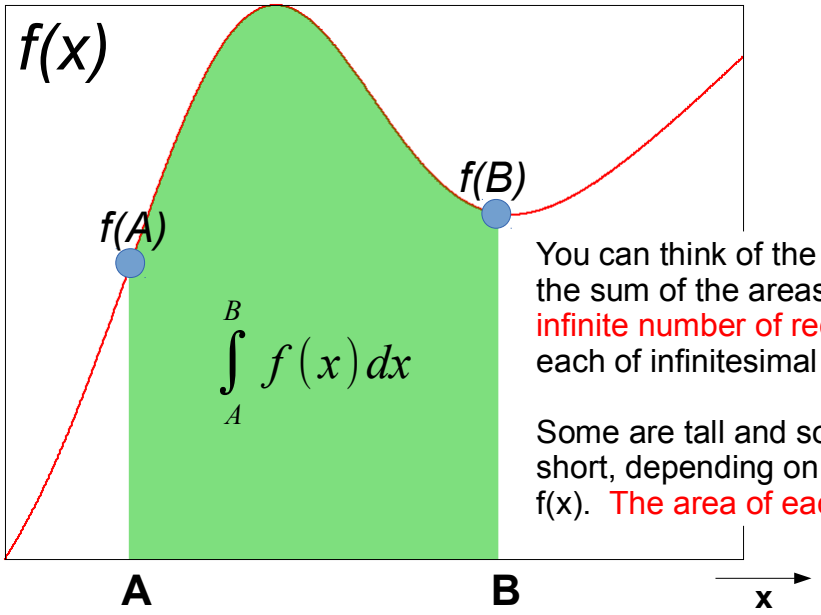


An integral is conceptually simple: it's just adding things up. (The integral sign is a stretched-out “S”, for “Sum”.) Since computers can add things very quickly and accurately, you'd think they'd be good at integration.

There are several ways that a computer program can compute an approximate value for a definite integral. Today we're going to look at one of them, called “The Trapezoid Rule”.

What's an Integral?

The integral of a f(x) below is just the area under the curve.



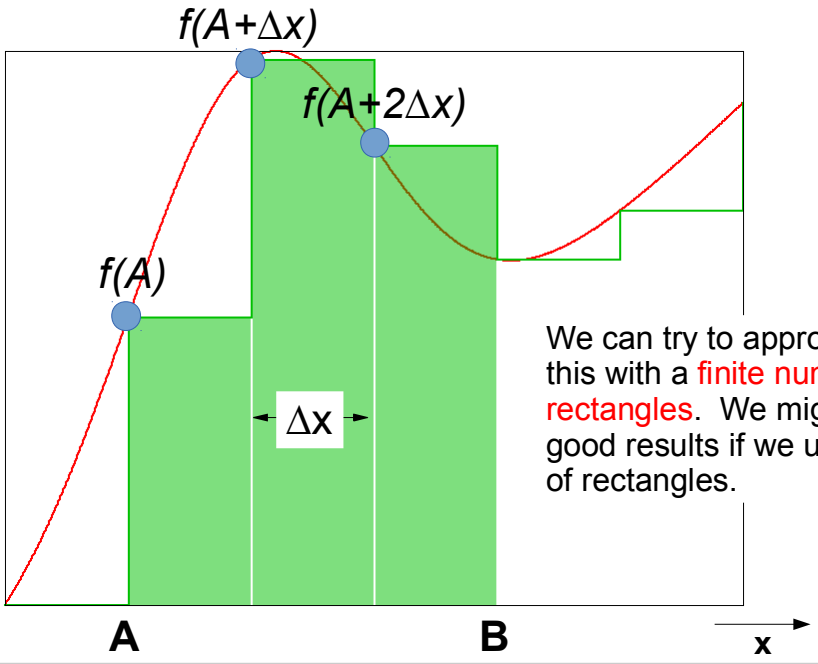
You can think of the integral as the sum of the areas of an infinite number of rectangles, each of infinitesimal width *dx*.

Some are tall and some are short, depending on the value of f(x). The area of each is f(x)*dx.

For many functions, we can calculate these integrals mathematically. Sometimes though, the integral is difficult or impossible to solve. Sometimes we don't even know what the function is: we just have some data points that define it.

In such cases, we might use a computer program to estimate the value of the integral by adding up a *finite* number of rectangles (or other shapes). This often gives us a good-enough approximation to the integral's true value.

First Approximation: Rectangles

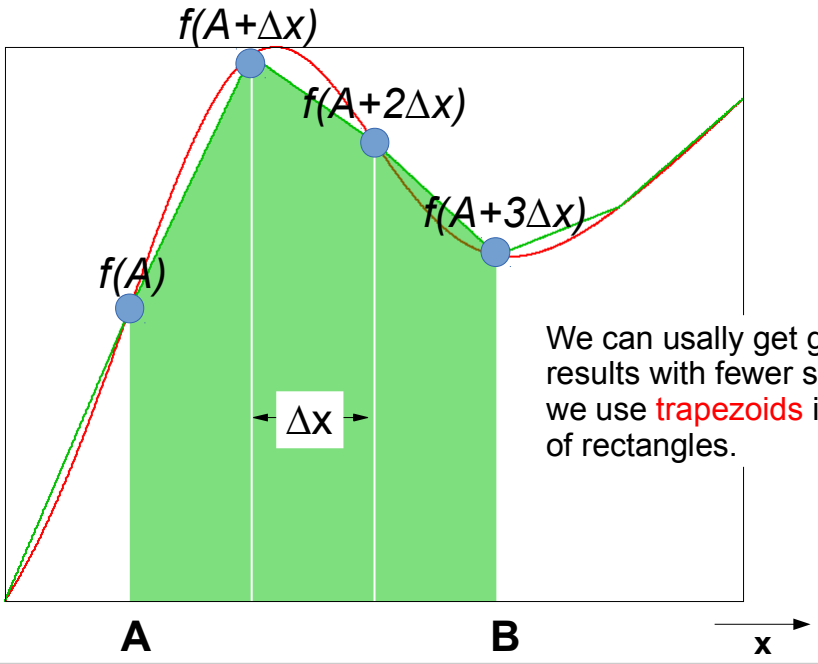


We could try estimating the area by just replacing infinitesimal “dx” with a finite-sized “Δx”. Then we could just split the range from A to B into some number of slices and compute the area Δx * f(x) for each of these.

This would probably work if we had enough slices, but (as you can see above) we might not get a very accurate result if we don't have enough.

Maybe we can find another shape that fits better....

Second Approximation: Trapezoids



A trapezoid is a four-sided shape with two parallel sides (the other sides may or may not be parallel.)

With trapezoids instead of rectangles, we can often get a more accurate estimate of the area with fewer slices. If you flip back and forth between this page and the last, you can see that, even with only three slices, the trapezoids already do a noticeably better job of approximating the area under the curve.

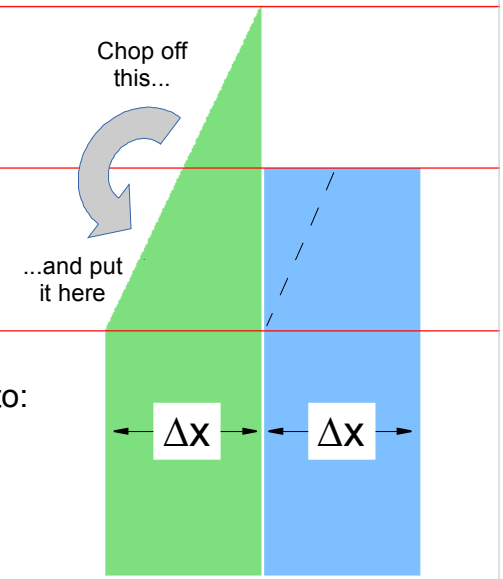
Finding the Area of a Trapezoid:

The area of one of these trapezoids is the same as the area of a rectangle with height equal to the average height of the trapezoid:

$$f(x_0+\Delta x)$$

$$\frac{f(x_0)+f(x_0+\Delta x)}{2}$$

$$f(x_0)$$



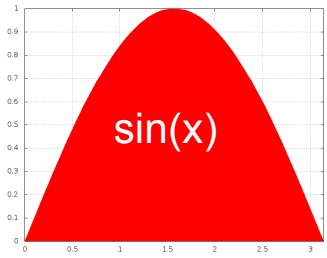
Green and blue areas are both equal to:

$$\Delta x \frac{f(x_0)+f(x_0+\Delta x)}{2}$$

You can see how this makes sense. Look at the green trapezoid. Imagine chopping off the top of it, and rotating it down to fill the area below. Voila! We've made a rectangle.

Now that we know how to calculate the area of each trapezoid, we just need to loop through all of the slices and add up the areas.

integrate.cpp:




```
#include <stdio.h>
#include <math.h>
double func( double x ) {
  double value;
  value = sin(x);
  return(value);
}
int main () {
  double x, delta, area=0;
  double height;
  double xmin=0.0, xmax=M_PI;
  int i, nsteps=100;
  delta = (xmax-xmin)/nsteps;
  x = xmin;
  for ( i=0; i<nsteps; i++ ) {
    height = ( func(x) + func(x+delta) ) / 2.0;
    area += delta * height;
    x += delta;
  }
  printf ( "Integral from %lf to %lf is %lf\n",
           xmin, xmax, area );
}
```

Function to integrate

main

Find Δx.

Start at xmin.

Find area of slice and add to total.

x=x+Δx.

Try it!

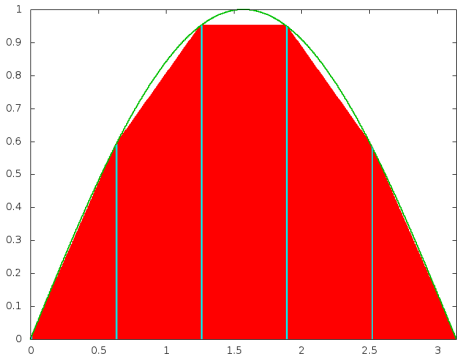
We could change our definition of “func” to make it any function we want. In this case, I've just picked sin(x) because we know how to integrate that mathematically, so we can check our program's estimate.

This technique won't work well for all functions. Imagine a function with a tall, narrow spike, for example. The trapezoids might end up just straddling the spike, and never “seeing” it.

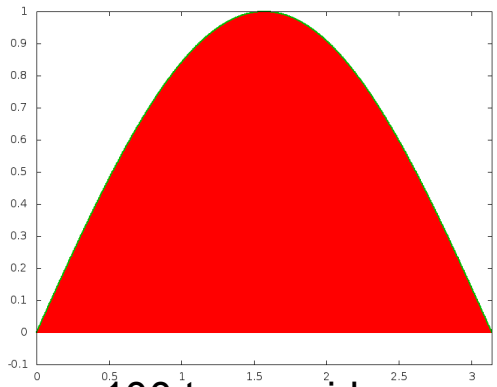
It would also have trouble with anything that goes to infinity. Think about trying to integrate 1/x from zero to one.

Our program will work for a wide range of well-behaved functions, though.

The Output:



5 trapezoids



100 trapezoids

```
"Integral from 0.000000 to 3.141590 is 1.999836"
```

From calculus, we know that the answer should really be 2.

As you can see, it's hard to distinguish between the shape made by 100 trapezoids and the actual sine function.

If you try this program with different numbers of slices, you'll find that the result is always somewhat smaller than the actual value (2). This is because the function we've chosen (sin(x)) is concave-downward in the range over which we're integrating. As you can see from the figures above, there will always be a litle space between the top of the trapezoids and the curve of the actual function. As we add more slices, the gap will get smaller and smaller and our result will asymptotically approach the true value.

Wouldn't it be Nice...?

Wouldn't it be nice if we could change tne number of boxes or xmin and xmax without having to recompile our program? We've already seen how we could do that by asking the user to type in numbers. But there's another way.

What if we could just give our program some parameters on the command line, like this:

nboxes xmin xmax

```
./integrate 100 0.0 3.14159
```

We can really do this, because C lets you give arguments to the "main" function.

Whenever you run a program, the operating system automatically gives the program any arguments you type on the command line. To use these arguments, we need to make a few changes to our program.

Giving Arguments to Your Program:

Here's how you could modify your previous program to make it accept command-line arguments:

integrate2.cpp

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
...
int main ( int argc, char **argv ) {
  double x, delta, area=0;
  double height;
  double xmin, xmax;
  int i, nsteps;

  nsteps = atoi( argv[1] );
  xmin = atof( argv[2] );
  xmax = atof( argv[3] );

....
```

Need stdlib for atoi and atof

integer, plus zero or more strings.

This list will always be the same, for any program that uses command-line arguments.

atoi and atof are “ASCII to integer” and “ASCII to float”.

Try it!

Compile it, then run it like this:

```
./integrate2 100 0.0 3.14159
```

argv[1] argv[2] argv[3]

As we noted last time, “main” is just another C function, and like other functions it can have arguments. The arguments of “main” will always be the same: first, and integer (“argc”) which tells you the number of command-line arguments, then a list of strings (represented by char **argv) which contains any parameters you give when you run the program.

The parameters are given to the program as character strings. If we want to make numbers out of them, we need to use the function atoi (for integers) or atof (for floating-point numbers).

stdlib.h is needed because that's where the functions atoi and atof are defined.

Checking Syntax:

We can further modify the program so that it checks to make sure we've supplied all of the necessary arguments:

integrate3.cpp

```
int main ( int argc, char **argv ) {
  double x, delta, area=0;
  double height;
  double xmin, xmax;
  int i, nsteps;

  if ( argc != 4 ) {
    printf ("Syntax: %s nsteps xmin xmax\n",
            argv[0]);
    exit(1);
  }

  nsteps = atoi( argv[1] );
  xmin = atof( argv[2] );
  xmax = atof( argv[3] );

....
```

Have we supplied enough arguments?

argv[0] is just the name of the program

Exit, with an error status

Try it!

If you try to run the program on the previous page without giving it any command-line parameters, it will crash with an error message. ("Segmentation fault").

Above, we show how to handle this more gracefully. Instead of allowing the program to crash, we check to see if the user gave us the required parameters. If not, we write out a polite error message and stop the program.

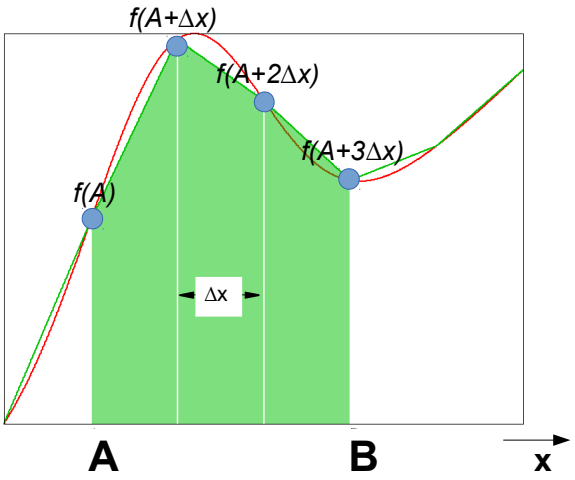
Note that argv[0] actually contains the name of the program, so we can use it as part of our message.

The "exit" function just stops the program. The argument we give it ("1" in this case) is a status inticator that's passed back to the operating system. Zero means that the program completed successfully. Anything else means that an error occurred.

Making it Faster:

Finally, note that we can make our integration program faster by realizing that we've been evaluating each f(x) *twice*.

We can do a little algebra and eliminate the duplication:



$$\int_A^B f(x) \simeq$$

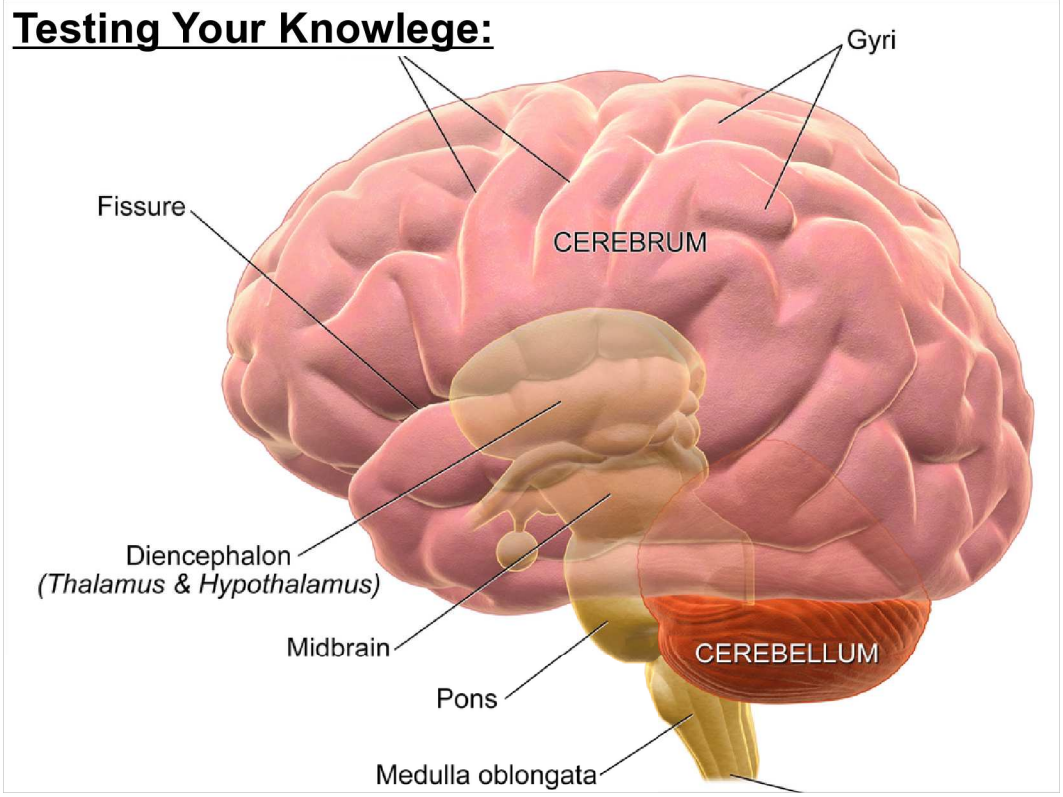
$$\Delta x \frac{f(A)+f(A+\Delta x)}{2} + \Delta x \frac{f(A+\Delta x)+f(A+2\Delta x)}{2} + \ldots + \Delta x \frac{f(A+(n-1))+f(B)}{2}$$

$$\int_A^B f(x) \simeq \Delta x \left[\frac{f(A)+f(B)}{2} + \sum_{i=1}^{n-1} f(A+i\Delta x) \right]$$

As you can see from the first line of math above, we calculate f(A+Δx) in the first term, and then again in the second term. We similarly calculate all of the f(x) values twice, except for f(A) and f(B). The bottom line of math shows how we can avoid this.

For many things, the duplication of effort won't matter. But imagine a case where we have many slices, and where f(x) is a complicated function that takes a long time to calculate. Then we might benefit from the streamlined version of the calculation.

We won't go through this here, but today's practice problem is to modify our program so that it uses the streamlined technique above.



Let's take a look at what we've learned so far. Here are a few quiz questions for you.

The next five pages will have the questions without answers. At the end of this document I've put the answers.

1. How would you write a program that prints out "Hello World!" ?

2. What format specifier would you use to print an integer variable?

3. What's wrong with the following?:

```
double x;
scanf("%lf",x);
```

4. What's the index of the last element in an array defined like this?:

```
int values[10];
```

1. How would you write a loop that prints out the numbers 1 through 10?

2. What's wrong with the following?:

```
if ( x=1 ){
  printf( “X equals one\n” );
}
```

3. What's wrong with this?:

```
double x[3];
x[1] = 1.0;
x[2] = 2.0;
x[3] = 3.0;
```

1. Is it safe to assume that variables have a value of zero before they're used? Why or why not?

2. What C function would you use to open a file?

3. What C function would you use to write into a file?

4. Why won't the following program compile?:

```
#include <stdio.h>
int main () {
  int i=0
  printf (“The value of i is %d\n”, i);
}
```

1. What Linux command would you use to compile a C or C++ program?

2. What Linux command would you use to edit or create the file “program.cpp”?

3. What Linux command would you use to copy file.cpp to file2.cpp?

4. What Linux command would you use to list your files?

1. Why won't this work?:

```
char numbers[5] = “12345”;
```

2. What's the right way to do the following?:

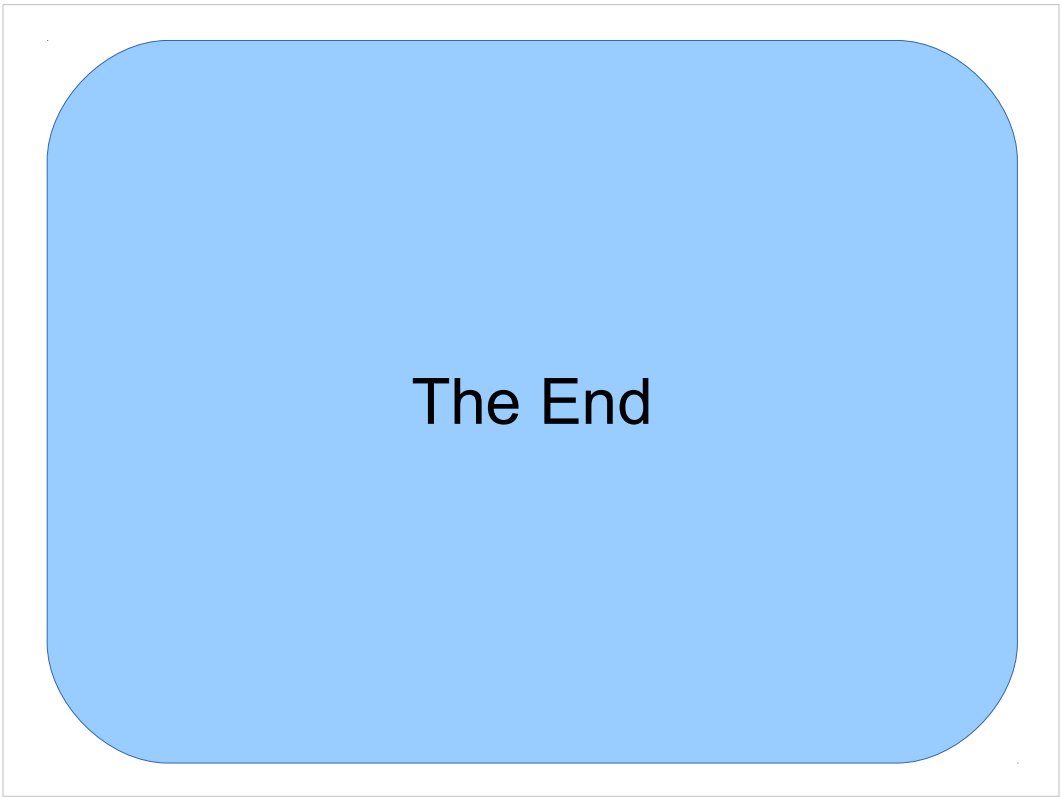
```
char a[] = “stuff”;
char b[] = “stuff”;
if ( a == b ) {
  printf(“They're equal\n”);
}
```

3. What format specifier would you use to write a character string?

Practice Problem:

Take a look at the “Making it Faster” slide from earlier today. Can you modify your integration program so that it implements the faster way of calculating the area?

Give this a try. Do you get the same results? You should.



Thanks!

(Answers to questions follow....)

1. How would you write a program that prints out "Hello World!" ?

2. What format specifier would you use to print an integer variable?

3. What's wrong with the following?:

```
double x;
scanf("%lf",x);
```

4. What's the index of the last element in an array defined like this?:

```
int values[10];
```

Answers:

1.

```
#include <stdio.h>
int main () {
   printf ("Hello World!\n");
}
```

2. %i

3. You need a "&" before the x in the scanf statement. Remember, numeric variables always need a "&" in scanf.

4. 9. The indices go from 0 to 9.

1. How would you write a loop that prints out the numbers 1 through 10?

2. What's wrong with the following?:

```
if ( x=1 ){
  printf( “X equals one\n” );
}
```

3. What's wrong with this?:

```
double x[3];
x[1] = 1.0;
x[2] = 2.0;
x[3] = 3.0;
```

Answers:

1.

```
for (i=0;i<10;i++){
  printf ( “%d\n”, i+1);
}
```

2. It should be (x==1). Use “=” to assign values to things, but “==” to compare things.

3. The indices should be 0,1,2. There is no x[3].

1. Is it safe to assume that variables have a value of zero before they're used? Why or why not?

2. What C function would you use to open a file?

3. What C function would you use to write into a file?

4. Why won't the following program compile?:

```
#include <stdio.h>
int main () {
  int i=0
  printf (“The value of i is %d\n”, i);
}
```

Answers:

1. No. Before we set a variable's value, it has whatever random stuff was left in that memory location by the last program that ran.

2. fopen, like:

```
FILE *output = fopen(“myfile.dat”,”w”);
```

3. fprintf, like:

```
fprintf( output, “%lf %lf\n”, x, y);
```

4. The line “int i=0” should have a semicolon at the end!

1. What Linux command would you use to compile a C or C++ program?

2. What Linux command would you use to edit or create the file “program.cpp”?

3. What Linux command would you use to copy file.cpp to file2.cpp?

4. What Linux command would you use to list your files?

Answers:

1. g++ -Wall -o file file.cpp

2. nano program.cpp

3. cp file.cpp file2.cpp

4. ls

1. Why won't this work?:

```
char numbers[5] = “12345”;
```

2. What's the right way to do the following?:

```
char a[] = “stuff”;
char b[] = “stuff”;
if ( a == b ) {
  printf(“They're equal\n”);
}
```

3. What format specifier would you use to write a character string?

Answers:

1. It should be at least “numbers[6]” to leave room for the NUL character that has to indicate the end of the string.

2. Instead of “a == b”, you should use strcmp, like this:

```
if ( !strcmp(a,b) ) {
```

3. %s