Overview

• Input and output (I/O)
  – Files

• Scope
  – Global variables
  – More on functions

• The art of programming
  – Before starting
  – Program structure/layout
  – Comments

• Root finding
I/O: talking to the computer

Your program
Generic stream I/O

How to access a pipe?
- with a function!
Generic stream I/O

3 steps:
1. Open the pipe (stream)
2. Send/receive data
3. Close the pipe (stream)
1. Opening a file (stream/pipe)

- Must decide: input or output? (read or write?)
- New data type (File type)
  - In addition to Int, Float, Double, etc

```python
FileIn = open("input.data", "r")
```

FileIn is the name of the “pipe” (variable of type File)

input.data is the name of the file
1. Opening a file (stream/pipe)

- $r$ is the access "mode"

Modes:
- $r$ – read
- $w$ – write
- $a$ – append

```
FileIn = open("input.data", "r")
...```

1. Opening a file (stream/pipe)

Can open two pipes at the same time (must have different names)

Modes:
- r  – read
- w  – write
- a  – append

... FileIn = open("input.data", "r")
    FileOut = open("stuff.out", "w")
    ...

Note: if output file (stuff.out) doesn’t exist, it will be created (if it does exist, all data inside will be lost unless mode is “a”)
1. Opening a file

- input.data
- FileIn
- Your program
- FileOut
- stuff.out
3. Closing a file (stream/pipe)

• Easy!

```python
FileIn = open("file.data", "r")
...
FileIn.close()
```
2. Writing to a file

• First: text

```python
FileOut = open("stuff.out", "w")
FileOut.write("my data\n")
```

write is a function which writes the data

\n means newline
2. Writing to a file

- (slightly) more difficult: outputting numbers

```python
x = 3.142

FileOut = open("stuff.out", "w")
FileOut.write("my data
")
FileOut.write("%f
" % x)
```

%f means "float" format (e.g. 3.142)
2. Writing to a file

- Converting numbers to output (write & print)

print "format-string" % (variables)

%f means "float" format
%d means "integer" format
%e means "exponential" format

print "%f" % (pi)
print "%e %e %e\n" % (x, y, z)
print "number %d is %f\n" % (7, 45.6)
2. Reading from a file

```
FileIn = open("input.data", "r")
line = FileIn.readline()
data = split(line)
i = int(data[0]);
x = float(data[1]);
```

**Python**

```
FileIn = open("input.data", "r")
line = FileIn.readline()
data = split(line)
i = int(data[0]);
x = float(data[1]);
```

_line is a string variable (line = “4  3.142”)_
2. Reading from a file

- Python

```python
FileIn = open("input.data", "r")
line = FileIn.readline()
data = split(line)
i = int(data[0])
x = float(data[1])
```

**split** is a function which splits line into the bits between spaces.

**data** is a list

(data = "4", "3.142")
2. Reading from a file

- Python

```python
fin = open("input.data", "r")
line = fin.readline()
data = split(line)
i = int(data[0]);
x = float(data[1]);
```

```
file.data
4 3.142
...
i x
```

`int()` is a function which converts the string “4” into the integer 4

(i = 4)

(same thing for x)
B. Scope (or namespace)
B. Scope

- Variables declared within a function can only be used within that function

```python
def average():
    return (y + x)/2

y = 3.0
x = 5.6
a = average()
...
```
B. Scope

- Variables declared within a function can only be used within that function

```python
def average():
    return (y + x)/2

y = 3.0
x = 5.6
a = average()  # Doesn't work
```
B. Scope

- Variables declared within a function can only be used within that function

```python
def average():
    return (y + x)/2

y = 3.0
x = 5.6
a = average()
...
```

These refer to different variables!

```
main.x
average.x
```
B. Scope

• Variables declared within a function can only be used within that function

```python
def average(x, y):
    return (y + x)/2

y = 3.0
x = 5.6
a = average(x, y)
...
```

Correct way

Pass variables into function
B. Scope

• Note that argument “passed” to a function do not need to have the same name as those defined in the function itself.

```python
def average(x, y):
    return (y + x)/2

d = 3.0
e = 5.6
a = average(d, e)
Print average(1.5, 1.8)
...
```

Can also pass constant arguments to a function
B. Scope

- Variables are not “visible” everywhere
  - Only within the local function

```python
def average(x, y):
    y = 3.0
    x = 5.6
    a = average(x, y)
    return (y + x)/2
```
B. Scope

- Can declare “global” variables at top of file

```python
OmegaM = 0.7
OmegaL = 0.3

def E(z):
    return sqrt(OmegaM*pow(1+z,3)+OmegaL)

Print E(8.5)
```

Declared before main (global variable)

But cannot assign to global variable in a function
B. Scope: Global variables

- Global variables visible everywhere (in file)

```
def E(x, y):
    return (OmegaM...
```

OmegaM = 0.3

Declared before main (global variable)
B. Scope

- But cannot assign to global variable in a function

```python
ntimes = 0

def average(x, y):
    ntimes = ntimes + 1
    return (x + y)/2

y = 3.0
x = 5.6
a = average(x, y)
...
print "called %d times\n" % ntimes
```

Declared before main (global variable)
B. Scope

- Can lead to unexpected behavior

```python
x=24
def foo():
    y=2
    print x+y

def bar():
    x=11
    foo()

>>> bar()
26
```
Aside on functions

Passing data into functions: the \textbf{wrong} way to do it

\begin{align*}
\text{C program} & \\
\text{float tickle(double } x) \\
& \{ \\
& \quad x = (x*2-3.5)/1.8; \\
& \} \\
\text{PYTHON program} & \\
\text{def tickle(x):} \\
& \quad x = (x*2-3.5)/1.8
\end{align*}
Aside on functions

Passing data into functions: the **wrong** way to do it

This has no effect, because `x` is a local copy!
- variables set within a function are thrown away after the function is finished.

(exception: arrays)
Aside on functions

Passing data into functions: the **wrong** way to do it

C program

```c
float tickle(double x) {
    x = (x*2-3.5)/1.8;
}

x = 4.0;
tickle(x);
tickle(4.0);
```

PYTHON program

```python
def tickle(x):
    x = (x*2-3.5)/1.8;

x = 4.0
tickle(x)
tickle(4.0)
```
Aside on functions

Passing data into functions: the **right** way to do it

### C program

```c
float tickle(double x)
{
    double result;
    result = (x*2-3.5)/1.8;
    return result;
}
...
... 
x = tickle(x);
...
```

### PYTHON program

```python
def tickle(x):
    result = (x*2-3.5)/1.8
    return result
...
... 
x = tickle(x)
...
The art of programming

• Before writing any code, stop and think
  – What needs to be done?
  – What are the steps to do it?
    • Write out “pseudo-code” (or flowchart)
    • Variables required?
  – Go through a short example if possible
A prime number algorithm

Step 1: List out the numbers from 1 to n
Step 2: Strike out all numbers divisible by 2 (except 2)
Strike out all numbers divisible by 3 (except 3)
and so on for 7,11,13…
Step 3: List out the remaining numbers, they are prime.
Prime number variables:

- Array of numbers (1000)
- Integer for sqrt(1000)
- Integers for loops (i,j)
Program structure

- Put often repeated steps into function
  - Functions at top of program
- Use consistent indentation

C program

```c
... for (i=1; i < n; i++)
{
    x = findroot(coeffs, n);
    sum = sum + x;
}
...
```

PYTHON program

```python
... for i in range(1, n):
    x = findroot(coeffs, n)
    sum = sum + x
...
```
Program structure

• Use descriptive variable names
  – (e.g. sum instead of s)

C program

```c
... for (i=1; i < n; i++)
{
    x = findroot(coeffs, n);
    sum = sum + x;
}
...
```

PYTHON program

```python
... for i in range(1, n):
    x = findroot(coeffs, n)
    sum = sum + x
...
```
• In Python: anything after a `#` symbol
Readability

C program

...  /* sum the roots of the polynomial */
for (i=1; i < n; i++)
{
    x = findroot(coeffs, n);
    sum = sum + x;
}
...

PYTHON program

...  # sum the roots of the polynomial
for i in range(1, n):
    x = findroot(coeffs, n)
    sum = sum + x
...

Compare to:

...  for (i=1; i < n; i++) { x = f(c, n);
    s = s + x;
}
...

...  for i in range(1, n):
    x = f(c, n)
    s = s + x
...

C is particularly easy to abuse
Q. What does this program do?

```c
#include <stdio.h>
#include <math.h>
double l;main(_,o,O){return putchar((_+-+22&&_ +44&&main(_,43,_,_&o)?(main(-43,++o,O),((l=(o+21)/ sqrt(3-O*22-O*O),l*l<4&&(fabs((time (0)-607728)%2551443)/405859.-4.7+acos(l/2))<1.57))[#"]):10);}
```
Q. What does this program do?

A. Prints the phase of the moon
The International Obfuscated C Code Contest

#define __ -F<00||--F-OO--;
int F=00,OO=00;main(){F_OO();printf("%.3f\n",4.*F/00/00);}F_OO()
{


http://www.ioccc.org/
1. Nonlinear Equations
(root finding)

One dimensional:
\[ f : \mathbb{R} \rightarrow \mathbb{R} \]
- e.g. \( f(x) = ax-b = 0 \) (linear)
- or \( f(x) = \sin(2x)-x = 0 \) (nonlinear)

Multidimensional
\[ f : \mathbb{R}^n \rightarrow \mathbb{R}^n \]
- e.g.
\[
\begin{align*}
\{ f(x,y) &= x^2+y^2+xy-1=0, \\
g(x,y) &= x^2+y^2-xy-1=0 \}
\end{align*}
\] (nonlinear: two ellipses)
1-D Root-Finding Methods

• Bisection Method
• Newton’s Method
• Secant Method
• Others…
Bisection Method (Binary Search)
Bisection Method (Binary Search)

\[ a \quad \frac{(a+b)}{2} \quad b \]
Bisection Method (Binary Search)
Bisection Method (Binary Search)

Given \( f() \), tol, and \([a,b]\) such that \( \text{sign}(f(a)) \neq \text{sign}(f(b)) \):

\[
\text{while } b-a > \text{tol}: \\
m = \frac{(a+b)}{2} \quad \# \text{midpoint} \\
\text{if } \text{sign}(f(a)) == \text{sign}(f(m)):\n\quad a = m \quad \# \text{recurse on right half} \\
\text{else:} \\
\quad b = m \quad \# \text{recurse on left half}
\]

Guaranteed convergence! (if you can find initial a,b) but only at a linear rate: \(|\text{error}_{k+1}| \leq c|\text{error}_k|, \ c<1\)
Newton’s Method

![Graph showing Newton's Method with points $x_0$ and $x_1$.]
Newton’s Method
Newton’s Method

Given $f()$, $f'(())$, tol, and initial guess $x_0$

$k = 0$
do

\[ x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \]

$k++$

while $x_k - x_{k-1} > tol$

Can diverge (especially if $f'' \approx 0$).

If it converges, does so at \textbf{quadratic rate}:

\[ |e_{k+1}| \leq c|e_k|^2. \]

Requires derivative computation.
Newton’s Method Can Diverge
Secant Method

Like Newton, but approximates slope using point pairs

Given $f()$, tol, and initial guesses $x_0$, $x_1$

$k = 1$

do

$$x_{k+1} = x_k - \frac{f(x_k)(x_k - x_{k-1})}{(f(x_k) - f(x_{k-1}))}$$

$k++$

while $x_k - x_{k-1} < \text{tol}$

Can diverge.

If it converges, does so at rate 1.618: $|e_{k+1}| \leq c|e_k|^{1.618}$. 
Secant Method
Linear Fractional Interpolation

Instead of linear approximation, fit with fractional linear approximation: \( f(x) \approx (x-u)/(vx-w) \) for some \( u, v, w \)

Given \( f() \), tol, and initial guesses \( a, b, c, f_a = f(a), f_b = f(b) \)
do
\[
h = \frac{(a-c)(b-c)(f_a-f_b)f_c}{[(a-c)(f_c-f_b)f_a-(b-c)(f_c-f_a)f_b]}
\]
\[
a = b; \ f_a = f_b
\]
\[
b = c; \ f_b = f_c
\]
\[
c += h; \ f_c = f(c)
\]
while \( h > \text{tol} \)
If it converges, does so at rate \( 1.839 \): \( |e_{k+1}| \leq c|e_k|^{1.839} \).
1-D Root-Finding Summary

- Bisection: safe: never diverges, but slow.
- Newton’s method: risky but fast!
- Secant method & linear fractional interpolation: $f'(x)$ not required, mid-speed.

- Hybrids: use a safe method where function poorly behaved, Newton’s method where it’s well-behaved.
Multi-dimensional Root-Finding

• No general good way to do this…
• Once close to a root, use multi-dimensional Newton-Raphson