As you arrive:

1. Start up your computer and plug it in
2. ***Log into Angel*** and go to CSSE 120
3. Do the ***Attendance Widget*** – the PIN is on the board
4. Go to the course ***Schedule Page***
5. Open the ***Slides*** for today if you wish
6. Check out today's project: **`Session21_C_Example1`**

Plus in-class time working on these concepts AND practicing previous concepts, continued as homework.

First C program

- Structure of a main file
- Defining and calling functions
- Definite (FOR) loops
- Declaring variables

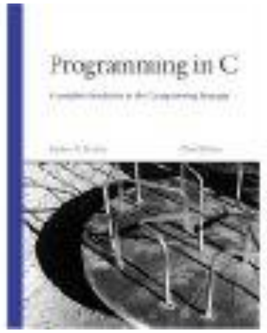
Project demonstrations

Announcements

- Exam 2 Monday
  - As described on the course schedule page
- Homework due Session 23 (Tuesday), per the course schedule page
  - Identify any problems in your Eclipse setup for C
  - Reading from course web site
  - No ANGEL quiz

Reminder: OPTIONAL C Textbook

- Kochan’s “Programming in C”
- Very readable, like Zelle.
- Recommended highly by two non-CSSE Rose professors
- We will add more resources on the schedule page
- See the **Course Resources** page
  - Linked from course’s main page
  - Explore Python_vs_C_Comparison document



```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

void squareRootTable(int n);

int main() {
    squareRootTable(20);
    return EXIT_SUCCESS;
}

void squareRootTable(int n) {
    int k;

    for (k = 1; k <= n; ++k) {
        printf("%3i  %9.3f\n", k, sqrt(k));
    }
}
```

```python
from math import sqrt

def squareRootTable(n):
    for k in range(1, n+1):
        print("{:3i}  {:9.3f}"
              .format(k, sqrt(k)))


def main():
    squareRootTable(20)


if __name__ == '__main__':
    main()
```

Parallel examples in Python and C

Comments in C

- Python comments begin with # and continue until the end of the line.
- C comments begin with /* and end with */.
- They can span any number of lines.
- Some C compilers (including the one we are using) also allow single-line comments that begin with //.

The inclusion of header files

`#include` is somewhat like Python's **`from`** ... **`import *`**

The most commonly included files are *header* files, whose names end with **.h**

```
#include <stdio.h>
#include <math.h>
```

angle brackets mean that it is a standard C header

If we include a file from our own project, surround it's name with quotes, as in **#include "myFile.h"**

A header file usually contains definitions of constants, and function signatures (without their bodies)

Two lines from **math.h** (we'll explain later):

```
#define M_PI 3.14159265358979323846
double sqrt (double);
```

Other headers: http://www.utas.edu.au/infosys/info/documentation/C/CStdLib.html

Focus on the `main()` Function

```
#include <stdio.h>
#include <math.h>
```

Every C program must have a function named **main()**

`main`'s return value (In this case 0) is the exit status of the program. Usually, we return 0 to indicate successful completion of the program

This **main( )** function has an empty formal parameter list

In a function definition, we must indicate its return type before the name of a function, - In this case, the return type is **int**

```
int main() {
   squareRootTable(10);
   return 0;
}
```

The body of a function definition is enclosed in curly braces { … }

Every simple C statement must be followed by a semicolon

The two statements in the body are just like corresponding Python statements

By looking at **main**, how can we tell that **printRootTable** doesn't have to return a value?

`printRootTable()`'s interface

```
#include <stdio.h>
#include <math.h>

void printRootTable(int n) {


}
```

What is the name of the "return type" of the printRootTable() function?
What does that mean?

The formal parameter is called **n**, its type is **int**

Note that this function has no **return** statement. In that case, the return type **must** be declared to be **void**

The type of every formal parameter must be declared

```
int main() {
   squareRootTable(10);
   return 0;
}
```

As in Python, if there are multiple formal parameters, they are separated by commas

As in Python, when printRootTable is called, the value of the actual parameter (10) is used to initialize the formal parameter (n)

Notice that we do not provide the type of the actual parameter. Its type is the type of whatever value we pass in. It must "match" the type of the formal parameter

(local) variable declaration

```
#include <stdio.h>
#include <math.h>

void printRootTable(int n) {
   int i;



}

int main() {
   printRootTable(10);
   return 0;
}
```

i is a local (to the function) variable of the **printRootTable** function

Its type is **int**

Unlike in Python, each C variable's and formal parameter's type must be declared before the variable can be used

Variable declarations must include a type. An optional initialization is allowed, such as `int i = 17;` or `int i = n + 5;`

A local variable cannot have the same name as a formal parameter of the same function

Because the variables **i** and **n** are local to printRootTable, you cannot refer to them from anywhere else in the program

i++

- `i++` is an abbreviation for `i = i + 1`
  - which can also be written `i += 1`
- `i--` is an abbreviation for `i = i - 1`
  - which can also be written `i -= 1`
- Some C-programmers write i++ or i-- as part of a more complicated expression.
  - We suggest that you avoid doing that for now.

C's for loop

```
#include <stdio.h>
#include <math.h>

void printRootTable(int n)
   int i;
   for (i=1; i<=n; i++) {
      printf(" %2d  %7.3f\n", i, sqrt(i));
   }
}
```

Basic syntax is

```
for (<init>; < test>; <update>) {
   body
}
```

- **init:** usually initializes variables used by the loop
- **test:** if the value of the test is true, the loop body executes
- **update:** After execution of the loop body, this code is executed. Then the **test** code is evaluated again, and if true …