C LANGUAGE INTRODUCTION

CSSE 120—Rose Hulman Institute of Technology

The C Programming Language

- Invented in 1972 by Dennis Ritchie at AT&T Bell Labs.
- Has been the main development language for UNIX operating systems and utilities for a couple of decades.
- Our Python interpreter was written in C!
- Used for serious coding on just about every development platform.
- Especially used for embedded software systems.
- Is usually compiled to native machine code.
  - Faster and less portable than Python or Java.

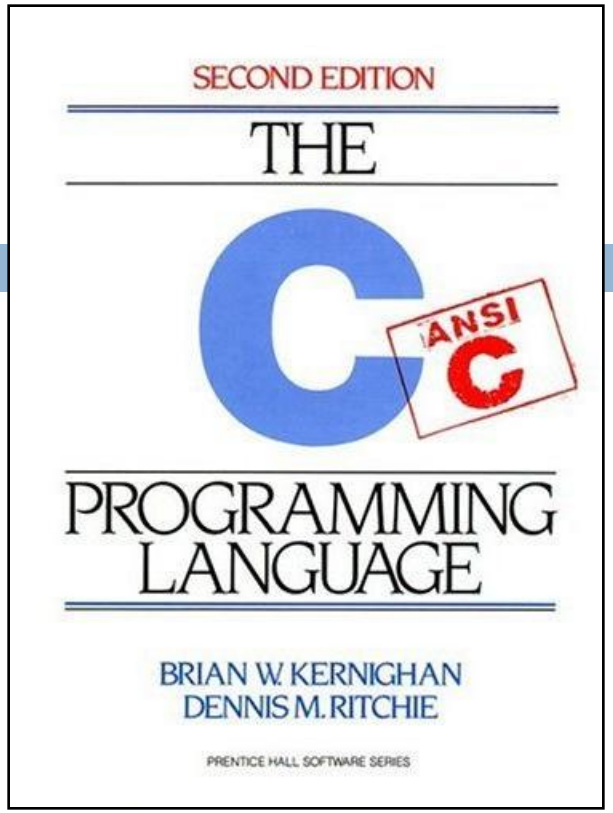
Why C in CSSE 120?

- Practical
  - Several upper-level courses in CSSE, ECE, and Math expect students to program in C.
  - None of these courses is a prerequisite for the others.
  - So each instructor has a difficult choice:
    - Teach students the basics of C, which may be redundant for many of them who already know it, or
    - Expect students to learn it on their own, which is difficult for the other students.
  - A brief C introduction here will make it easier for you (and your instructor!) when you take those courses.

Why C in CSSE 120?

- Pedagogical
  - Comparing and contrasting two languages is a good way to reinforce your programming knowledge.
  - Seeing programming at C's "lower-level" view than Python's can help increase your understanding of what really goes on in computing.
  - Many other programming languages (notably Java, C++, and C#) share much of their syntax and semantics with C.
    - Learning those languages will be easier after you have studied C.

Classic C text/reference



Product Details

Paperback: 274 pages

Publisher: Prentice Hall PTR; 2 edition (March 22, 1988)

Language: English

ISBN-10: 0131103628

ISBN-13: 978-0131103627

Product Dimensions: 9.1 x 6.9 x 0.8 inches

Shipping Weight: 1.3 pounds (View shipping rates and policies)

Average Customer Review: ★★★★½ based on 241 reviews. (Write a review.)

Amazon.com Sales Rank: #3,121 in Books (See Bestsellers in Books)

Pretty amazing for a 20-year-old programming book!

For comparison, Harry Potter #3's rank is 25,578.

Some C Language trade-offs

- Programmer has more control, but fewer high-level language features to use.
- Strong typing makes it easier to catch programmer errors, but there is the extra work of declaring types of things
- Lists and classes are not built-in, but arrays and structs can be very efficient
  - and a bit more of a pain for the programmer.

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 //.

Parallel examples in Python and C.

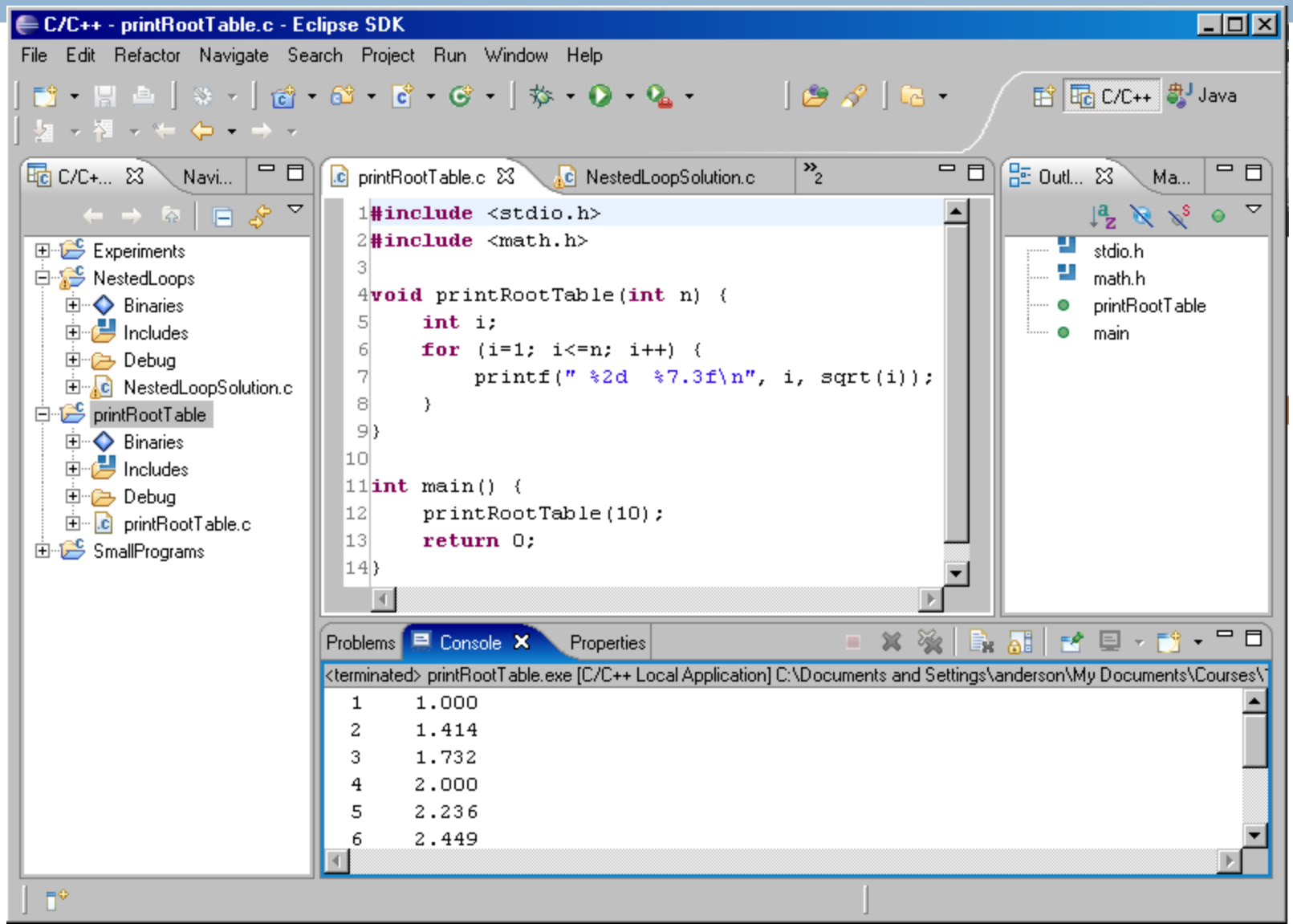
```python
from math import *

def printRootTable(n):
    for i in range(1,n):
      print " %2d  %7.3f" % (i, sqrt(i))

def main():
    printRootTable(10)

main()
```

```c
#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));
    }
}

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

Using C with Eclipse

- We assume that you have already installed the MinGW compiler and C++ tools for Eclipse, as described in the Installation links from the course's Resources page on ANGEL
- You must use a different Eclipse workspace for your C programs than the one you use for Python programs. If you have not already created it,
  - In Windows explorer, create a folder to use for this
  - File→ Switch Workspace, then the Browse button
  - Browse to the folder you created. Click OK

C/C++ perspective



Starting a New Project

- New→Managed Make C Project (Call it **RootTable**)
- Right-click the project, choose New C Source File.
- Call the file **rootTable.c** . Finish.
- Note that if you right-click rootTable.c,
  Run as … is missing from the context menu.
  - Why? unlike in PyDev, each Eclipse C Project must have exactly one code file containing the **main()** function.
  - Thus **Run As …** is not even an option for an individual C code file.

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>


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

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**

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) {


}

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

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

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

```
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 is evaluated again, and if true …

String constants in C

- In Python, character strings can be surrounded by single quotes (apostrophes), or double quotes (quotation marks).
- In C, only double quotes can surround strings (whose type in C is char*).
  - ```
    char *s = "This is a string";
    printf(s); /* more about printf() soon */
    ```
- Single quotes indicate a single character, which is not the same as a string whose length is 1. Details later.
  - ```
    char c = 'x';
    printf("%c\n", c);
    ```

printf statement

```
C:      printf(" %2d  %7.3f\n", i, sqrt(i));
```

```
Python equivalent: print " %2d  %7.3f" % (i, sqrt(i))
```

- printf's first parameter is used as a format string.
- The values of **printf**'s other parameters are converted to strings and substituted for the conversion codes in the format string.
- **printf** does not automatically print a newline at the end.

printf – frequently used conversion codes

code	data type	Example

d	decimal (int, long)	`int x=4, y=5;`
		`printf("nums %3d, %d%d\n", x, y, x+y");`
		`/*prints nums  4, 59*/`

f	real (float)	`float p = 1.3/9, q = 2.875;`
		`printf ("%7.4f %0.3f %1.0f %f\n", p, p, q, q);`
		`/* prints  0.1444 0.144 3 2.875000 */`

lf	real (double)	`double p = 1.3/9, q = 2.875;`
		`printf ("%7.4f %0.3f %1.0f %f\n", p, p, q, q);`
		`/* prints  0.1444 0.144 3 2.875000 */`

c	character (char)	`char letter = (char)('a' + 4);`
		`printf ("%c %d\n", letter, letter);`
		`/* prints e 101 */`

s	string (char *)	`char *isString = "is";`
		`printf("This %s my string\n", isString);`
		`/* prints This is is my string! */`

e	real (scientific notation)	`double c = 62345892478;`
		`printf("%0.2f %0.3e  %14.1e", c, c, c);`
		`62345892478.00 6.235e+010  6.2e+010`

Rectangular output in C

```
#include <stdio.h>

void rectangleOfIs(int numRows, int numCols) {
        int i, j;
        for (i=1; i<= numRows; i++) {
                for (j=1; j<=numCols; j++)
                    printf ("%d", i);
                printf ("\n");
        }
}
int main() {
        rectangleOfIs(3, 8);
}
```

Output:

```
11111111
22222222
33333333
```

It's easier than Py$_{thon}$ because `printf()` does not automatically add spaces like Python's `print`.