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 about 30 years
- 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 but less portable than Python or Java
  - Compiled, not interpreted, so no interactive mode

Why C in CSSE 120?

- Practical
  - Several upper-level courses in CSSE, ECE, ME, and Math expect students to program in C
  - None of these courses is a prerequisite for the others.
  - So each instructor had 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
  - But 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 a program
  - Many other programming languages (notably Java, C++, and C#) derive much of their syntax and semantics from C
    - Learning those languages will be easier after you have studied C

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 thing
  - "Once an int, always an int"
- Lists and classes are not built-in, but arrays and structs can be very efficient
  - and a bit more challenging for the programmer

```
from math import *

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

def main():
    printRootTable(10)

main()
```

Parallel examples in Python and C.

Next slides go through this example in detail.

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

void printRootTable(int n);

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

```
void printRootTable(int n) {
    int k;

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

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

void printRootTable(int n);

int main() {
    printRootTable(10);
    return EXIT_SUCCESS;
}

void printRootTable(int n) {
    int k;

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

Next slides use this example to show ten ways that C differs from Python.

How C differs from Python, #1:

#include instead of **import**

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

void printRootTable(int n);

int main() {
    printRootTable(10);
    return EXIT_SUCCESS;
}

void printRootTable(int n) {
    int k;

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

How C differs from Python, #2:

Functions (except *main*) should have ***prototypes*** which specify the form of the function

simple C statements end in a semicolon

In the prototype

```c
void printRootTable(int n);
```

has a single parameter that is an int (i.e. integer)

doesn't return anything

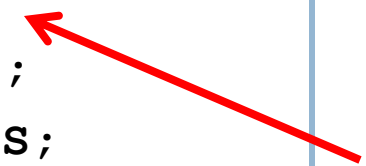
```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

void printRootTable(int n);

int main() {
    printRootTable(10);
    return EXIT_SUCCESS;
}

void printRootTable(int n) {
    int k;

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



How C differs from Python, #3:

Execution starts at the special function called *main*. Every C program has exactly one *main* function.

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

void printRootTable(int n);

int main() {
    printRootTable(10);
    return EXIT_SUCCESS;
}

void printRootTable(int n) {
    int k;

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

How C differs from Python, #4:

Bodies of functions, loops, if clauses, etc., are not delimited by indentation. Instead, *curly-braces* begin and end the body.

Note the style for where the braces are placed. Use this style.

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

void printRootTable(int n);

int main() {
    printRootTable(10);
    return EXIT_SUCCESS;
}

void printRootTable(int n) {
    int k;

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

How C differs from Python, #5:

Simple C statements end in a *semicolon*.

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

void printRootTable(int n);

int main() {
    printRootTable(10);
    return EXIT_SUCCESS;
}

void printRootTable(int n) {
    int k;

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

How C differs from Python, #6:

All variables must have their *type declared* at the point the variable is introduced. Parameters, local variables, and return value from functions. Types include:

-- int for integers

-- double and float for floating point numbers

-- char for characters

For return values from functions, void means nothing is returned.

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

void printRootTable(int n);

int main() {
    printRootTable(10);
    return EXIT_SUCCESS;
}

void printRootTable(int n) {
    int k;

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

How C differs from Python, #7:

No lists or range expressions. The *for statement* is more primitive:

Parentheses, no colon at end

```c
for (k = 1; k <= n; ++k)
```

k starts at 1

loop continues while k <= n

Note that semicolons separate the 3 parts of a for loop

at end of each iteration of the loop, k increases by 1. ++k and k++ are shorthand for k = k + 1

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

void printRootTable(int n);

int main() {
    printRootTable(10);
    return EXIT_SUCCESS;
}

void printRootTable(int n) {
    int k;

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

How C differs from Python, #8: *printf* is similar but not identical to one way of using Python's print.

In the example:

-- note parentheses, quotes, commas

-- %2d means integer, using 2 spaces

-- %7.3f means floating point, using 7 spaces, 3 spaces after the decimal point (use just %f for floating point with default number of decimals)

-- %c for printing a character

-- \n means newline

-- double quotes for string literals

-- single quotes for character literals, e.g. `'R'` for the R character

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

void printRootTable(int n);

int main() {
    printRootTable(10);
    return EXIT_SUCCESS;
}

void printRootTable(int n) {
    int k;

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

How C differs from Python, #9:

if statements have their condition in parentheses, e.g.

```c
if (k <= n) {
    ...
}
```

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

void printRootTable(int n);

int main() {
    printRootTable(10);
    return EXIT_SUCCESS;
}

void printRootTable(int n) {
    int k;

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

How C differs from Python, #10:

comments are different:

```c
/*
 ... (multi-line comment)
 */

// single line comment
```

Using C with Eclipse

1. You must use a different Eclipse workspace for your C programs than the one you use for Python programs.
   - In Windows explorer, create a folder to use for your C projects
     - Important: Put it directly below the C drive, in a path with NO SPACES, e.g. C:\CProjects
   - Back in Eclipse: File→ Switch Workspace, then the Browse button
   - Browse to the folder you created. Click OK
2. In Eclipse, select Window → Open Perspective, then Other, then C/C++
   - You probably have a C/C++ perspective. But if you don't, follow the instructions at this link – ask for help walking through these instructions.

Using C with Eclipse (continued)

- Once you are in Eclipse in the C/C++ perspective, set your individual repository:
  - Window → Show View,
    then Other,
    then SVN → SVN Repositories
  - In the SVN Repositories tab that appears at the bottom, right-click and select New→ Repository Location
  - For the URL, enter
    http://svn.cs.rose-hulman.edu/repos/csse120-201030-USERNAME
    where you replace USERNAME with your own Kerberos username
- Checkout your 23-CForLoops project and browse the code in the *src* folder. Run the project (use the Run button).

Rest of today

- Work through the TODO's, as numbered.
- Ask questions as needed!
- Use this exercise to get comfortable with the basics of C notation, and C in Eclipse. Pay attention to what you are doing!
- Finish the exercise for homework