

CSc 352

C - Syntax, Numbers, Math, I/O

Benjamin Dicken

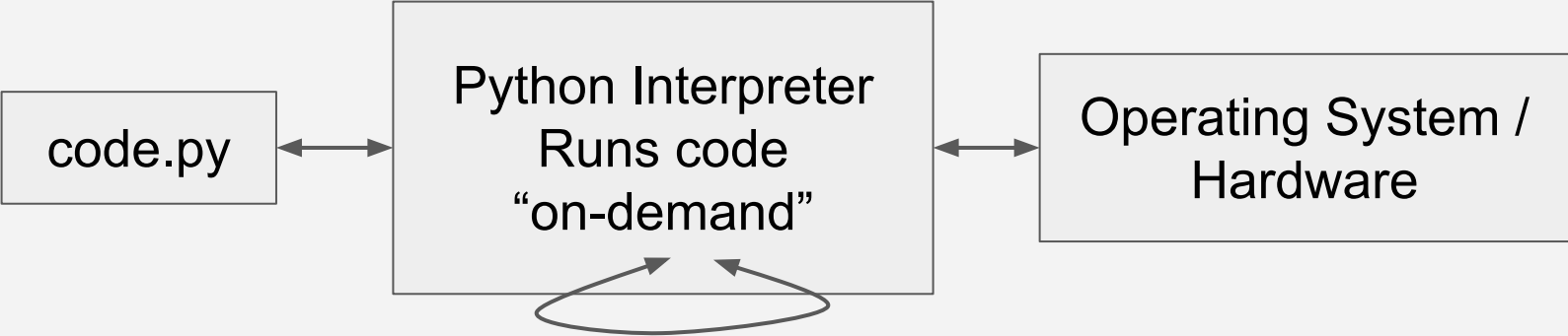
C Language

- Expectation: You already know Python and Java
- C syntax similar to Java, less so Python
 - Variables assignment, ifs, loops, curly-braces, etc

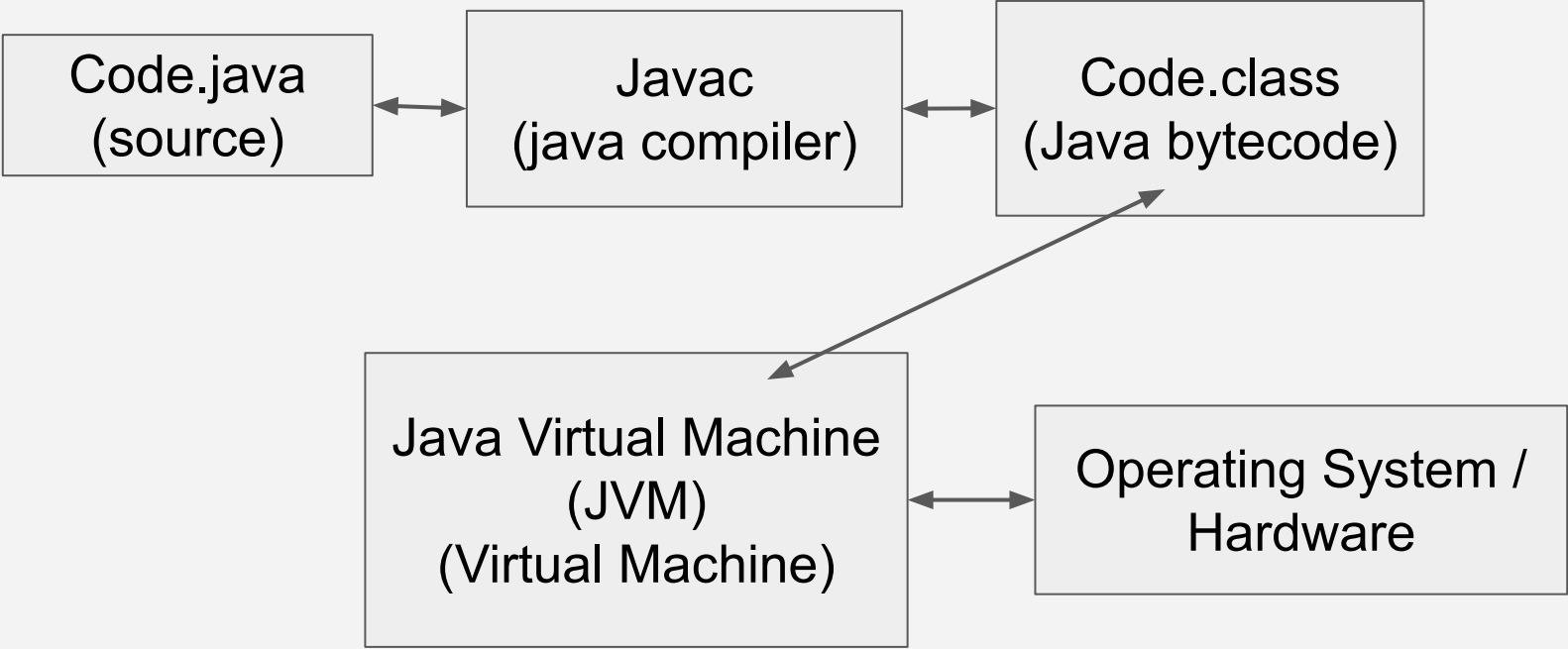
Differences between C and Java

- NOT object oriented (no classes, inheritance, methods, etc)
- Low-level (not run with interpreter / VM)
- Memory Management, Garbage Collection
- Pointers (similar to references)
- Less stuff is built-in, have to **#include** functionality
- No array boundary protection
- Less hand-holding :)

Running Python

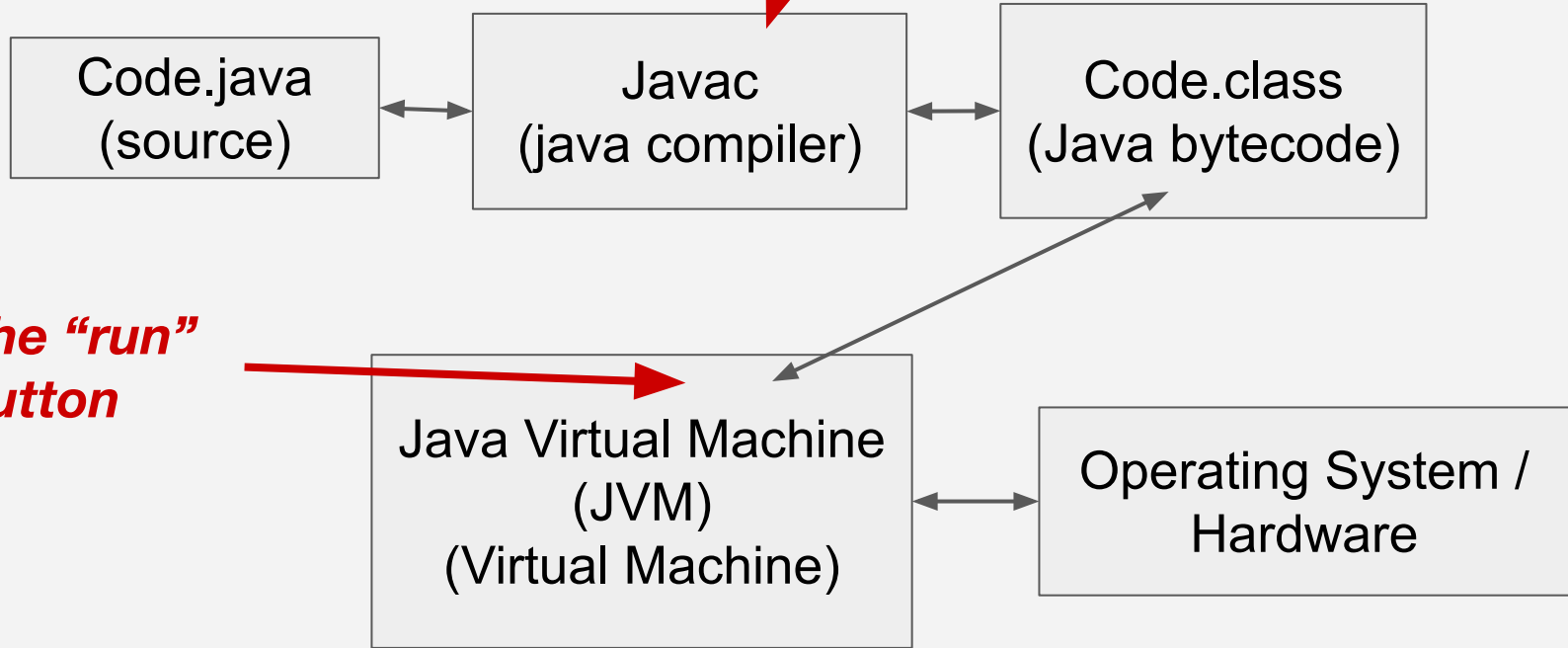


Running Java

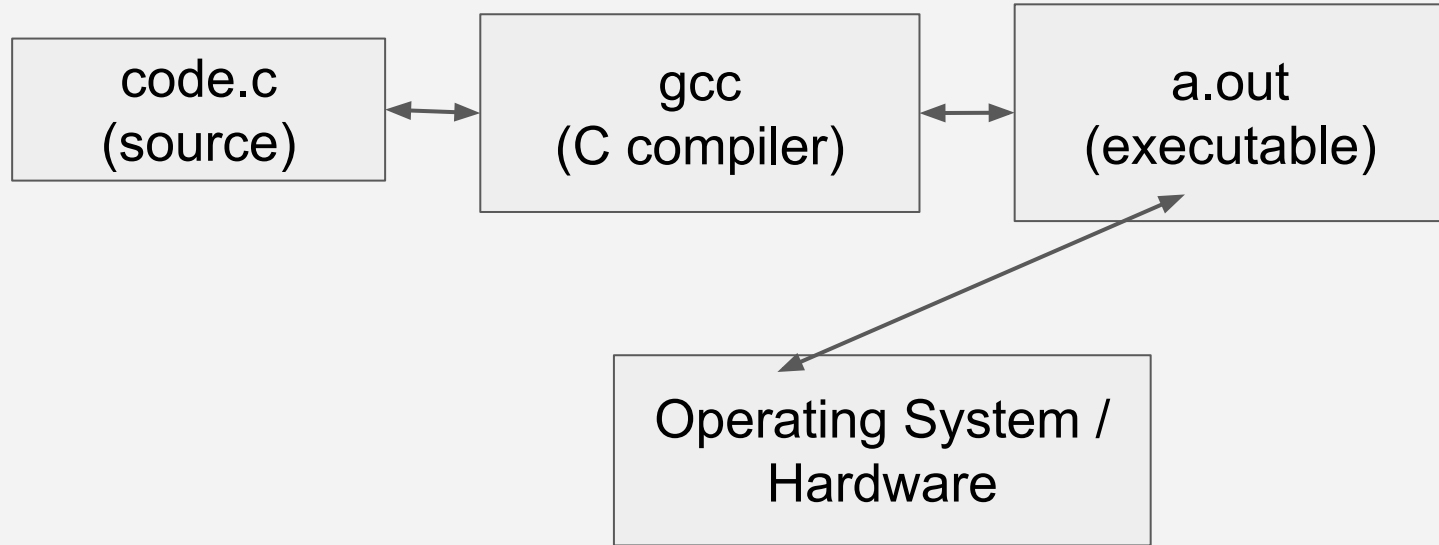


Running Java

*The “compile”
button*

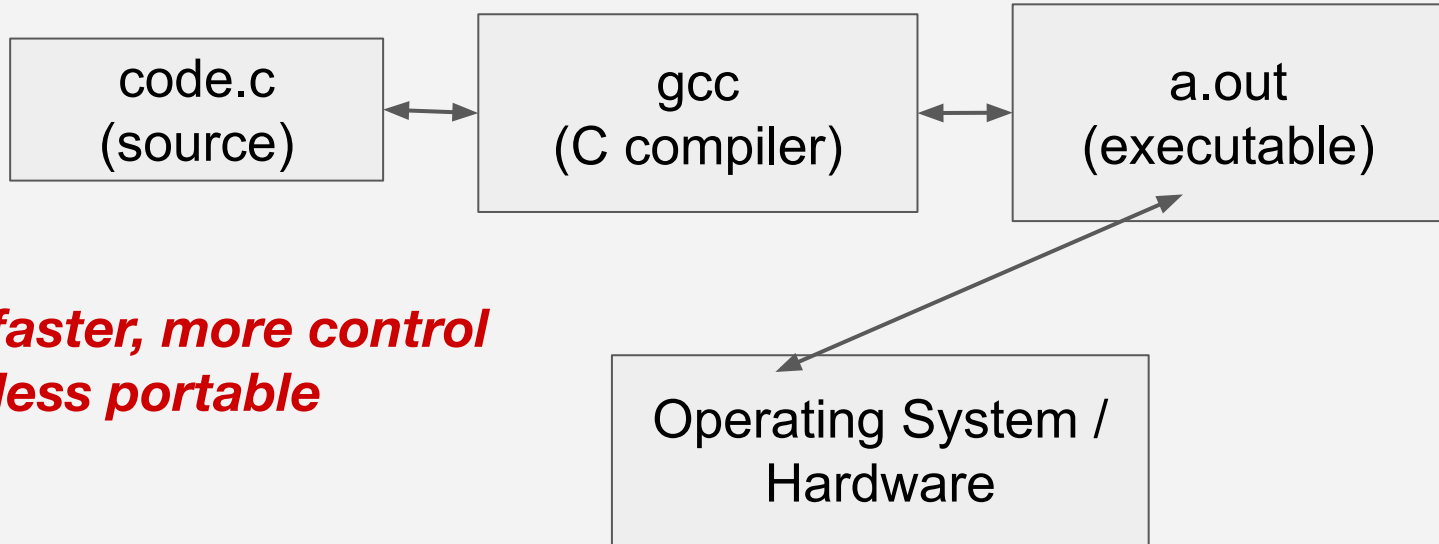


Running C



Running C

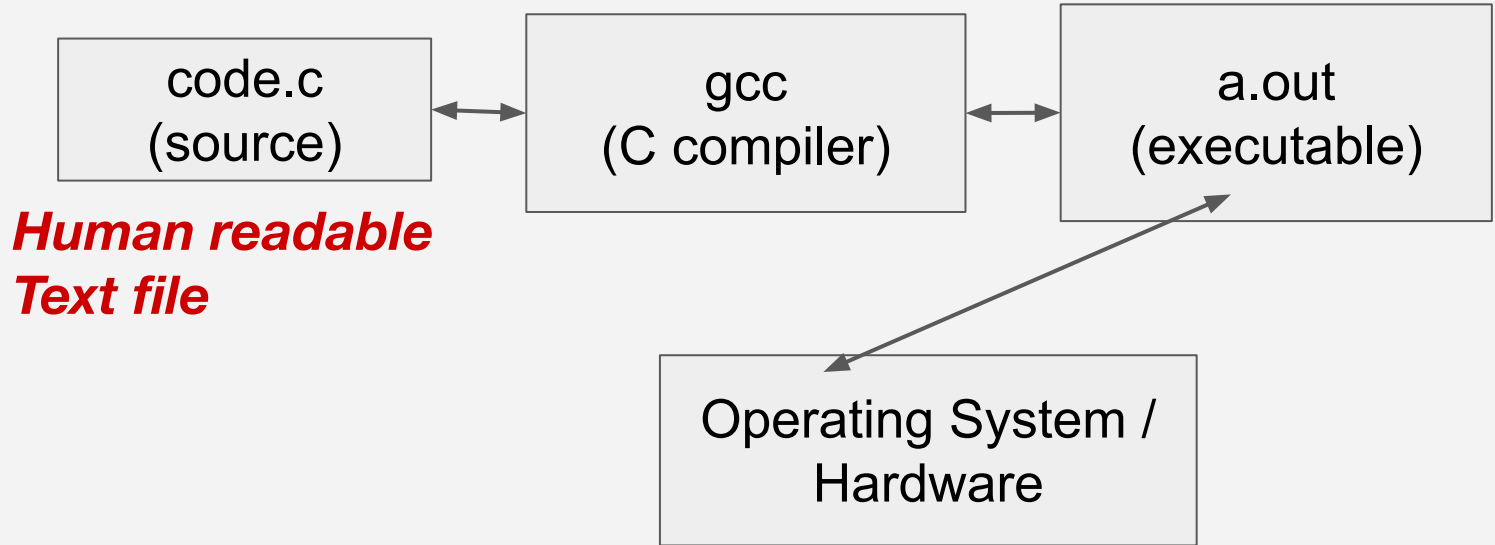
***Compiled and then run,
cuts out VM middle-man***



***PRO: faster, more control
CON: less portable***

Running C

Machine code
Not (easily) human readable
Binary file



C Compilers

- A program that takes C source code (text) as input, and produced an executable file (binary) that can run directly on an operating system, as output
- Two most popular: **clang** and **gcc**
- For this course: **gcc**

`$ man gcc` # so many options, what should we use?

gcc Options

-Wall

C compilers differentiate warnings from errors by default

Warnings can be turned on / off

This option enables **all** warnings

-Werror

Treat all warnings as errors

Won't compile unless there are *no* warnings / errors)

-std=c11

Multiple C standard / versions

For this class: C11 (as opposed to C89, C99, C17)

Compiling with gcc

```
$ ls
```

```
some_code.c
```

```
$ gcc -Wall -Werror -std=c11 some_code.c
```

```
$ ls
```

```
a.out      some_code.c
```

```
$
```

Compiling with gcc

```
$ ls
```

```
some_code.c
```

```
$ gcc -Wall -Werror -std=c11 some_code.c
```

```
$ ls
```

```
a.out      some_code.c
```

```
$
```

So what goes in a .c file?

Compile and run a C Program

```
int main() {  
    printf("hi\n");  
}
```

- Log on to lectura (or local)
- Create file named some_code.c
- Put this in it, then run:

```
$ gcc some_code.c  
$ ./a.out
```

Compile and run a C Program

```
int main() {  
    printf("hi\n");  
}
```

Now try:

```
$ gcc -Wall -Werror -std=c11 some_code.c
```

```
$ ./a.out
```

Also try with c89

Compiling with gcc

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

```
int main() {  
    printf("hi\n");  
    return 0;  
}
```


Compiling with gcc

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

```
int main() {  
    printf("hi\n");  
    return 0;  
}
```

return type, function name, param
sequence (like Java)

Function calls, arguments,
params, etc works similar to Java
(more on that in future class)

Curly-braces for functions
Also used for ifs, loops, scope
(like Java)

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

```
int age = 45;
```

```
int height = 104;
```

```
int main() {
```

```
    int weight = 180;
```

```
    printf("age: %d\n", age);
```

```
    printf("height: %d, weight: %d\n", height, weight);
```

```
    return 0;
```

```
}
```

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

Global variables of type int
C uses static types, like Java



```
int age = 45;  
int height = 104;
```

```
int main() {  
    int weight = 180;
```

Local integer variable



```
    printf("age: %d\n", age);  
    printf("height: %d, weight: %d\n", height, weight);
```

```
    return 0;
```

Print formatting
See: man 3 printf

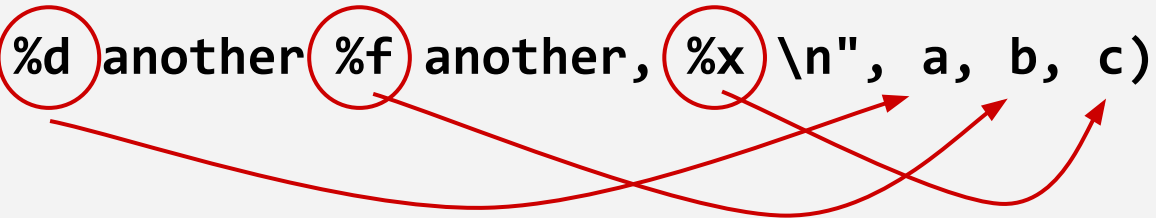


```
}
```

printf format strings

The first argument is a string that can contain regular characters, escape characters (starting with \) and conversion specifiers (starting with %)

```
printf("word %d another %f another, %x \n", a, b, c)
```



Number of conversion specifiers must match values following

Each conversion specifier can have multiple options

D for dec int, **x** for hex int, **f** for float number, etc

See man page

Announcements

PA 2 - <https://benjdd.com/courses/cs352/spring-2022/pas/pa2.pdf>

Style Guide - <https://benjdd.com/courses/cs352/spring-2022/style/>

Playlist

Man pages

Different types of man pages:

1. User commands
2. System Calls (OS / kernel functions)
3. Library calls (program libraries)
4. Special files (usually from /dev)
5. File formats and conventions
6. Games
7. Miscellan
8. System admin commands
9. Nonstandard Kernel Routines

When we see something like **CAT(1)** this tells us it is from category 1

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

```
int main() {
```

```
    int height = 0;
```

```
    int weight = 0;
```

```
    printf("Enter height: ");
```

```
    scanf("%d", &height);
```

```
    printf("Enter weight: ");
```

```
    scanf("%d", &weight);
```

```
    printf("\nYour height and weight is: ");
```

```
    printf("    height: %d, weight: %d\n", height, weight);
```

```
    return 0;
```

```
}
```

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

The scanf function is in the stdio library as well

```
int main() {
```

```
    int height = 0;
```

```
    int weight = 0;
```

```
    printf("Enter height: ");
```

```
    scanf("%d", &height);
```

```
    printf("Enter weight: ");
```

```
    scanf("%d", &weight);
```

Call scanf, specify expected type in the format string

Why the "&" ?

For now, just know that you need to put it there, will address further later

```
    printf("\nYour height and weight is: ");
```

```
    printf("    height: %d, weight: %d\n", height, weight);
```

```
    return 0;
```

Print the results

```
}
```


Averaging Numbers

Write a C program that:

- Asks the user for three integer numbers
- Computes the average
- Prints the result

Math in C

Most of the standard / simple math operators work as-expected

+ - / * % ++ --

Some of the more “advanced” operations in the `<math.h>` module

`.... exp(base, exponent) sqrt(number) fabs(a, b)`

Look at some man pages

(What are these “floats” and “doubles”?) (-lm)

Primitive Types in C

Integer representation
Floating-point representation

char short int long long long float double long double

- **Integers** can be preceded by **signed** or **unsigned** (signed default)
- Why so many types? Sizes
 - Some use different amount of bytes
 - Less bytes = less memory, but less range
- Keep in mind: behind the scenes, **all** of these types are just binary sequences of 1s and 0s

Primitive Types in C

<code>char</code>	min = 8 bits	-128 to 127
<code>short</code>	min = 16 bits	-32,768 to 32,767
<code>int</code>	min = 16 bits	-32,768 to 32,767
<code>long</code>	min = 32 bits	-2,147,483,647 to 2,147,483,647
<code>Long long</code>	min = 64 bits	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
<code>float</code>	typically 32 bits	
<code>double</code>	typically 64 bits	
<code>long double</code>	typically 128 bits	

- Specifics varies from machine to machine
- Use `sizeof()` and `limits.h`

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

```
int main() {
```

```
    int          a = 100;
```

```
    long long    b = 10000;
```

```
    float        c = 1.76891401;
```

```
    double       d = 12875.1002713;
```

```
    printf("Int: %d\n", a);
```

```
    printf("Long long: %lld\n", b);
```

```
    printf("Float: %f\n", c);
```

```
    printf("Double: %lf\n", d);
```

```
    printf("Double with four dec: %.4f\n", d);
```

```
    return 0;
```

```
}
```

What will it print?

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

```
int main() {  
    signed char x = 0;  
    printf("%ld\n", sizeof(x)); // prints out 1 (1 bytes = 8 bits)  
    printf("%d\n", x);  
    long i = 0;  
    while (i < 257) {  
        x = x + 1;  
        i += 1;  
    }  
    printf("%d\n", x);  
    return 0;  
}
```

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

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

```
void bin(char n) {  
    unsigned int i;  
    for (i = 1 << 7; i > 0; i = i / 2) {  
        (n & i) ? printf("1") : printf("0");  
    }  
    printf("\n");  
}
```

```
int main() {  
    char x = 127;  
    for (int i = 0; i < 3; i+= 1) {  
        printf("%d\n", x);  
        bin(x);  
        x += 1;  
    }  
    return 0;  
}
```

What will be in output.txt?

```
$ gcc -Wall -Werror -std=c11 -o add add.c  
$ echo "addition is: " >> output.txt  
$ cat input.txt | ./add > output.txt
```

```
20  
30
```

input.txt

```
#include <stdio.h>  
  
int main() {  
    int x1 = 0, x2 = 0;  
    scanf("%d", &x1);  
    scanf("%d", &x2);  
    printf("result: %d\n", (x1 + x2));  
    return 0;  
}
```

add.c

The first PA

At this point, you should know enough C for the first PA

The first PA is not about getting to from zero to hero

Use ssh, write bash commands, compile / run basic C, testing

Man pages

Look at the man pages for **scanf** and **printf**

What header file do those come from?

What will happen?

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

int main() {
    int x;
    printf("%d\n", x);
    return 0;
}
```

Value initialization

- C does not automatically initialize data of primitive types (Java too)
 - In Java - will give an error if you try to use one of these variables
 - In C - Unpredictable value
- C also does not auto-init array values, unlike Java

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

While functions basically the same as Java



```
int main() {
    int i = 0;
    while (i < 20) {
        printf("counting: %d\n", i);
        x += 1; // or i++;
    }
    return 0;
}
```

A thruthy / non-zero value will cause loop to continue

A falsy / zero value will stop the loop

++ and += both work in C



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

```
int main() {
    int x = 0;
    scanf("%d", &x);
    if (x < 100) {
        fprintf(stderr, "BAD INPUT\n");
        return 1;
    } else if (x < 80) {
        printf("large\n");
    } else if (x < 50) {
        printf("medium\n");
    } else {
        printf("small\n");
    }
    return 0;
}
```

if / else if / else chains also looks the same as Java

A thruthy / nonzero value will cause a true condition

A falsy / zero value will cause a false condition

How to print to stderr,
Also note return value

The diagram consists of four arrows pointing from the explanatory text on the right to the code on the left. The first arrow points from the text 'A thruthy / nonzero value will cause a true condition' to the condition 'x < 100'. The second arrow points from 'A falsy / zero value will cause a false condition' to the condition 'x < 80'. The third arrow points from 'How to print to stderr, Also note return value' to the 'fprintf(stderr, ...)' line. The fourth arrow points from the same text to the 'return 0;' line.

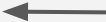
For loops also work!



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

```
int main() {
    for (int i = 0; i = 20; i ++ ) {
        printf("counting: %d\n", i);
    }
    return 0;
}
```

3 components within parentheses:

- (1) Run before first iteration
 - (2) Condition checked before each iteration
 - (3) Code executed at end of each iteration
- 

Tribonacci Numbers

Write a C program that:

- Asks the user for one numeric value **N** from standard input
- Prints out the first **N** numbers in the Tribonacci sequence
- The Tribonacci sequence is defined by:
 - First three numbers in sequence: 0, 0, 1
 - Fourth and on: Defined by the sum of the previous three numbers in the sequence
 - 0, 0, 1, 1, 2, 4, 7, 13, 24 . . .