Hiii! We'll start at 3:03pm. In the meantime, get ready to introduce yourself – name, pronouns, year, and where you are right now (House/dorm or location)!

CS50 Section 1

Phyllis Zhang

- Lubbock, TX!
- Sophomore in Pforzheimer House
- Studying CS (CS50, CS51, CS109A, STAT110, STAT115, MCB60, CHEMS17, MATH21A)
- Tutorials:
  - Thursday 4pm – 5pm ET
  - Saturday 12pm – 2pm ET
- Contact (24hr period):
  - Email: phylliszhang@college.harvard.edu
  - Text: 806-392-3098

Section Logistics

- Wednesday 3:05pm – 5pm ET (Take the first 5 minutes to do the problem on the screen)
- Breakout Rooms: Randomized in the first 3 sections; Based on Preference Survey afterwards
- Ask Questions! If asking in front of the class is uncomfortable, just private message me and I'll read/answer the question out loud
- Submit the lab we start today by tomorrow! Please do *not* start the lab before section.
- Required elements:
    - Submit Code for Exercises at the End of Every Section at *tinyurl.com/phyllis-cs50*
    - Cameras On
    - Active Participation

- C
- Compiling
- Strings
- Variables
- Types
- Loops
- Conditions
- Imprecision
- Overflow
- Extras

Machine Code

- CPUs understand **machine code**. These are the zeroes and ones that tell the machine what to do. Machine code might look like this: 01111111 01000101 01001100 01000110 00000010 00000001 00000001 00000000. Can you read this?

Machine Code

- CPUs understand **machine code**. These are the zeroes and ones that tell the machine what to do. Machine code might look like this: 01111111 01000101 01001100 01000110 00000010 00000001 00000001 00000000. Can you read this?

Assembly Code

- Assembly code includes more english-like syntax. Assembly code is an example of source code, syntax than can be translated to machine code.

- Some sequences of characters in assembly code include these: ***movl***, ***addq***, ***popq***, and ***callq***, which we might be able to assign meaning to. What do you think ***addq*** does?

Machine Code

- CPUs understand **machine code**. These are the zeroes and ones that tell the machine what to do. Machine code might look like this: 01111111 01000101 01001100 01000110 00000010 00000001 00000001 00000000. Can you read this?

Assembly Code

- Assembly code includes more english-like syntax. Assembly code is an example of source code, syntax than can be translated to machine code.

- Some sequences of characters in assembly code include these: ***movl***, ***addq***, ***popq***, and ***callq***, which we might be able to assign meaning to. For example, perhaps ***addq*** means to add or ***callq*** means to call a function. What values are we doing these operations on

- The smallest unit of useful memory is called a **register**. These are the smallest units that we can do some operation on. These registers have names, and we can find them in assembly code as well, such as ***%ecx, %eax, %rsp,*** and ***%rsb***.

Compilers are pieces of software that know both how to understand source code and the patterns of zeroes and ones in machine code and can translate one language to another.

When we code in c, it sufficient to write this in the terminal:

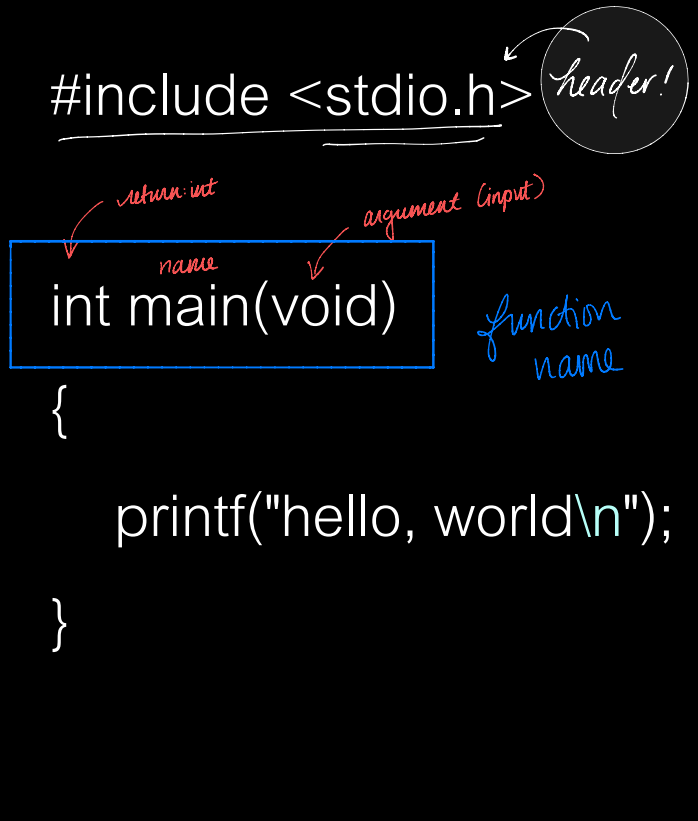
make file to generate the machine readable file

./file to execute the machine readable file

make hello.c → create hello

./ hello → "hello" : LOOKS FOR EXECUTABLE

- C
- Compiling
- Strings
- Variables
- Types
- Loops
- Conditions
- Imprecision
- Overflow
- Extras


#include <stdio.h>
header!
return: int
argument (input)
name
int main(void)
{
printf("hello, world\n");
}
function name
function
get_int( );
library
is in
cs50.h
FUNCTION
return type name (args)

```c
#include <stdio.h>

int main(void)
{
    printf("hello, world\n");
}
```

Escape Sequence!

Types

bool true / false

char 'a' 'b' ... single quotes '.'

double 5.5 64 bit

float 32 bit

int 2, 3, -217... 32bit

long 4, 6, ... 64 bit

string "hi!"

"hi!"

...

'h' 'i' '!'

Functions

Getting inputs (import cs50)	Printing (with printf)
get_char	%c
get_double	%f
get_float	%f
get_int	%i
get_long	%li
get_string	%s

Functions

Getting inputs (import cs50)	Printing (with printf)
get_char	%c
get_double	%f
get_float	%f
get_int	%i
get_long	%li
get_string	%s

Syntax

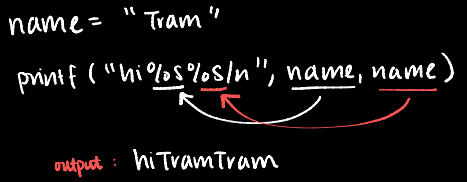
```
#include <cs50.h>
#include <stdio.h>

int main()
{
        int a = get_int("Input an integer: ");
        printf("Value: %i \n", a);
}
```

Functions

Getting inputs (import cs50)	Printing (with printf)
get_char	%c
get_double	%f
get_float	%f
get_int	%i
get_long	%li
get_string	%s

Syntax

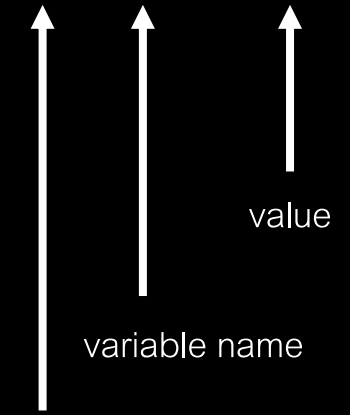
```
#include <cs50.h>
#include <stdio.h>

int main()
{
        int a = get_int("Input an integer: ");
        printf("Value: %i \n", a);
}
```



%i is a placeholder for our variable a

Variables

```
int x = 50;
```




```
char x = 'a';
string greeting = "panda";
```

- C
- Compiling
- Strings
- Variables
- Types
- Loops
- Conditions
- Imprecision
- Overflow
- Extras

Conditions

parentheses

```
if (x > 0)
{
        printf("x is positive\n");
}
else if (x < 0)
{
        printf("x is negative\n");
}
else
{
        printf("x is 0\n");
}
```

Loops

```
for (int i = 0; i < 10; i++)
{
        printf("%i\n", i);
}
```

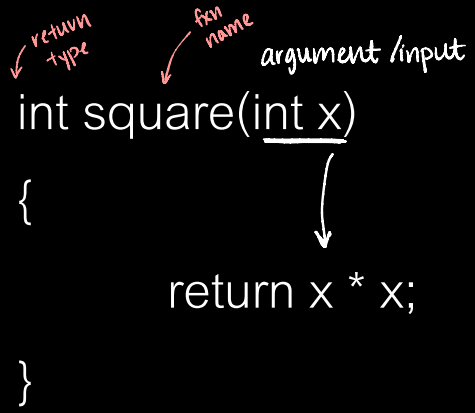
i = 0
print %i, i 0
i++ ⇒ 1
print 1

. . .

i = 9
print 9
i++ ⇒ 10
Is 10 < 10? NO X

```
for ( start ; condition ; change )
{
    STUFF!
}
```

Functions



USING A HELPER FUNCTION

```
int main(void)
{
    int x = 3;
    int y = 4;
    int x2 = square(x);
    int y2 = square(y);
    printf("%d", Math.sqrt(x2+y2));
}
```

HELPER FUNCTIONS

- call them in main on specific inputs to return & use a value you want

- C
- Compiling
- Strings
- Variables
- Types
- Loops
- Conditions
- Imprecision
- Overflow
- Extras

x: 1 y: 10

x / y = 0.10000000000000000555111515123126

Why?

x: 1 y: 10

x / y = 0.10000000000000000555111515123126

We don’t have enough bits to store the entire precise value, so the computer approximates the quotient.

This is called **floating-point imprecision.**

Consider a number that has been allocated three digits. We start by counting.

Suppose we count until 999. We carry, and we get 1000.

However, the computer has only allocated three digits, so our 1000 gets mistaken for 000.

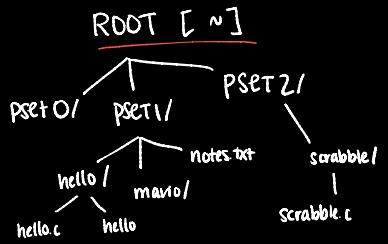
This is an example of integer overflow, where our large number has *wrapped* to a small number.

Y2K
Boeing planes

What's the common problem with floating point imprecision and integer overflow?

MEMORY!

CS50 IDE




```
~
ls    pset0/ pset1/ pset2/
cd    pset1
ls    hello/ mario/ notes.txt
cd    hello
ls    hello.c  hello
./hello
```

Terminal Functions:

ls: listing of all folders in the directory

rm file.txt: remove a file named “file.txt”

cd dir_name: change directories into dir_name

cd ..: changes directories into the parent directory

cd : changes directories into the root directory

```
root :   cd
parent:  cd ..
```

Terminal Functions:

If you try to make a file that is not in your current directory, it will say that the target for your filename is not found. Ensure that you cd into the correct folder.

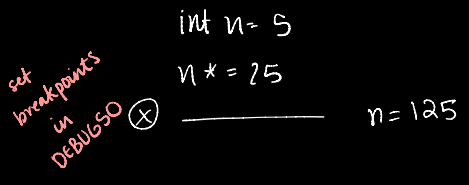
Quick Terminal Exercise!

Given the following list of commands, what might my file structure be? Assume I start from my root directory.

```
ls
cd CS121
cd LectureNotes
cd ..
cd CS50
ls
cd Section
cd Week1
```

help50

debug50



man.cs50.io

cs50.h

stdio.h

Exercises

1. algorithm/pseudocode
2. start implementing

Create two files in your IDE: sum.c and money.c.

1. Write a program sum.c that asks the user to provide three integers as input and prints the sum.

```
#include <cs50.h>
#include <stdio.h>
int main(void) { STUFF }
```

→ get 3 integers from user
  get_int("Give me an int");
→ add 3 integers ⇒ store in new variable
→ printf("Sum %i\n", var)
  %s (%i)

pseudocode

2. Write a program money.c that asks the user for integers representing the number of quarters, dimes, nickels, and pennies as input and prints the amount total.

dollar amount
decimal!

```
#include <cs50.h>
#include <stdio.h>
int main(void) { STUFF }
```

```
int qs = get_int("# of Quarters: ");
(int)  (double) (double)
3 x 0.25 = 0.75
```

pseudocode

3 / 5 = FLOAT 0.6
int div int → int 0

((float) 3) / 5 → float
float / int
(1.0 x 3) / 5

floats & ints

Design!

- clutter :(
- do not be afraid of extra lines/longer code
- come to tutorials & I can look over your code!
- write some notes

Lab!

- n llamas
- every year $n/3$ [integer] llamas
  $-n/4$ [integer] llamas
- START (ask) ≥ 9
- END int $\geq$ START

} while!

- calc # of years
- print # of years

Start 20

end 100

```
start
while (input bad) {
    input = get_int();
}
while (end bad) {___}
int years = _
while (_____) {
    start = _____
    years ++
}
```

CS50 Section 1