Week 01

C → (traditional) language

Is my code correct?

According to our specifications or the like.

. You need to:

- design good code;
- efficient algorithms;
- getting your code nice and clean;
- making sure your code looks pretty, which we'd describe as a matter of style.

~ Correctness, Design and Style ~

So how do you write code?

Integrated Development Environments (IDE's)

or text editors ... ☺

↳ Is a tool that a programmer uses pretty every day to write their code. (VSCode etc)

source code → COMPILER → machine code

convert one language to another.

zeros and ones

GUI → Graphical User Interface

CLI → Command Line Interface ≠

Functions and arguments

printf → is the closest one goes for function for say.

f stands for formatted

string of words → double quotes

printf ("Hello, world!")

arguments → functions → side effects

return values, variables

ask block ⟷ get_string ("What's your name?");

answer = get_string ("What's your name?");

It's not the equal sign in math, is the ASSIGNMENT OPERATOR!

Assign a value means to STORE a value in some variable.

And you read these things RIGHT TO LEFT. the return of the function get_string it's going to get stored over here on the left because of the assignment operator.

But... we need to tell to computer WHAT TYPE OF VALUE it is storing.

string answer = get_string ("What's your name?");

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

int main(void)
{
    string answer = get_string("What's your name? ");
    printf("hello, %s\n", answer);
}
```

%s → formats code which serves as a PLACEHOLDER

formats, it's input for you by using these placeholders for things like strings, represented again by %s.

How many inputs might you infer printf is taking now? (2).

("hello, %s\n", answer);
1º 2º
printf can handle more than one type of variable or value.

the functions we've been talking about take inputs, otherwise now known as arguments, or parameters, pretty much synonymous.

that's just the fancy word for an input to a function.

And some functions have either side effects, like ours — printing something, saying something on the screen, sort of visually or audibly — or they return a value, which is a reusable value, like name or answer, in this case.

arguments → functions → return values

When we get to Python and other languages later in the term, there's actually easier ways to do this.

When clicked ↔
```
int main(void)
{

}
```

./ → this directory (here)

Types

bool, char, double, float, int, long, string

%c, %f, %i, %li, %s

%li → long integer

Variables and Syntaltic Sugar

```
int counter = 0
counter = counter + 1  <->  counter++;
```

counter = counter + 1 → this is not equality!

Integer Overflow →

```
#include
#include

int main(void)
{
    // Prompt user for x
    long x = get_long("x: ");

    // Prompt user for y
    long y = get_long("y: ");

    // Perform addition
    printf("%li\n", x + y);
}
```

Conditions

C uses a single equals sign for assignment from right to left, minor difference between the two worlds.

LEFT ← RIGHT

Constant in a programming language is just an additional hint to the computer that essentially invites you to program more defensively.

If you don't trust yourself necessarily to NOT screw up later, or honestly, in practice, if you know that number should never change, make it constant and never think about it again. kkk

this tells the compiler to make sure that even you later in your code cannot change the number.

And another convention in C and other languages, when you have a constant, it's often common to just capitalize the variable.

example:

```
const int MINE = 2;
```

Kind of like you're yelling, but it really just visually makes it stand out. So it's kind of like a nice rule of thumb that helps you realize, that must be a constant.

Capitalizing does not make it constant the word const does. But the capitalization is just a visual reminder that this is somehow a constant.

Kind of heuristics = strategies / readily accessible.

" Tells me if a number the human types is even or odd? "

we can use a framework for that in place

Now, using syntax we've seen, might I determine if n is EVEN or ODD?

there's this little operator, the remainder operator that will let you do exactly that.

If you divide any number by 2, that mathematical heuristic is going to tell you if it's even or odd based on whether there's a remainder of 0 or 1.

% = remainder operator → It does numerator

divided by denominator and returns NOT the result of that but, rather, the remainder of that.

```
if (n % 2 == 0)
{
    printf("even\n");
}
else
{
    printf("odd\n");
}
```

now this is the equality operator

≠ assignment from right to left.

→ "What is another new piece of syntax, apparently, besides the percent sign?"

You can't use the word OR in C.

AND = &&

|| → multiple characters

| → single character

```
for (int i = 0; i < 3; i++)
{
    printf("meow\n");
}
```

- initialize your variable
- condition that's going to constantly get checked every cycle through this loop
- What you do AFTER each loop, which in this case is going to be count up. (++).

A FOR loop and WHILE loop can both be used to do exactly the SAME thing.

≠

While loop, that I declared my variable OUTSIDE of the loop, that is going to continue to exist elsewhere in my program. So that's one of the minor differences there.

Subtle differences with issues of scope, which we'll discuss before long, where when you create a variable in a FOR loop - notice that it was, again, inside of those parentheses, which technically means it's only going to exist in these 4 lines of code.

```c
#include <stdio.h>

void meow(void);   → PROTOTYPE

int main(void)
{
    for(int i=0; i<3; i++)
    {
        meow();
    }
}

void meow(void)
{
    printf("meow\n");
}
```

A DO but it checks the condition last instead of first. Recall earlier on the slide, we had while, open parenthesis, closed parenthesis. And I kept claiming that we check whether (i) is LESS than - whatever it was, 3 in advance again and again. A do while loop just inverts the logic so that you can actually do something like this =

(example in code)

```c
#include <stdio.h>

void meow(int n);   → PROTOTYPE

int main(void)
{
    meow(3);
}

void meow(int n)
{
    for (int i=0; i<n; i++)
    {
        printf("meow\n");
    }
}
```

WHILE loop is similar to a WHILE loop,

Floating Point Imprecision

Type conversion We can explicitly tell the computer that we actually want to treat this int as though it's a floating-point value.

I can convert (y) to a float by doing this, I can CAST y to a float by literally writing the type float inside of parentheses right before the y.

```
float z = (float) x / (float) y;
```

But it's imprecise ...

So in the world of integers, if you're only using three bits, those three bits eventually overflow when you count past 8.

Because what should be 8 can't fit, so to speak, so it rolls back over to 0.

Integer Overflow

Suppose that we have three bits:

0 0 0

And we counted from 0 to 7, 0, 1, 2, 3, 4, 5, 6, 7. How would we count to 8?

1 1 1

Someone proposed, we need a fourth bit. If you have a 4 bit, if you have access to another "light bulb" or transistor.

If you don't, though, the next number after this is technically 1000.

But if you don't have space for or hardware for that fourth bit, you might or will just be representing the number 0.

So in the world of integers, if you're only using 3 bits, those 3 bits eventually overflow when you count past 8. *

Because what should be 8 could [illegible], so to speak, so it rolls back over to 0.

And so, one of this problem might seem, we humans have done this a couple of times. You might recall knowing about or reading about the Y2K problem, where a lot of people thought the world was going to end.

Because on January 1st of 2000, a lot of computers, presumably, were going to update their clocks from 1999 to the year 2000.

The problem is, though, for decades, for efficiency, we humans were honestly in the habit of not storing years as 4 digits. Because that's just a lot of space to waste, especially since centuries don't happen that often.

So a lot of computer systems, especially early on when hardware was very expensive and memory was very tight, just stored the last 2 digits of any year.

The problem on January 1st of 2000 is that 99 rolls over to 00. But if you don't have room for another digit it's just 00. And if your code assumes a prefix of 19, well, we just went from the year 1999 back to the year 1900. Thankfully, a lot of people wrote a lot of code.

Hardware is so much cheaper these days, computers are so much faster, it's not as big of a deal as it might have been decades ago. But that's indeed the solution.