Week 4 - Monday

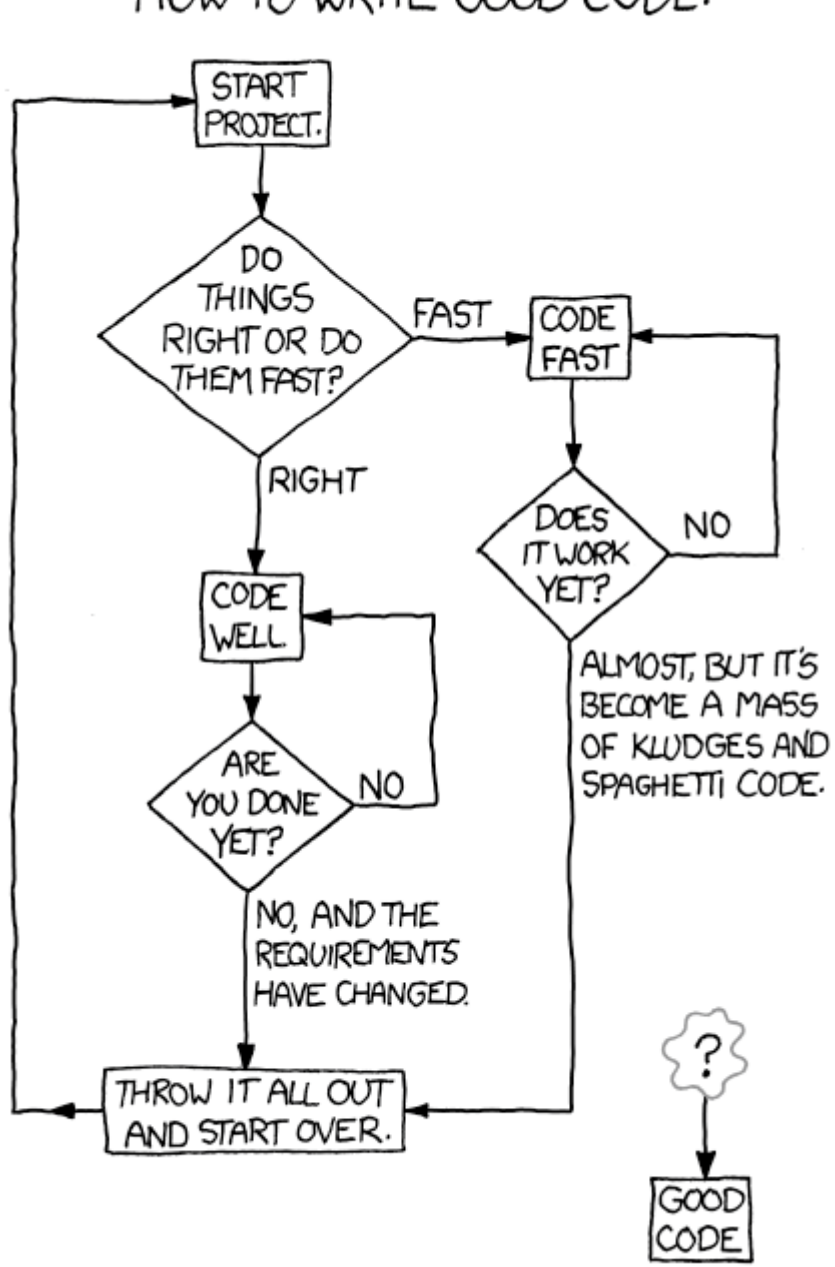
COMP 2400

Last time

- What did we talk about last time?
- Loop errors
- System programming

Questions?

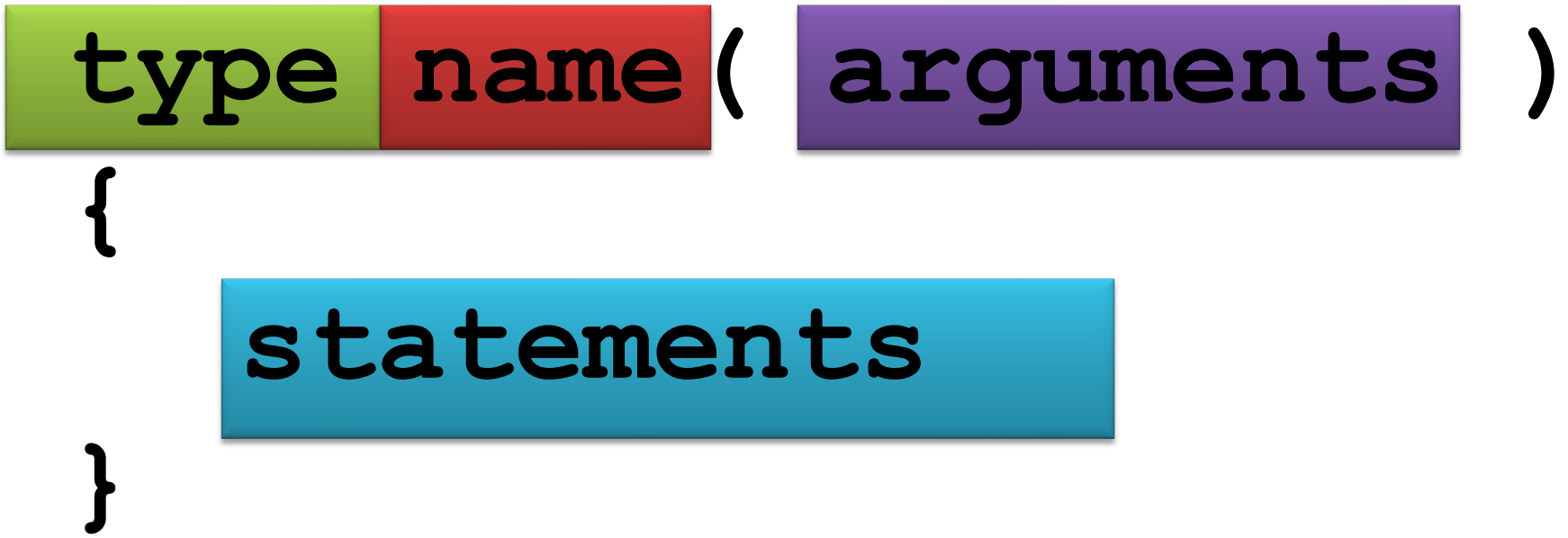
Project 2



From:
https://xkcd.com/844/

Functions

Anatomy of a function definition



Differences from Java methods

- You don't have to specify a return type
  - But you **should**
  - `int` will be assumed if you don't
- If you start calling a function before it has been defined, it will assume it has return type `int` and won't bother checking its parameters

Prototypes

- Because the C language is older, its compiler processes source code in a simpler way
- It does no reasonable typechecking if a function is called before it's defined
- To have appropriate typechecking for functions, create a **prototype** for it
- Prototypes are like declarations for functions
  - They usually come in a block at the top of your source file

Prototype example

- Parameter names in the prototype are optional (and don't have to match)
- Both of the following work:

```
int root(int);
int root(int blah);
```

- You can also declare a prototype locally (inside a function), but there isn't a good reason to do so

```
#include <stdio.h>

int root (int value); // Integer square root

int main()
{
   int output = root(19);
   printf("Value: %d\n", output);
   return 0;
}

int root (int value)
{
   int i = 0;
   while (i*i <= value)
       i++;
   return i - 1;
}
```

Insanity

- If your function takes nothing, you should put **`void`** in the argument list of the prototype
- Otherwise, type checking is turned off for the arguments

```
double stuff(void);

int main()
{
  double output =
  stuff(6.4, "bang"); // Error
  return 0;
}
```

```
double stuff();

int main()
{
  double output =
  stuff(6.4, "bang"); // Legal
  return 0;
}
```

Return values

- C does not force you to return a value in all cases
  - The compiler may warn you, but it isn't an error
- Your function can "fall off the end"
- Sometimes it works, other times you get garbage

```
int sum(int a, int b)
{
      int result = a + b;
      return result;
}
```

```
int sum(int a, int b)
{
      int result = a + b;
}
```

Self defense

- C allows terrible things:
  - Functions without a return type
  - Situations where nothing gets returned
  - Uninitialized variables getting used
- You can get warnings about these things and more by turning on **all** warnings in **`gcc`** with the **`-Wall`** flag:

```
> gcc -Wall program.c -o program
```

- Warnings won't stop your program from compiling, but you can investigate why they're happening

Programming practice

- Let's write a function that:
  - Takes an unsigned integer as a parameter
  - Returns the location of the highest 1 bit in the integer (0-31) or -1 if the integer is 0

Parameter	Return Value
0	-1
2	1
3	1
2000	10
4294967295	31

More programming practice

- Let's update the function from the lab so that it can read a **`double`** value

```
// Original function
int readInt()
{
	int c = 0;
	int i = 0;
	while ((c = getchar()) != EOF && c != '\n')
	{
		if (c >= '0' && c <= '9')
			i = i * 10 + (c - '0');
	}
	return i;
}
```

More programming practice

- Write a function that checks whether an integer is a palindrome
- 141 and 666 are palindromes
- 123 is not a palindrome
- Since you don't know how to convert an int to a string, that approach isn't available
- Hint:
  - Remove each 1's digit from the number and add it to another number, multiplying that number by 10 as you go
  - If and only if the original was a palindrome, the new number will be equal to it

Upcoming

Next time…

- Recursion
- Variable scope

Reminders

- Keep reading K&R chapter 4
- Keep working on Project 2
  - Due by midnight on Friday