CIS241

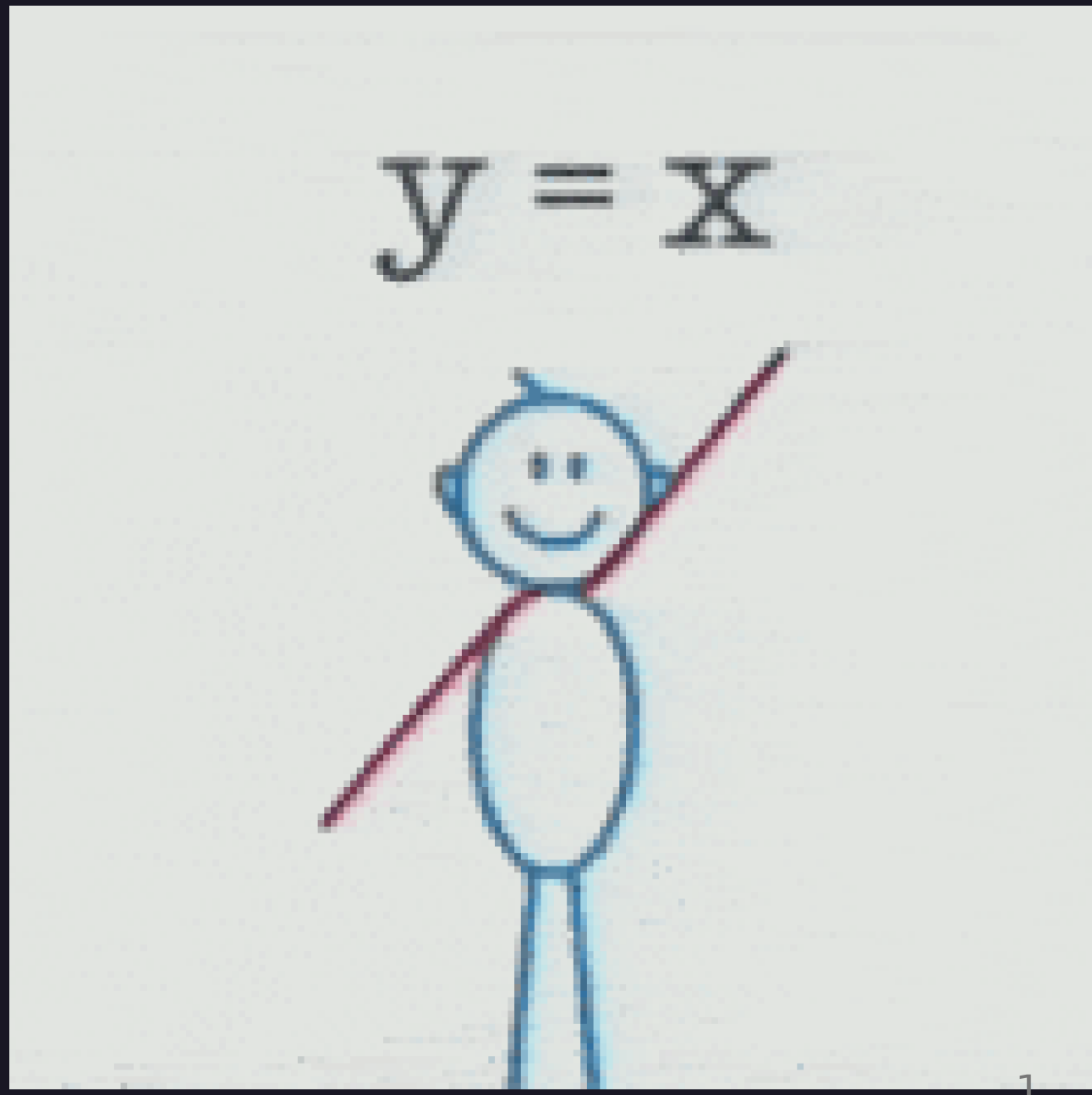
System-Level Programming and Utilities

C Functions

Erik Fredericks, frederer@gvsu.edu

Fall 2025

> **Based on material provided by Erin Carrier, Austin Ferguson, and Katherine Bowers**



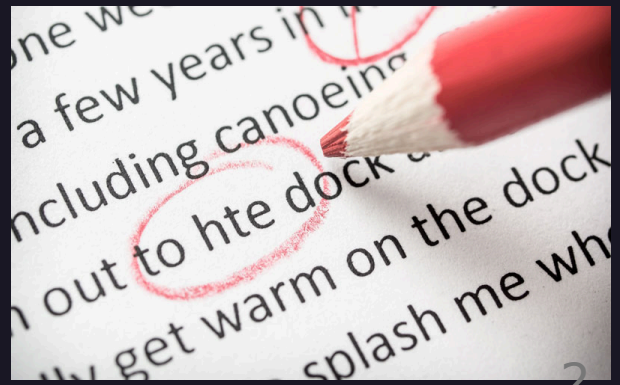
RECAP (from bash + 162/163)

What is a function?

- Blocks of reusable code
- **Typically** should accomplish **one** specific task

Why use them?

- Save time writing code
- Improves readability
- Modularity
- PREVENTS ERRORS



Compare to Python

In Python:

```
def get_smaller_value(a, b):
    if a < b: return a
    return b
```

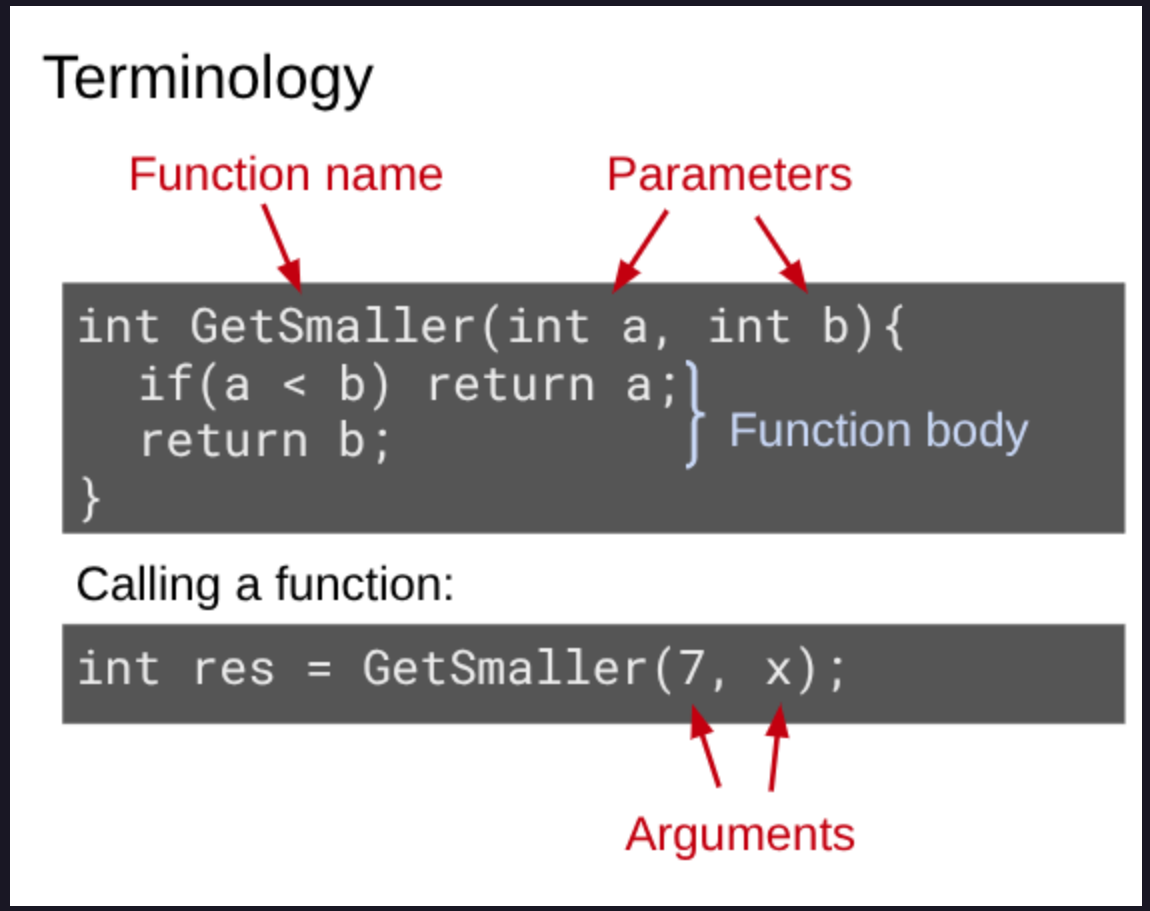
In C:

```
int get_smaller_value(int a, int b)
{
  if (a < b) return a;
  return b;
}
```

Calling it!

```
int result = get_smaller_value(7, x);
```

c/o Ferguson


Terminology
Function name
Parameters
int GetSmaller(int a, int b){
if(a < b) return a;
return b;
}
Function body
Calling a function:
int res = GetSmaller(7, x);
Arguments

If definition is **after** main, you *must* have prototype before main!

And now - the *fun* of C

Function prototype

```
int get_smaller(int a, int b);
```

OR

```
int get_smaller(int, int);
```

Function definition

```
int  get_smaller(int a, int b)
{
     if (a < b) return a;
     return b;
}
```

e.g.

```c
int  get_smaller(int a, int b);  // prototype

int  main(int argc, char* argv[])
{
     if (get_smaller(1, 2) == 1) return 1;
     return 0;
}

int  get_smaller(int a, int b)  // definition
{
     if (a < b) return a;
     return b;
}
```

Return types

ALL functions must declare a return type

- If no return, then must return `void`

```
int  ex1(char ch)
{
      return  (int)ch;
}
```

```
void  ex2(int a, int b)
{
      // do things

      return;    // but not necessary
}
```

Wat does *this* do?

```c
int increment(int x)
{
  x++;
  return x;
}
int main(void)
{
  int i = 0;
  printf("i = %d\n", i);
  int res = increment(i);
  printf("res = %d\n", res);
  printf("i = %d\n", i);
  return;
}
```

Wat does *this* do?

```c
int increment(int x)
{
  x++;
  return x;
}
int main(void)
{
  int i = 0;
  printf("i = %d\n", i);      // 0
  int res = increment(i);
  printf("res = %d\n", res);  // 1
  printf("i = %d\n", i);      // 0
  return;
}
```

Arguments

C is always pass-by-value

- Not pass-by-reference

How can we change arguments?

- Pass a pointer!

And wot does this do?

```
int  increment(int* p)
{
   (*p)++;
   return  *p;
}
int  main(void)
{
   int  i = 0;
   printf(“i = %d\n”, i);
   int  res = Increment(&i);
   printf(“res = %d\n”, res);
   printf(“i = %d\n”, i);
   return  0
}
```

And wot does this do?

```c
int  increment(int* p)
{
   (*p)++;
   return  *p;
}
int  main(void)
{
   int  i = 0;
   printf("i = %d\n", i);       // 0
   int  res = Increment(&i);
   printf("res = %d\n", res);  // 1
   printf("i = %d\n", i);       // 1
   return  0
}
```

And how about passing arrays (ツ)

And can we write a function to print an array nicely?

And how about passing arrays (ಠ◡ಠ)

And can we write a function to print an array nicely?

```
void  PrintArray(int* arr, int len)
{
   printf("[");
   int  i = 0;
   for(i = 0; i < len; i++)
      printf(" %d", arr[i]);
   printf(" ]\n");
}
```

Must also pass length!

Scope

```
void  DoStuff()
{
   int  x = 10;
   printf(“In func: x = %d\n”, x);
   ...
}
int  main()
{
   int  x = 0;
   printf("x = %d\n", x);
   DoStuff();
   printf("x = %d\n", x);
}
```

Scope

```
void DoStuff()
{
   int x = 10;                  // x = 10
   printf(“In func: x = %d\n”, x);
   ...
}
int main()
{
   int x = 0;
   printf("x = %d\n", x); // x = 0
   DoStuff();
   printf("x = %d\n", x); // x = 0
}
```

Scope

Functions have their own scope

- Variables in function don’t exist outside it
- Functions can’t access variables in main

Is this code valid?

```
int* MakeInt()
{
   int x = 10;
   return &x;
}
```

- Nope! `x` falls out of scope when we return
- Memory address is then pointing at nothing :(