# Demystifying C pointers

Manodeep Sinha
Swinburne University

# What is a variable?

Think algebra, “x” is the variable

For humans, a variable can represent ANY type of value

# How about a computer variable?

- For humans, “x” is an unknown quantity, can represent any type, and contain anything

- For a computer, “x” is a string of bits to be interpreted
- Need to know length of each item
- Need to agree on what the representation means

# High-level vs low-level

- Python (high-level): variables can be altered, and can contain anything

- C, C++, Java (low-level): variables have type

- Example types: (unsigned) int, float, double, long, char

- (side-note: you should really use (u)intN_t)

# Low-level variable type: Pointers

- Pointers “point” to memory addresses
- Pointers contain a value that is a memory address
- This is exactly analogous to “human” address

# Why pointers?

- Easy to refer to large chunks of data
- When the total data-size is not known in advance

# Why pointers?

Copying large variables


Sending the address with pointers

# Calling C functions

```
int add_two_func(const int x)
{
    return x + 2;
}

int y = add_two_func(5);
```

# C functions: Getting updated values

- By storing the return value

# Calling C functions

```
int x = 5;

void add_two_func(void)
{
   x = x + 2;
}
```

# C functions: Getting updated values

- By storing the return value
- By modifying global variables

# Calling C functions

```
void add_two_func(int *x)
{
    *x = (*x) + 2;
}

int y = 5;
add_two_func(&y);
```

# C functions: Getting updated values

- By storing the return value
- By modifying global variables
- By updating the underlying memory address

# C functions: Getting updated values

- By storing the return value
- By modifying global variables
- By updating the underlying memory address (need a pointer)

# C pointers

- Pointers contain memory addresses
- You are modifying the memory address
- De-referencing: Looking inside memory address (*ptr)
- Treat as normal variable when updating the value of the pointer (what address to refer to)
- Find address: ptr = &variable

# C pointers

- Function calls -> room analogy
- All function parameters are passed by value
- Including pointers (whose value refers to memory address, rather than being a standard numeric type)
- Modifying the pointer within function only changes address (as in, where you are looking)

# C pointers: How to?

- Need to know: Are you changing the address (where you are looking) or the contents of the address (what is located at that address)
- If you modify address, only changing "where" some underlying variable is located
- If you modify contents at address, updated content visible externally (i.e., after function exits)