CSE 333 – SECTION 4

OOC, pass-by-value,

*pointers vs. references,

- C: Everything is pass-by-value
- Simple example of params

```
void swap(int a, int b) {
  int tmp = a;
  a = b;
  b = tmp;
}

int a = 1;
int b = 2;
swap(a, b);
```

- Simple example of return vals

```
typedef struct {
  double x;
  double y;
} Point;

Point create(double a, double b) {
  Point p1 = {.x = a, .y = b}
  return p1;
}
```

Remember not to pass back pointers into this stack frame!!!

```
Point* create(double a, double b) {
  Point p1 = {.x = a, .y = b}
  return &p1; // NOT GOOD
}
```

Preview: references in C++

This or that?

- Consider the following code:

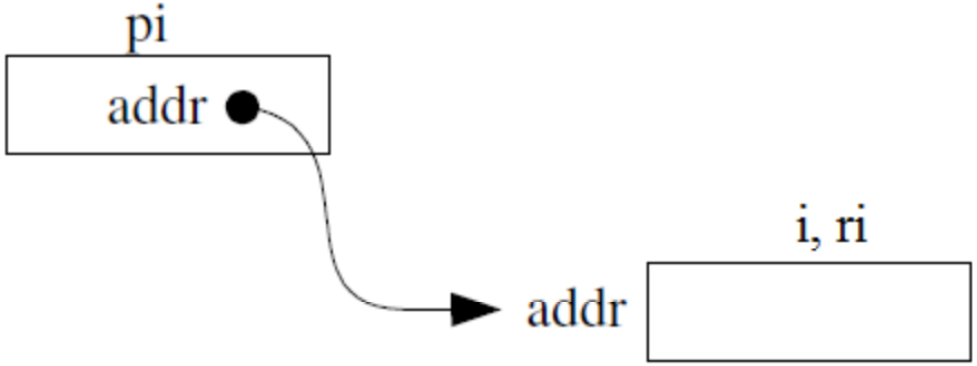
Pointers:

```
int i;
int *pi = &i;
```

References:

```
int i;
int &ri = i;
```

In both cases,



The difference lies in how they are used in expressions:

```
*pi = 4;
```

```
ri = 4;
```

References Example

```
// Part 1
int i = 0, j = 4;
int *pi = &i;

// Part 2
int &ri = i;

// Part 3
*pi = 3;

// Part 4
 ri = j;
```

Pointers and References

- Once a reference is created, it cannot be later made to reference another object.
  - Compare to pointers, which are often reassigned.
- References can’t be initialized to *null*, whereas pointers can.
- References can never be uninitialized. It is also impossible to reinitialize a reference.

When to use?

- **Pointers**: may point to many different objects during its lifetime. Pointer arithmetic (++ or --) enables moving from one address to another. (Arrays, for e.g.)
- **References**: can refer to only one object during its lifetime.
- **Style Guide Tip:**
  - use const reference parameters to pass input
  - use pointers to pass output parameters
  - input parameters first, then output parameters last