

# CSE 333 – SECTION 4

---

C++ References, const and classes

# Reminders

- **HW2 due next Thursday, October 25th**
- **Exercise due tomorrow**
- **Makefile needed for exercise due Monday**

# This or that?

- Consider the following code:

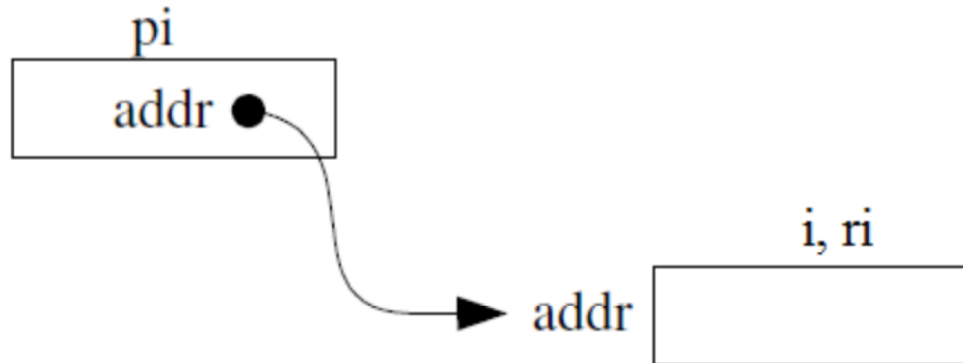
## Pointers:

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

## References:

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

In both cases,



References are aliases – the same memory location with more than one name

```
*pi = 4;
```

```
ri = 4;
```

# 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.
- Demo: `experiments.cc`

# 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

# C++ const declaration

- As a declaration specifier, `const` is a type specifier that makes objects unmodifiable.

```
const int m = 255;
```

- Reference to constant integer:

```
int n = 100;
```

```
const int &ri = n; // ri becomes read only
```

- Uses of `const` for magic numbers

```
const int BUFFER_SIZE = 100;
```

```
char input[BUFFER_SIZE]
```

- Demo: `const.cc`

# C++ Classes

/\* Note: This code is unfinished! Beware! \*/

```
class Point {
```

```
    public:
```

```
        Point(const int x, const int y); // constructor
```

```
        int get_x() const { return x_; } // inline member function
```

```
        int get_y() const { return y_; } // inline member function
```

```
        double distance(const Point &p) const; // member function
```

```
        void setLocation(const int x, const int y); //member function
```

```
    private:
```

```
        int x_; // data member
```

```
        int y_; // data member
```

```
}; // class Point
```

# Const Practice

Refer to the following poorly-written class declaration. (10 min)

```
class MultChoice {
public:
    MultChoice(int q, char resp) : q_(q), resp_(resp) { } // 2-arg ctor
    int get_q() const { return q_; }
    char get_resp() { return resp_; }
    bool Compare(MultChoice &mc) const; // do these MultChoice's match?

private:
    int q_; // question number
    char resp_; // response: 'A','B','C','D', or 'E'
}; // class MultChoice
```

a) Indicate (Y/N) which *lines* of the snippets of code below (if any) would cause compiler errors:

```
const MultChoice m1(1, 'A');
MultChoice m2(2, 'B');
cout << m1.get_resp();
cout << m2.get_q();
```

```
const MultChoice m1(1, 'A');
MultChoice m2(2, 'B');
m1.Compare(m2);
m2.Compare(m1);
```



# Section Exercise

- Define a class `Rectangle` whose instance variables are a pair of `Point` objects (upper left, lower right).
- Include at least one constructor. Make sure you get `const` right in the right places.
- Methods:
  - **`getul()`**, **`getlr()`** - returns upper and lower points. (upper-left, lower-right)
  - **`intersect(Rectangle &r)`** – returns a `Rectangle` representing the overlap.
  - **`area()`** - returns the `Rectangle`'s area.
  - **`contains(Point &p)`** - returns true or false depending on whether point `p` is inside the rectangle.
- The C++ Primer text and [cplusplus.com](http://cplusplus.com) contain good reference material.