Name: ____

Today's Date: ____

Today's Goals

- Describe the difference between a read-only iterator and a read-write iterator
- Reason about both types of iterators in ChunkyString
- Know when and how to use some convenient modern C++ language features

Today's Question(s)

What is the difference between these?

```
int * const x;
const int * y;
```

Lingering Questions

Read-only iterators

Goal: allow

```
MySet<char> values;
...
for (MySet<char>::iterator i = values.begin(); i != values
    cout << *i << endl;
}
```

but disallow:

```
for (MySet<char>::iterator i = values.begin(); i != values
   *i  = 'X';
}
```

Why do the following not work?

- Declaring begin as a `const` member function:
- Declaring values as `const MySet<char>`:
- Declaring iter as `const MySet<char>::iterator`:

std::conditional

```
bool use_int = true;

// creates a struct with a type called mytype that is eith
typedef std::conditional<true, int, double>::mytype Type1;
typedef std::conditional<false, int, double>::mytype Type2
typedef std::conditional<use_int, int, double>::mytype Typ

Type1 x = 3;
Type2 y = 4;
Type3 z = 5;
```

Deciding which begin to use

```
template <bool is_const>
class Iterator {
public:
    using value_type = char;
    using reference = typename std::conditional<is_const,
                                                  const valu
                                                  value_type
    using pointer = typename std::conditional<is_const,
                                                const value_
                                                value_type*>
}
```

C++11, 14, 17 . . .

You've earned the right to use some new things. . .

auto: Type inference

First try: print a list of ints:

```
void printAnyList(const list<int>& elements) {
  for (list<int>::const_iterator i = elements.begin(); i !
    cout << *i << endl;
  }
}
```

How does `auto` work?

Range fors

auto + range for