Learning Targets

1. I can read complicated C++ types
2. I can explain why iterators are useful in C++
3. I can describe what functionality a class must support to have iterators
4. I can write code that uses iterators to loop through the collections

Review

1. You can read any declarations using the "inside out" rule. To do this, start"on the inside" at the variable name, go right, then left, and "spiral" outwards as needed.

   1. *int x;*
   2. *Cow barn[10];*
   3. *Cow* v;*
   4. *const int * w1;*
   5. *int * const w2;*

   6. int *z[5];

   7. int (*y)[4];

   8. const Cow (* const (*q)[4])[6]

   Answers;

   1. *x is an int*
   2. *barn is an array of 10 cows*
   3. *v is a pointer to a cow*
   4. *w1 is a pointer to a constant int*
   5. *w2 is a const pointer to an int*
   6. *z is an array of 5 pointers of ints*
   7. *y is a pointer to a size 4 array of ints*
   8. *q is a pointer to an array of size 4 const pointers to an array of size 6 const Cows*

2. Which Member Functions should be const?

```
class StringStack {
public:
void push(const std::string& pushee);
```

```
bool empty();
std::string& top(); // Access top element
void pop(); // Discard top element
private: ...
};
```

3. Implementation 1. Dynamic Array
4. Implementation 2. Linked List

Option 1. Strengths and Weaknesses?

```
class StringStack {
public:
    void push(const std::string& pushee);
    ...etc...

    void print() const;
    bool hasString(const std::string& searches) const;
    bool hasDuplicates() const;
    ...etc...
};
```

Weaknesses: * We need to run through all the values for print, finding a string, and having duplicates. * The user is forced to go implement the StringStack functions, or have some external program to do the same thing.

Option 2. Strengths and Weaknesses?

```
class StringStack {
public:
    void push(const std::string& pushee);
    ...etc...

    std::string& operator[](size_t index);
    const std::string& operator[](size_t index);
    ...etc...
};
```

Weaknesses:

Arrays and Pointer Arithmetic

The following are quivalent (if it's an array of strings)

```
for(size_t i = 0; i < DATA_LEN; ++i) {
    std::cout << data[i];
}

for (std::string* p = data; p != data + DATA_LEN; ++p) {
    std::cout << *p;
}
```

In this case the != on the second one emphasizes the "while you haven't gone too far out of the data structure".

Using Iterators Effectively

Every collection type that we have in C++ all have iterators. 1. Set my iterator to the beginning of the data structure 2. As long as I haven't fallen off the data structure, keep going.

```
// Print the integers in the vector<int> v
for (vector<int>::iterator i = v.begin(); i != v.end(); ++1)
    cout << *i << endl;

// Print characters of string s
for (string::iterator i = s.begin(); i != s.end(); ++i)
    cout << *i << endl;

// Print strings of set<string> t
for (set<string>::iterator i = t.begin(); i != t.end(); ++i)
    cout << *i << endl;

// Print booleans in list<bool> l
for (list<bool>::iterator i = l.begin(); i != l.end(); ++i)
    cout << *i << endl;
```

What does an iterator need? * .begins() * .end() * .operator++() * .operator*() If I'm pointing there, how to I handle the necessary data* .operator==() * .operator!=() * Constructor (Copy Assignment Constructor at least) * Destructor (?)

Using iterators, how could we: 1. Print all the elements in a StringStack? 2. Check if a StringStack contains "swordfish"? 3. Check if a StringStack is empty (without calling .empty())?

```
// Printing all the elements in a StringStack
StringStack ss = ...;
for (string::iterator i= ss.begin(); i != ss.end(); ++i)
    cout << *i;
```

```
// Check if StringStack contains "swordfish"
bool found = false;
for (string::iterator i= ss.begin(); i != ss.end(); ++1) {
    if (*i == "swordfish") {
        found = true;
    }
}

// Check if a StringStack is empty
ss.begin() == ss.end();
```