Lecture 6a: Interfaces and Iterators

CS 70: Data Structures and Program Development

Tuesday, February 25, 2020

Learning Goals

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 collections.

The "Inside Out" (or "Right-to-Left") Rule

Start "on the inside" at the variable name, and spiral outwards

- `int x`
- `Cow barn[10]`
- `Cow* v`
- `const int * w1`
- `int * const w2`

The "Inside Out" (or "Right-to-Left") Rule

Start "on the inside" at the variable name, and spiral outwards

- `int *z[5]`
- `int (*y)[4]`
- `const Cow (* const (*q)[4])[6]`

StringStack Interface

Example `const` member function:

In stringstack.hpp

```
class StringStack {
public:
   size_t size() const; // can't change data members!
    ...
};
```

In stringstack.cpp

```
size_t StringStack::size() const {
  return size;
}
```


StringStack Interface
Which Member Functions should be `const`?

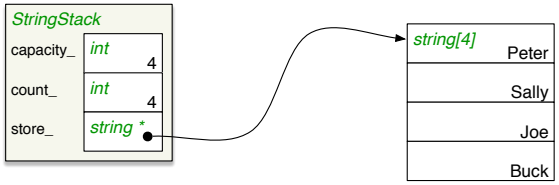
```
class StringStack {
public:
  void push(const std::string& pushee);
  bool empty();
  std::string& top();  // access top element
  void pop();          // discard top element
  ...
private:
  ...
};
```


StringStack Interface
Improved version

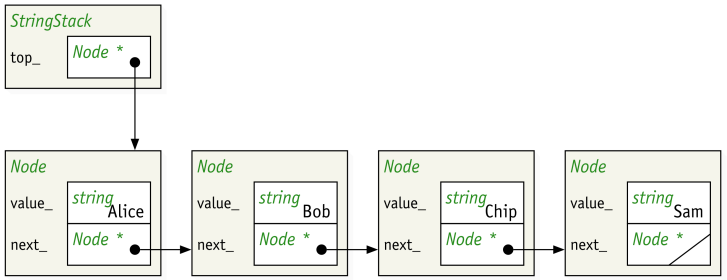
```
class StringStack {
public:
  void push(const std::string& pushee);
  bool empty() const;
  std::string& top();
  const std::string& top() const;
  void pop();
  ...

private:
  ...
};
```

Implementation Idea 1: Dynamic Array



Implementation Idea 2: Linked Lst



Extending the interface

Suppose we want to access all elements of a `StringStack`?

- Print the current stack elements for debugging
- Check whether the stack contains a particular string
- Check whether the stack contains two copies of the same string
- …

Interface Idea 1. Strengths and weaknesses?

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

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

Interface Idea 2. Strengths and weaknesses?

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

  std::string& operator[](size_t index);
  const std::string& operator[](size_t index) const;

private:
  ...
};
```


Iterators

- Iterator: an object for iterating through each element in some collection or container

- C++ syntax example for StringStack:

```
// Print strings in StringStack ss
for (StringStack::iterator i = ss.begin();
      i != ss.end(); ++i)
{
    cout << *i << endl;
}
```

Using Iterators Effectively ("Classic" C++)

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


Recall: Arrays and pointer arithmetic

For a primitive array `data`, the following are equivalent:

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

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


Iterators *similar* to pointers

```
std::string data[DATA_LEN]={…};

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

Iterators are NOT (necessarily) pointers

- The iterator syntax is similar, and pointers are a nice metaphor to reason about the syntax of iterators.
- The iterator implementation can be wildly different under the hood.

What operations do we need to implement?

```
// Print strings in StringStack ss
for (StringStack::iterator i = ss.begin();
    i != ss.end(); ++i)
{
  cout << *i << endl;
}
```

Supporting Stack Iterators

```
class StringStack {
 public:
   ...push, pop, top, empty...

   class iterator {
    public:
      iterator(const iterator&) = default;
      bool operator!=(const iterator& rhs) const;
      iterator&    operator++();
      std::string& operator*() const;
    private:
     ...
   };

   iterator begin();
   iterator end();

 private:
    ...
};
```


Using iterators, how could we

1. Check if a `StringStack` contains `"swordfish"`?
2. Check if a `StringStack` is empty (without calling `.empty()`)?