**Today**

- Abstract classes in C++

**Assignments**

- HW2 out (due thurs)
- Quiz 2 on Thurs
Overloaded operators as non-member function (friends)

A **friend** function of a class is:

- a non-member function
- that can access protected & private class members

Example:

```cpp
class Rectangle
{
  public:
  ...

  friend Rectangle operator+(const Rectangle& lhs, const Rectangle& rhs);

};

void main()
{
  Rectangle a(10, 20);
  Rectangle b(20, 30);
  Rectangle c = a + b;
}
```

Help provide greater control of the left-hand versus right-hand operand types
Stream insertion and extraction operators

Stream insertion (<<) and extraction (>>) must be non-member functions!

- e.g., for: `cout << r1;` or `cin >> r1;`

Example

```cpp
class Rectangle
{
    public:
        ...

    friend std::ostream& operator<<(std::ostream& out, const Rectangle& rhs);

    friend std::istream& operator>>(std::istream& in, Rectangle& rhs);
};
```

// implementation ...

```cpp
std::ostream& operator<<(std::ostream& out, const Rectangle& rhs)
{
    out << rhs.length << " " << rhs.width;
    return out;
}
```

```cpp
std::istream& operator>>(std::istream& in, Rectangle& rhs)
{
    in >> rhs.length;
    in >> rhs.width;
    return in;
}
```

Q: Explain the signatures (const, references, return types)

Q: Why do we return the streams?
C++ Vector Basics

The vector class encapsulates an implementation of a “resizable array”

Some of the basic member functions:

- void push_back(const value_type& val); add val to end
- size_t size() const; get length (int n = int(v.size()))
- bool empty() const; true if size is zero
- void clear(); removes all elements
- const_ref at(size_type n) const; get element at pos n
- const_ref operator[](size_type n) const; eg: x = v[2];

The vector class is a “parametric” (or “generic”) type ...

```cpp
// create a vector of ints
vector<int> xs;
xs.push_back(10);
xs.push_back(20);
assert(xs[0] == 10);

// create a vector of strings
vector<string> ys;
y.push_back("foo");
y.push_back("bar");
y.push_back("baz");
```

Note that need to use std::vector as opposed to just vector
Removing an element with the `erase` function

```cpp
// assuming v.size() >= 1
v.erase(v.begin()); // removes first element

// assuming v.size() >= 2
v.erase(v.begin() + 1); // removes second element

// for some int variable i
v.erase(v.begin() + i); // removes i-th element
```

Need to watch out for vector length when removing elems!

Inserting elements within a vector using the `insert` function

```cpp
vector<int> v; // creates {}

v.insert(v.begin(), 13); // result is {13}

v.insert(v.begin() + 1, 15); // result is {13, 15}

v.insert(v.begin() + 0, 11); // result is {11, 13, 15}

v.insert(v.begin() + 1, 12); // result is {11, 12, 13, 15}

v.insert(v.begin() + 3, 14); // result is {11, 12, 13, 14, 15}
```

Again, be careful of vector length

- if 2 element vector (e.g., `{13, 14}`)
- inserting at index 3 or higher, will result in a segmentation fault
Managing collections of key-value pairs

- A “key” represents an identifier
- And each key has a corresponding value
- Together they form a “key-value” pair
- Examples:
  - Stock symbol (key) to stock price (value)
  - An SSN (key) to a name (value)
  - Dictionary word (key) to the word definition (value)
  - A state (key) to its area, capital, total population (“record” value)

We’ll create different implementations of key-value pair collections

- A key-value collection is an “abstract data type” (ADT)
- Similar to a queue or stack
- And underlying implementations via common “data structures”
- For HW3 we’ll use vector (resizable arrays) as the data structure

The main functions our key-value collection ADT will support:

- add ... a specific k-v pair
- remove ... given a key, remove the k-v pair
- find value ... given a key, find corresponding value
- find range ... find a range of values
- sort keys ... return a sorted list of keys
The `collection.h` file

```cpp
#ifndef COLLECTION_H
#define COLLECTION_H

#include <vector>

template<typename K, typename V>
class Collection
{
public:

   // add a new key-value pair into the collection
   virtual void add(const K& a_key, const V& a_val) = 0;

   // remove a key-value pair from the collection
   virtual void remove(const K& a_key) = 0;

   // find and return the value associated with the key
   virtual bool find(const K& search_key, V& the_val) const = 0;

   // find and return the values with keys >= k1 and <= k2
   virtual void find(const K& k1, const K& k2, std::vector<V>& vals) const = 0;

   // return all of the keys in the collection
   virtual void keys(std::vector<K>& all_keys) const = 0;

   // return all of the keys in ascending (sorted) order
   virtual void sort(std::vector<K>& all_keys_sorted) const = 0;

   // return the number of key-value pairs in the collection
   virtual int size() const = 0;

};
#endif
```
Breaking down the Collection class: Abstract Classes

An **abstract class** has one or more “abstract” member functions

- which means the functions are not implemented as part of the class
- and the compiler won’t yell at you about it

If a class is abstract, it cannot be instantiated ...

- Only non-abstract derived classes can be instantiated
- For example, assuming `MyConcreteClass` is a subclass of `MyAbstractClass`

```cpp
MyAbstractClass c1;    // compile error!
MyConcreteClass c2;    // this is fine
MyAbstractClass& c3 = c2;    // this is also ok
c3.f();        // can call functions just fine
...

void g(MyAbstractClass& c)
{ ... does something to change c ... }

g(c2);            // this is also ok
```

Abstract classes in C++ are defined by having one or more **pure virtual functions**

- Adding “= 0” at the end of the member function prototype
- For example:

```cpp
// remove a key-value pair from the collection
virtual void remove(const K& a_key) = 0;
```