Generics and Operator Overloading

generics
- allow us to implement abstract collections of “things”
- maybe we want a List of int or float or char or string
- we wouldn't want to rewrite the List for each type
- instead we can make it generic and able to contain any/all of these!
  - even works for more “complicated” objects (like lists of “people”)

e.g. of non-generic (can only contain int)

```
#include <iostream>
using namespace std;

const int MAX_SIZE = 100;

class List
{
	public:
		friend ostream& operator<<(ostream& out, List& l)
		{
			for (int i=0; i<l.curr; i++)
				out << l.list[i] << " ";

			return out;
		}

		List(int n)
		{
			size = n;
			curr = 0;
		}

		void Add(int n)
		{
			list[curr++] = n;
		}

		int itemAt(int n)
		{
			return list[n];
		}

	private:
		int list[MAX_SIZE];
		int size;
		int curr;
};

int main()
{
	List l(5);

	for (int i=0; i<5; i++)
		l.Add(i+1);
	cout << l << endl;
}
```

e.g. of generic (can contain any type)

```
#include <iostream>
using namespace std;

template <class list_type, int max_size>
```

```
class List
{
	public:
		friend ostream& operator<<(ostream& out, List& l)
		{
			for (int i=0; i<l.curr; i++)
				out << l.list[i] << " ";

			return out;
		}

		List()
		{
			size = max_size;
			curr = 0;
		}

		void Add(int n)
		{
			list[curr++] = n;
		}

		list_type itemAt(int n)
		{
			return list[n];
		}

	private:
		list_type list[MAX_SIZE];
		int size;
		int curr;
};

int main()
{
	List<int,5> l1;
	List<char,26> l2;

	for (int i=0; i<5; i++)
		l1.Add(i*i);
	for (int i=0; i<26; i++)
		l2.Add(i+97);
	cout << "l1=" << l1 << endl;
	cout << "l2=" << l2 << endl;
}
```

operator overloading

	suppose we want to do something like the following

```
List<int,10> l1;
List<int,10> l2;
List<int,10> l3;

for (int i=0; i<5; i++)
{
	l1.Add(i+1);
	l2.Add(i*i);
}
l3 = l1 + l2;// how so that l3=1 2 3 4 5 0 1 4 9 16???
```

	we could manually add items to l3 by iterating through l1 and l2
	or we could overload the + operator and define what this operation "means" for a List

```
List& operator+(const List& l)
{
	List t(*this);
```

```
        t.curr = t.end;
        for (int i=0; i<l.end; i++)
                t.Add(l.list[i]);

        return t;
}
```

note *this
- sometimes we wish to refer to the entire class (not just member variables)
- to do this, we use *this (a pointer to "this" instance of the class)
- we'll cover pointers later

notice that this is essentially "syntactic sugar"
- in the background, we are essentially adding items manually

there are many operator overloads
- binary operators
  - +, -, *, /, %, &, |, <<, >>
- unary operators
  - +, -, !, ~, ++, --
- relational operators
  - ==, !=, <, >, <= , >=
- and more

- operator= is slightly different
  - we treat it just like the copy constructor
  - `void operator=(const List& l);`

you may have noticed the "friend" function
- a non-member function that has access to all members (public/private) of the class
- i.e. we wish to borrow this functionality (defined elsewhere) and use it in our class

HANDOUT classes3