Templates and Iterators

Template

Templates are a **blueprint** of a function that let you use the same function for a variety of types:

```
//genaric min(a,b)
template<typename T> //---> template parameter
T min (T a, T b) {
    return (a < b) ? a:b;
}
usage: min<T>(a, b)
```

Any time template instantiation occurs, the compiler will check that all operations used on the templatised type are supported by that type.

```
template<typename T>
void myfunc (T x) {
    x.push_back(8);//it is ok since vector support push_back();
    cout<<x;//operator "<<" is not supported by vector

}

int main() {
    vector<int> v;
    myfunc(v);
}
```

Iterator Types

There are **5** different types of iterators

1. `Input: single-pass input, can only be dereferenced on the right side of expression`
2. `Output: single-pass outpu, can only be dereferenced on the left side of expression`
3. `Forward: can make multiple passes, can read from and write to`
4. `Bidirectional can move forward and backward`
5. `Random access can move to arbitrary position`

All iterators share the following common traits

- Can be advanced using `++`
- Can be compared using `==` and `!=`