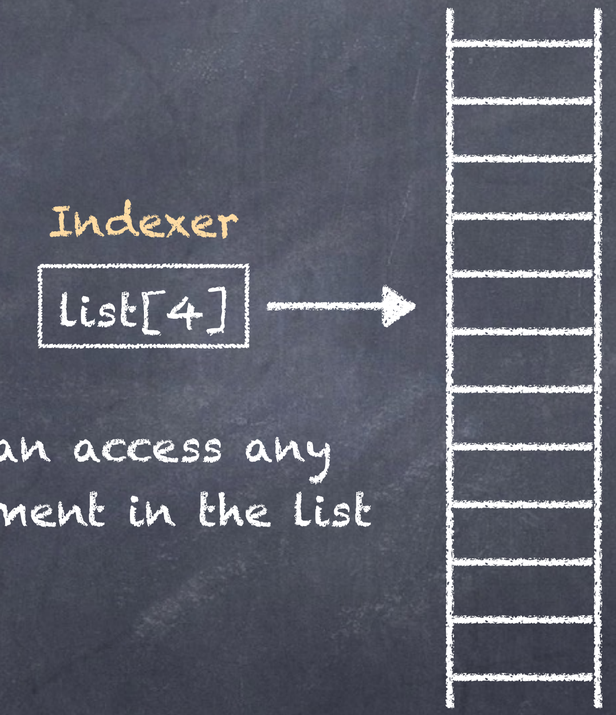
For versus Foreach

For

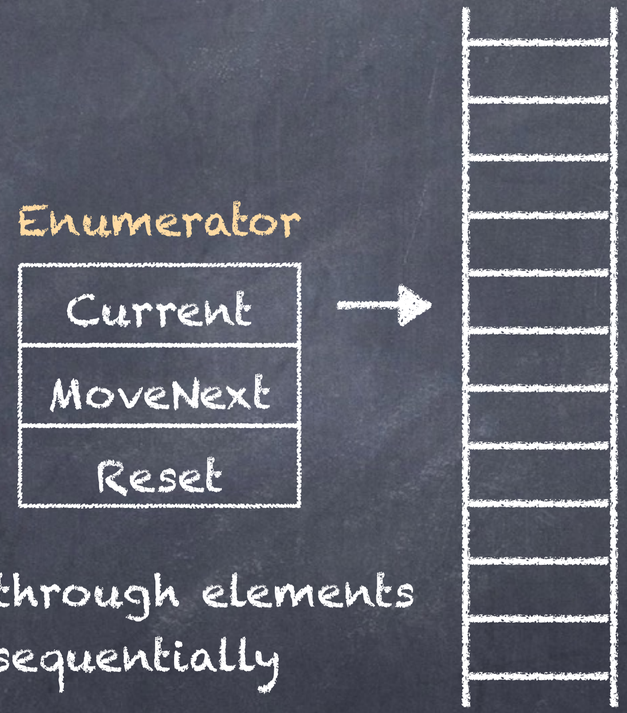
```
for (int i=0; i < list.Length; i++) {
    ...
    int result = list[i];
    ...
}
```



Can access any element in the list

Foreach

```
foreach (int i in list) {
    ...
    int result = i;
    ...
}
```



Steps through elements sequentially

Pros and cons

For
- Pro: fastest but requires an indexer
- Con: indexer needs all values loaded in memory

Foreach
- Pro: works on any collection
- Pro: loads values on demand
- Con: slower because it requires an enumerator

Takeaway

Non-generic enumerators return the current value as an object. Do not use them for value types to avoid boxing and unboxing.

Always use generic enumerators if possible

How to use for and foreach

- Array: do not refactor code, not worth it.
- List<>: refactor foreach to for to get 1.6x improvement
- ArrayList: refactor foreach to for to get 2.8x improvement, but consider using List<> instead

For value type collections, enumerate over IEnumerable<T>, not IEnumerable, to avoid boxing and unboxing