Iterators

Le Van Huynh

October 16, 2023

Contents

1 Question

The following program has at least four iterator-related problems. How many can you find?

```
std::vector<Date> e;
copy(istream_iterator<Date>(cin),
     istream_iterator<Date>(),
     back_inserter(e));
vector<Date>::iterator first = find(e.begin(), e.end(),
                                    "01/01/95");
vector<Date>::iterator last = find(e.begin(), e.end(),
                                   "12/31/95");
*last = "12/30/95";
copy(first,
     last,
     ostream_iterator<Date>(cout, "\n"));
e.insert(--e.end(), TodaysDate());
copy(first,
```

```
    last,
    ostream_iterator<Date>(cout, "\n"));
```

2 Answer

The first 4 statements are OK:

```
std::vector<Date> e;
copy(istream_iterator<Date>(cin),
    istream_iterator<Date>(),
    back_inserter(e));
vector<Date>::iterator first = find(e.begin(), e.end(),
                                    "01/01/95");
vector<Date>::iterator last = find(e.begin(), e.end(),
                                   "12/31/95");
```

The first error is that `last` may be `e.end()`, which cannot be deferenced:

```
*last = "12/30/95";
```

The second error is that `[first, last)` may not be a valid range. The variable names are misleading, but there is no guarantee that `first` comes before `last`.

```
copy(first,
    last,
    ostream_iterator<Date>(cout, "\n"));
```

The third potential error involves the expression `--e.end()`. `e.end()` is most likely of type `Date*` (pointer type). In this case, it's a *prvalue* and cannot be modified.

```
e.insert(--e.end(), TodaysDate());
```

We can instead write:

```
e.insert(e.end() - 1, TodaysDate());
```

However, this still does not work if `e` is empty.

The last issue is related to iterator invalidation. Because the vector has changed (with `insert`), `first` and `last` may have been invalidated. Therefore, the final statement

```
copy(first, last,
     ostream_iterator<Date>(cout, "\n"));
```

may no longer work.

3 Summary

Four things to remember when using iterators:

1. Valid values: Can the iterator be dereferenced?
2. Valid lifetimes: Is the iterator valid or has been invalidated?
3. Valid ranges: Are `first` and `last` pointing to the same container, and does `first` come before `last`?
4. Illegal built-in manipulation: e.g., is the code trying to modify a temporary of built-in type? A good thing is that compilers often catch this error.