Doc No: N2388=07-0248
Date: 2007-09-07
Author: Pablo Halpern
Bloomberg, L.P.

phalpern@halpernwightsoftware.com

Pointer Arithmetic for `shared_ptr`

Contents

Motivation

Shared pointers are a type of smart pointer intended to replace raw pointers in situations where the lifetime of the pointed-to object cannot easily be determined in advance. The use of operator* and operator-> for dereferencing give them a pointer-like syntax, but unlike real pointers, `shared_ptr` cannot participate in pointer arithmetic. At the July 2007 meeting in Toronto, aliasing was added to smart pointers, allowing two smart pointers to share ownership of one object but point to a different object, typically a sub-part of the owned object. Aliasing allows us to create two shared pointers that share ownership of an array but point to two different elements within that array. For example:

```
const int SZ = 100;
shared_ptr<char> dataBegin(new char[SZ], ArrayDeleter<char>());
shared_ptr<char> dataEnd(dataBegin, dataBegin.ptr() + SZ);
```

Wouldn't it be nice if the same operation could be accomplished with `operator+()`?

```
shared_ptr<char> dataEnd = dataBegin + SZ;
```

Adding pointer arithmetic operations to `shared_ptr` would make them more like real pointers.

Document Conventions

All section names and numbers are relative to the August 2007 working draft, N2369.

> Existing and proposed working paper text is indented and shown in dark blue. Small edits to the working paper are shown with ~~green strikeouts for deleted text~~ and green underlining for inserted text within the indented blue original text. Large proposed insertions into the working paper are shown in the same dark blue indented format.

Comments and rationale mixed in with the proposed wording appears as shaded text.

Requests for LWG opinions and guidance appears with yellow shading.

Summary

New Member Operators

I propose to add the following members to `shared_ptr<T>`:

```
shared_ptr<T>& operator++();
shared_ptr<T>  operator++(int);
shared_ptr<T>& operator--();
shared_ptr<T>  operator--(int);
shared_ptr<T>& operator+=(ptrdiff_t);
shared_ptr<T>& operator-=(ptrdiff_t);
T& operator[](ptrdiff_t index);
```

New Free Operators

I propose to add the following free operators:

```
shared_ptr<T>   operator+(const shared_ptr<T>&, ptrdiff_t);
shared_ptr<T>&& operator+(shared_ptr<T>&&,      ptrdiff_t);
shared_ptr<T>   operator-(const shared_ptr<T>&, ptrdiff_t);
shared_ptr<T>&& operator-(shared_ptr<T>&&,      ptrdiff_t);
ptrdiff_t operator-(const shared_ptr<T>&,
                    const shared_ptr<T>&);
```

Changes to Shared Pointer Comparison

It is a basic axiom of pointer arithmetic that if q = p + 1, then p < q. The semantics of `operator<()` in the current working draft, however, would have p and q be *equivalent* because they share ownership. I propose that we redefine `operator<()` such that p < q iff p.get() < q.get(). This definition of `operator<()` would be consistent with the definition of `operator==()`.

In order to continue to support ownership comparisons, I also propose adding a new member function:

```
template <typename U>
int compare_ownership_group(const shared_ptr<U>& other);
```

This function would return 0 if `*this` and `other` *share ownership* of an object (i.e., they belong to the same ownership group) and –1 or 1 otherwise, such that `x.compare_ownership_group(y) < 0` is a strict weak ordering on all shared pointers. The combination of `operator<` and `compare_ownership_group` allows a user to compare shared pointers based on value, ownership, value then ownership, or ownership then value. Note that the other comparison operators are defined in terms of `operator<`, so no change is necessary there.

Because the language does not guarantee that comparing pointers yields a strict week ordering unless the pointers point into the same array, `std::less`, `std::greater`, `std::greater`, etc., are specialized for pointers such that it produces a total ordering over the entire domain. A similar specialization would be needed for `shared_ptr`:

```
template <class T>
struct less<shared_ptr<T> > :
    binary_function<shared_ptr<T>, shared_ptr<T>, bool>
{
    bool operator()(const shared_ptr<T>& x,
                    const shared_ptr<T>& y) const
        { return std::less<T*>()(x.get(), y.get()); }
};
```

Limitations

This proposal is limited to those aspects of `shared_ptr` that cannot be implemented outside the library itself. In particular, it does not propose methods for creating shared pointers to arrays i.e. array-based deleters, and array-based `make_shared` and `allocate_shared` factory functions.

Proposed Wording

Proposed wording is not yet available due to time constraints. If there is interest in Kona, proposed wording should be simple to add, possibly during the Kona meeting itself.