Sparsity Parsery

Or

Compile-time trickery for dealing
with sparse key sets

Mike Seymour

github.com/mikeseymour/wocca

Problem

- Receiving key-value tags with integer keys
- Keys cover a large range of values
- Messages might contain many tags
- We're only interested in a small, fixed subset
- Examples:
  - Music tagging (like ID3v2)
  - Financial protocols (like FIX)

Solution

- Parse the interesting tags into a small array:

```
parser<title,album,artist> p(reader);
```

- Read them by key, calculating the array index at compile time:

```
out << “Title: “ << p.at<title>();
out << “Album: “ << p.at<album>();

p.at<bpm>(); // ERROR! unspecified tag
```

Basic types

- Values: perhaps a view over received data

```
using view = std::string_view;
```

- Tags: nullable pairs (philosophically awkward)

```
struct tag {
    explicit operator bool() const;
    int key;
    view value;
};
```

- Reader: functor returning sequential tags

The Parser

- Contains an array of values
- Initialised from a reader

```
while (tag t = reader()) {
    int i = index(t.key);
    if (i >= 0) values[i] = t.value;
}
```

- Read by key

```
static_assert(index(key) >= 0);
return values[index(key)];
```

Gory details: Key sets

- A compile-time set of integer keys:

```
template <int... Keys> struct keyset {
  static constexpr int keys[] {Keys...};
};
```

- Operations, including

```
// gory details omitted
template <class Keys> using sort = keyset<???>;
```

Gory details: Finding the index

- Binary search in a sorted key-set's array

```
using sorted = sort<keyset<Keys...>>;
int first = 0, last = std::size(sorted::keys);
while (first != last) {
    int mid = first + (last-first)/2;
    if (sorted::keys[mid] == key)
        return mid;
    if (sorted::keys[mid] < key)
        first = mid+1;
    else
        last = mid;
}
return -1;
```

Gory details: Sorting the keys

```
template <int Key, class Keys> struct prepend_;
template <int Key, class Keys> using prepend = typename prepend_<Key, Keys>::result;

template <int Key, class Keys> struct prepend_ {using result = keyset<Key>;};
template <int Key, int... Keys> struct prepend_<Key, keyset<Keys...>>
    {using result = keyset<Key, Keys...>;};

template <int Key, class Keys> struct remove_;
template <int Key, class Keys> using remove = typename remove_<Key, Keys>::result;

template <int Key, class Keys> struct remove_ {using result = keyset<>;};
template <int Key, int... Tail> struct remove_<Key, keyset<Key, Tail...>>
    {using result = keyset<Tail...>;};
template <int Key, int Head, int... Tail> struct remove_<Key, keyset<Head, Tail...>>
    {using result = prepend<Head, remove<Key, keyset<Tail...>>>;};

template <class Keys> struct min_;
template <int Single> struct min_<keyset<Single>> {static constexpr int result = Single;};
template <int Head, int... Tail> struct min_<keyset<Head, Tail...>> {
    static constexpr int tail = min_<keyset<Tail...>>::result;
    static constexpr int result = Head < tail ? Head : tail;
};
template <class Keys> static constexpr int min = min_<Keys>::result;

template <class Keys> struct sort_;
template <class Keys> using sort = typename sort_<Keys>::result;

template <> struct sort_<keyset<>> {using result = keyset<>;};
template <int... Keys> struct sort_<keyset<Keys...>> {
    static constexpr int first = min<keyset<Keys...>>;
    using result = prepend<first, sort<remove<first, keyset<Keys...>>>>;
};
```