Proposal of std::upto, std::indices and std::enumerate

Introduction

We propose the addition of three tiny helper functions: std::upto, std::indices and std::enumerate:

std::upto(n) returns a range of 0,1,2,...,n-1
std::indices(c) returns std::upto(std::size(c))
std::enumerate(c) returns the zipped range of std::indices(c) with c

for (auto i : std::upto(5))
    std::cout << i; // prints 0 1 2 3 4

std::string fruits[] = {"apple", "orange", "banana");

for (auto i : std::indices(fruits))
    std::cout << i; // prints 0 1 2

for (const auto& [i, fruit] : std::enumerate(fruits))
    std::cout << i << fruit; // prints 0 apple 1 orange 2 banana

Motivation

Zero-based indexing is an extremely common use case in C++. Based on analysis of ACTCD19, 30% of C/C++ files contain for statements, and of those 50% are zero-index-based for loops of the form:

for (T i = 0; i < N; i++) ...
It makes sense to provide a simple short-hand. For not the least of reasons that people often mess up the type of the loop variable:

```cpp
for (int i = 0; i < vec.size(); i++) // oops, .size() is decltype(vec)::size_type not int
```

Also the end bound is evaluated once.

**Design**

We considered, in addition, adding more complicated forms like `std::upto(a,b)` and `std::downfrom(n)` and steps - but this created complications. Inclusive or exclusive bounds? What about if the upper bound is inclusive and is at max of the type?

We have a `std::ranges` library for doing all sorts of fancy things. `std::upto`, `std::indices` and `std::enumerate` are just about addressing tersely the simple single-step zero-indexing use case that come up the most often.

**Should these be in the ranges subnamespace?**

No. They should be very terse, as they come up so often.

**Specification**

In header `<ranges>`...

```cpp
namespace std {

  template<typename Int>
  constexpr auto upto(Int&& i) noexcept { return std::ranges::iota_view{Int(),std::forward<Int>(i)}; }

  template<typename Container>
  constexpr auto indices(const Container& c) { return std::upto(std::size(c)); }

  template<typename Container>
  constexpr auto sindices(const Container& c) { return std::upto(std::ssize(c)); }

  template<typename Container>
  auto enumerate(Container&& c) { return std::ranges::zip_view(std::indices(c),c); }

  template<typename Container>
  auto senumerate(Container&& c) { return std::ranges::zip_view(std::sindices(c),c); }

} // namespace std
```

Outstanding Issues

- Uses zip_view from p1035r4