CS 3460

Introduction to Concepts

Concepts

- A way to constrain or specify requirements for template parameters
- A concept is a predicate
  - Evaluated at compile time, no run-time cost
  - Considered part of the template interface
- Nice side effect: Better compiler error messages
- Added with C++ 20

Concepts - Defining

```
template<typename T>
concept Incrementable = requires(T x)
{
    x++;
    ++x;
};
```

- Two new keywords: `concept` and `requires`
- Inside the `concept`, define requirements
  - In simple terms, code that must compile with the type T

Concepts – Another Example

- A concept requirement can be more than a simple code expression: `x.size()` example below

```
template<typename T>
concept Array = requires(T x)
{
    x.operator[](0);
    { x.size() } ->std::convertible_to<std::size_t>;
};

template<typename T>
concept BeginEnd = requires(T x)
{
    x.begin();
    x.end();
};
```

Concepts – Another Example

- A concept requirement can be more than a simple code expression: `x.size()` example below
- Can combine concepts into a single concept: `ArrayRanges`

```
template<typename T>
concept Array = requires(T x)
{
    x.operator[](0);
    { x.size() } ->std::convertible_to<std::size_t>;
};

template<typename T>
concept BeginEnd = requires(T x)
{
    x.begin();
    x.end();
};

template<typename T>
concept ArrayRanges = Array<T> && BeginEnd<T>;
```

Concepts – Using Concepts

- Write template code as normal, but also *require* concepts

```
template <typename T>
void report(T data) requires Array<T> && BeginEnd<T>
{
    std::cout << "--- Using BeginEnd Concept ---" << std::endl;
    for (auto i = data.begin(); i != data.end(); i++)
    {
        std::cout << *i << std::endl;
    }

    std::cout << "--- Using Array Concept ---" << std::endl;
    for (decltype(data.size()) i = 0; i < data.size(); i++)
    {
        std::cout << data[i] << std::endl;
    }
}
```

```
template <typename T>
requires Array<T> && BeginEnd<T>
void report(T data)
{
    std::cout << "--- Using BeginEnd Concept ---" << std::endl;
    for (auto i = data.begin(); i != data.end(); i++)
    {
        std::cout << *i << std::endl;
    }

    std::cout << "--- Using Array Concept ---" << std::endl;
    for (decltype(data.size()) i = 0; i < data.size(); i++)
    {
        std::cout << data[i] << std::endl;
    }
}
```

Concepts – Standard Library

- Many concepts already defined, including…
  - `derived_from`
  - `signed_integral`, `unsigned_integral`
  - `integral`, `floating_point`, …
  - `constructable`, `destructible`, `move_constructible`, …
  - `movable`, `copyable`
  - `sortable`
  - and so much more...
- Reference: https://en.cppreference.com/w/cpp/concepts