Variants of variadic AND

Dr. Roland Bock

http://ppro.com
rbock at eudoxos dot de

https://github.com/rbock/sqlpp11
https://github.com/rbock/kiss-templates

CppCon 2016, 2016-09-20

The task

```
template <bool... Args>
struct all_t
{
    static constexpr bool value = (true if all Args are true);
};
```

Recursion with template structs

```
template<bool... Args>
struct all_t;
```

Recursion with template structs

```
template<bool... Args>
struct all_t;

template<>
struct all_t<>
{
  static constexpr bool value = true;
};
```

Recursion with template structs

```
template<bool... Args>
struct all_t;

template<>
struct all_t<>
{
  static constexpr bool value = true;
};

template<bool Arg, bool... Rest>
struct all_t<Arg, Rest...>
{
  static constexpr bool value = Arg && all_t<Rest...>::value;
};
```

Recursion with constexpr functions

```
constexpr auto all() -> bool
{
  return true;
}
```

Recursion with constexpr functions

```
constexpr auto all() -> bool
{
  return true;
}

template<typename Arg, typename... Rest>
constexpr auto all(Arg arg, Rest... rest) -> bool
{
  return arg && all(rest...);
};
```

Recursion with constexpr functions

```
constexpr auto all() -> bool
{
  return true;
}

template<typename Arg, typename... Rest>
constexpr auto all(Arg arg, Rest... rest) -> bool
{
  return arg && all(rest...);
};

template<bool... Args>
using all_t = std::integral_constant<bool, all(Args...)>;
```

Using noexept

```
template <bool Arg>
struct nx_helper
{
  constexpr explicit nx_helper() noexcept(Arg) {}
};
```

Using noexept

```
template <bool Arg>
struct nx_helper
{
  constexpr explicit nx_helper() noexcept(Arg) {}
};

template <typename ...T>
void nx_join(T const&...) noexcept;
```

Using noexept

```
template <bool Arg>
struct nx_helper
{
  constexpr explicit nx_helper() noexcept(Arg) {}
};

template <typename ...T>
void nx_join(T const&...) noexcept;

template <bool ...Args>
struct all_t: std::integral_constant<
      bool,
      noexcept(nx_join(nx_helper<Args>{}...))>
{};
```

Using std::is_same

```
template<bool...>
struct all_helper
{};
```

Using std::is_same

```
template<bool...>
struct all_helper
{};

template<bool... Args>
using all_t = std::is_same<all_helper<true, Args...>,
                           all_helper<Args..., true>>;
```

Using fold expressions (C++17)

```
template<bool... Args>
using all_t = std::integral_constant<bool, (true && ... && Args)>;
```

What's the point?

What's the point?

Readability and performance.

Compile time [s] for different numbers of bools*

	1
Recursive struct	0.00
Recursive function	0.00
noexcept	0.00
std::is_same	0.00
fold expression	0.00

Using `clang++-trunk -std=c++1z -ftemplate-depth=16384`

*Disclaimer: These numbers are not too accurate. They also depend on compiler version flags, Hardware, OS, etc.

Compile time [s] for different numbers of bools*

	1	256
Recursive struct	0.00	0.06
Recursive function	0.00	0.08
noexcept	0.00	0.01
std::is_same	0.00	0.01
fold expression	0.00	0.01

Using `clang++-trunk -std=c++1z -ftemplate-depth=16384`

*Disclaimer: These numbers are not too accurate. They also depend on compiler version flags, Hardware, OS, etc.

Compile time [s] for different numbers of bools*

	1	256	1024
Recursive struct	0.00	0.06	0.80
Recursive function	0.00	0.08	xxx
noexcept	0.00	0.01	0.01
std::is_same	0.00	0.01	0.01
fold expression	0.00	0.01	0.01

Using `clang++-trunk -std=c++1z -ftemplate-depth=16384`

*Disclaimer: These numbers are not too accurate. They also depend on compiler version flags, Hardware, OS, etc.

Compile time [s] for different numbers of bools*

	1	256	1024	16384
Recursive struct	0.00	0.06	0.80	xxx
Recursive function	0.00	0.08	xxx	xxx
noexcept	0.00	0.01	0.01	0.11
std::is_same	0.00	0.01	0.01	0.03
fold expression	0.00	0.01	0.01	0.71

Using `clang++-trunk -std=c++1z -ftemplate-depth=16384`

*Disclaimer: These numbers are not too accurate. They also depend on compiler version flags, Hardware, OS, etc.

Compile time [s] for different numbers of bools*

	1	256	1024	16384	65536
Recursive struct	0.00	0.06	0.80	xxx	xxx
Recursive function	0.00	0.08	xxx	xxx	xxx
noexcept	0.00	0.01	0.01	0.11	xxx
std::is_same	0.00	0.01	0.01	0.03	0.12
fold expression	0.00	0.01	0.01	0.71	25.50

Using `clang++-trunk -std=c++1z -ftemplate-depth=16384`

*Disclaimer: These numbers are not too accurate. They also depend on compiler version flags, Hardware, OS, etc.

Morale

Avoid recursion.

Morale

Avoid recursion.

Strive for readability and performance.

Thank you very much!