C++ now

Fun with Flags

Type-safe Bitwise Operations

Tobias Loew

2024

Outline

- Using bits for Boolean options
- Builtin bitwise operators
- The Complement-Problem
- Boost.Flags Library
- Other Approaches

Using bits for Boolean options

Easy to define, e.g. as macros, integer constants or enumerators

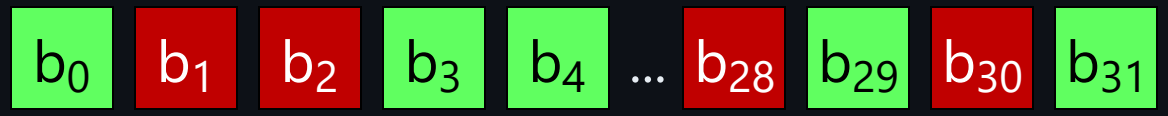
- ```
  #define OPTION_ONE   0x1
  #define OPTION_TWO   0x2
  #define OPTION_THREE 0x4
  ```

- ```
  static constexpr int option_one   = 0x1;
  static constexpr int option_two   = 0x2;
  static constexpr int option_three = 0x4;
  ```

- ```
  enum options_enum {
      option_one   = 0x1,
      option_two   = 0x2,
      option_three = 0x4
  };
  ```
  ```

Compact representation in memory

- `int` can hold up to 16 / 32 different Boolean options



- Simple & efficient bulk transfer in interfaces
- Compressed version of a `std::array<bool, 32>`

Enforces the usage of names / prevent bool parameters

- Declaration

```
void foo(bool use_opt1, bool use_opt2, bool use_opt3);

void foo(options_enum options);
```

- Call site

```
foo(true, false, true);

foo(option_one | option_three);
```

Builtin support through bitwise operators

- `bitwise AND: int operator&(int, int)  int& operator&=(int&, int)`
- `bitwise OR:  int operator|(int, int)  int& operator|=(int&, int)`
- `bitwise XOR: int operator^(int, int)  int& operator^=(int&, int)`
- `bitwise Negation: int operator~(int)`

Everything good!

Everything good! - Really?

Bitwise operators in action …

- integer types
  - all operators supported
  - no type-safety at all
- unscoped enums
  - all except assignment operators supported by implicit integer promotion
  - no type-safety for bitwise ops, `static_cast` to enum required
- scoped enums
  - no operator supported
  - `static_cast` to underlying and back to enum required

A short fairy-tale

- The Fun in this talk

Once upon a time...

Don't lose your head!

https://de.freepik.com/

Don't lose your head!

https://disney.fandom.com/

Don't lose your head!

https://upload.wikimedia.org/wikipedia/commons/c/c4/Iron_Rule.jpg

The Dayly Queen

The Dayly Queen

Proclamation: Lorem ipsum dolor sit amet, *The Queen*!

Proclamation: At vero eos et accusam, *The Queen*!

Proclamation: Stet clita kasd gubergren, *The Queen*!

Proclamation: Duis autem vel eum, *The Queen*!

Proclamation: Nam liber tempor cum soluta, *The Queen*!

Proclamation: Ut wisi enim ad minim veniam, *The Queen*!

The Dayly Queen

Proclamation:

All proclamations shall be encoded in C++,

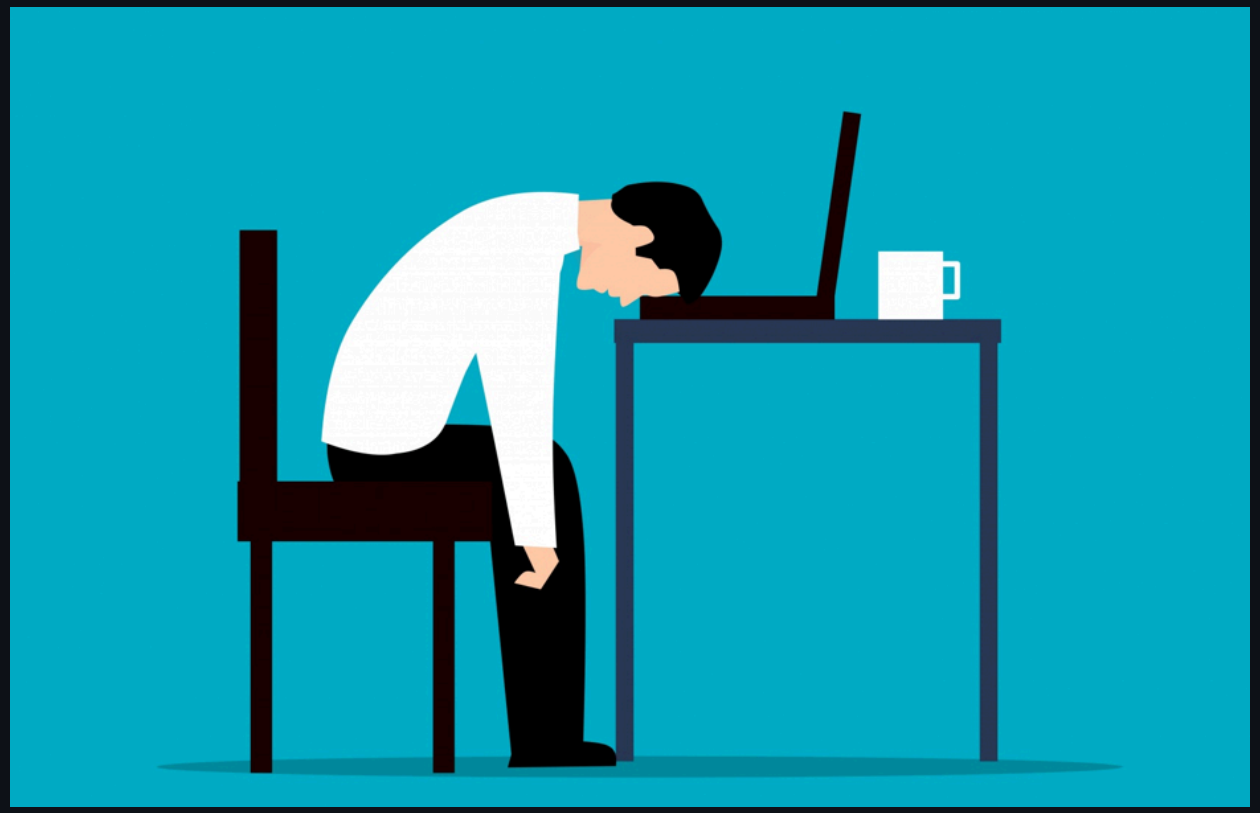
The Queen!

```
enum font_styles : unsigned int {
    bold = 1,
    italic = 2,
    underline = 4
};
```

```
enum font_styles : unsigned int {
    bold = 1,
    italic = 2,
    underline = 4
};

// special style for the Queen's name
static constexpr auto queen_of_hearts_name = italic |
                                             underline;
```

```
queen_of_hearts_name queen_of_hearts_name queen_of_hearts_name queen_of_hearts_name
queen_of_hearts_name queen_of_hearts_name queen_of_hearts_name queen_of_hearts_name
```




```
queen_of_hearts_name queen_of_hearts_name queen_of_hearts_name queen_of_hearts_name
queen_of_hearts_name queen_of_hearts_name queen_of_hearts_name queen_of_hearts_name
```

```
queen_of_hearts_name queen_of_hearts_name queen_of_hearts_name queen_of_hearts_name
queen_of_hearts_name queen_of_hearts_name queen_of_hearts_name queen_of_hearts_name
```


```
queen_of_hearts_name queen_of_hearts_name queen_of_hearts_name queen_of_hearts_name
queen_of_hearts_name queen_of_hearts_name queen_of_hearts_name queen_of_hearts_name
```

```
queen_of_hearts_name
```

```
-->
```

```
~bold
```

Don't lose your head!

Microsoft Copilot

Don't lose your head!

https://disney.fandom.com/

```
enum font_styles : unsigned int {
    bold = 1,
    italic = 2,
    underline = 4,
    strikeout = 8   // for Alice's name
};
```

The Dayly Queen

Proclamation:

The Dayly Queen

Proclamation:

All persons whose name is striken out are enemies of the kingdom,

The Dayly Queen

Proclamation:

All persons whose name is striken out are enemies of the kingdom,

~~*The Queen*~~!

Don't lose your head!

Microsoft Copilot

https://disney.fandom.com/

The End

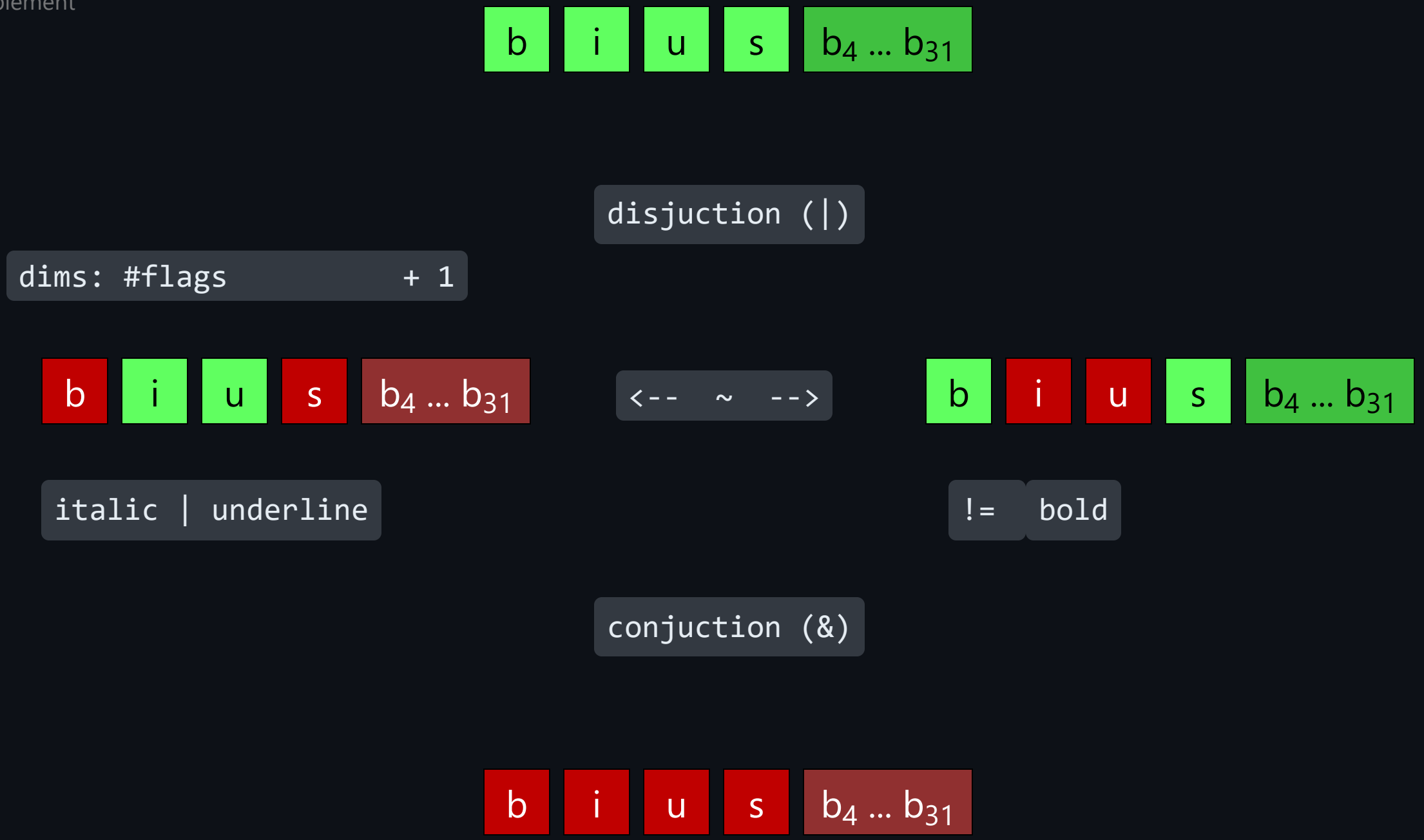
Moral:

```
queen_of_hearts_name != ~bold
```

- Syntactical

```
std::to_underlying(queen_of_hearts_name) != std::to_underlying(~bold)
```

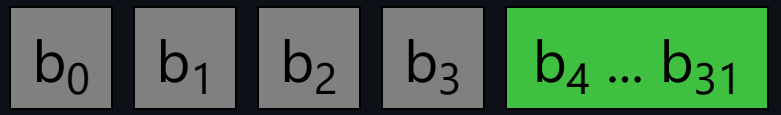
- Semantical
  - `~bold` is not a set of font-modifications
  - Excludes boldness from a given set of flags


b i u s b4 … b31
disjuction (|)
dims: #flags + 1
b i u s b4 … b31
<-- ~ -->
b i u s b4 … b31
italic | underline
!= bold
conjuction (&)
b i u s b4 … b31


```
#flags: 4
#bit underlying type U: 32
```

- Underlying type U with `~, &, |`: *(bitcount of U)*-dim Boolean algebra
- `flags` with `&, |` form a substructure F with dimension `#flags`
  - in general *is not closed under bitwise negation* `~`.
- Closure $\overline{F}$ wrt. $\sim$
  - dimension `#flags + 1`, Boolean subalgebra of U

- Elements of $\overline{F} \setminus F$: negative flag-masks



  - put into different Type

- For given enumeration `E`
  - Type for flags: `E`
  - Type for negative flag-masks: `complement<E>`
- Meta-level: type-set `{E, complement<E>}` with `~`, `&` and `|` is a Boolean algebra

```
enum flags {
    flag_0 = 0x1,
    flag_1 = 0x2
};
inline flags operator~ (flags x) {
    return static_cast<flags>(~static_cast<int>(x));
}
// other operators ...
void foo() {
    flags f = flag_0 | flag_1;
    flags mask = ~flag_1;
    f = f & mask;
}
```

```cpp
enum flags {
    flag_0 = 0x1,
    flag_1 = 0x2
};
inline flags operator~ (flags x) {
    return static_cast<flags>(~static_cast<int>(x));
}
// other operators ...
void foo() {
    flags f = flag_0 | flag_1;
    flags mask = ~flag_1;
    f = f & mask;
}
```

- unscoped enum w/o explicit underlying type: two types inferred:
  - `underlying_type`
  - hypothetical integer value type with minimal width such that all enumerators can be represented -> defining the valid value-range
- `complement<flags>` uses `underlying_type_t<flags>` for value

Design Rationale of Boost.Flags

- type-safety
  - enforce compatible types for binary operators
  - `complement` for `operator~`
- non-intrusive
- zero-overhead in optimized builds
- everything is `constexpr`
- operators are found by ADL
- at least C++11, use newer features if available

Boost.Flags: life example...

https://godbolt.org/z/5ffq6eGaW

Orders on flags-enumerations

- induced by `<` on underlying integer type
  - total order
  - depends on enumerator-values
  - Syntax: used for sorting, sorted containers
- induced by entailment of flags
  - `a < b` <=> "flags in a" $\subset$ "flags in b"
  - partial order
  - independent of enumerator-values
  - Semantics

- sorted containers / algorithms require total order
  - `BOOST_FLAGS_SPECIALIZE_STD_LESS(E)` for sorted containers / std::algorithms
  - NOT working with `std::ranges`: calling `operator <`
- enums abstract away underlying values
  - program logic should not depend on underlying values
- `BOOST_FLAGS_REL_OPS_DELETE(E)` deletes all relational operators
- constexpr objects `total_order`, `partial_order`
- functions for partial-order comparison: `subset`, `subseteq`, `intersect`, `disjoint`

Boost.Flags: additional features

- `make_null`, `make_if`
- `modify`, `modify_inplace`
- `add_if`, `add_if_inplace`
- `remove_if`, `remove_if_inplace`
- `get_underlying`, `from_underlying`
- Flags-generator:
  - `flags_from_to(first, last)` iterates over all flags from `[first, ..., last]`
  - `flags_to(last)` iterates over all flags from `[1, ..., last]`

Boost.Flags: applying to existing code-bases

- change macro / integer constants to enums
- opt-in by with `BOOST_FLAGS_ENABLE(E)`
- use scoped enums, `using enum E;`
- change interfaces from integer-types to enums
- minor adjustments:
  - ```
    if (flags & LBS_NOTIFY) {...}   // scoped enum -> bool
    ```
  - ```
    (cond ? LBS_NOTIFY : 0)
    ```

    ```
    boost::flags::make_if(LBS_NOTIFY, cond)
    ```
- also works with enums in classes: `BOOST_FLAGS_ENABLE_LOCAL(E)`

Boost.Flags: in real code

- EBSILON® Professional: 122 uses of `BOOST_FLAGS_ENABLE...`
- Clang source code: enabled 16 enums - no errors detected
- Github-repo: tests run on a large matrix:
  - linux, windows, mac
  - clang (3.9 - 17), gcc (4.8-13), msvc (v140 - v143)
  - C++11 - C++23

Some other available Flags libraries

	QFlags	foonathan	BOOST_BITMASK	.net

Some other available Flags libraries

	QFlags	foonathan	BOOST_BITMASK	.net
non-intrusive	-	-	+	+

Some other available Flags libraries

	QFlags	foonathan	BOOST_BITMASK	.net
non-intrusive	-	-	+	+
zero-overhead	-	+	+	+

Some other available Flags libraries

	QFlags	foonathan	BOOST_BITMASK	.net
non-intrusive	-	-	+	+
zero-overhead	-	+	+	+
`constexpr`	-	+/-	+/-	N/A

Some other available Flags libraries

	QFlags	foonathan	BOOST_BITMASK	.net
non-intrusive	-	-	+	+
zero-overhead	-	+	+	+
`constexpr`	-	+/-	+/-	N/A
type-safe bin-ops	-	+	-	+

Some other available Flags libraries

	QFlags	foonathan	BOOST_BITMASK	.net
non-intrusive	-	-	+	+
zero-overhead	-	+	+	+
`constexpr`	-	+/-	+/-	N/A
type-safe bin-ops	-	+	-	+
large underlying_t	-	+	-	+

Some other available Flags libraries

	QFlags	foonathan	BOOST_BITMASK	.net
non-intrusive	-	-	+	+
zero-overhead	-	+	+	+
`constexpr`	-	+/-	+/-	N/A
type-safe bin-ops	-	+	-	+
large underlying_t	-	+	-	+
`complement`	-	+	-	-

Code & Documentation:

github.com/tobias-loew/flags

questions / remarks?