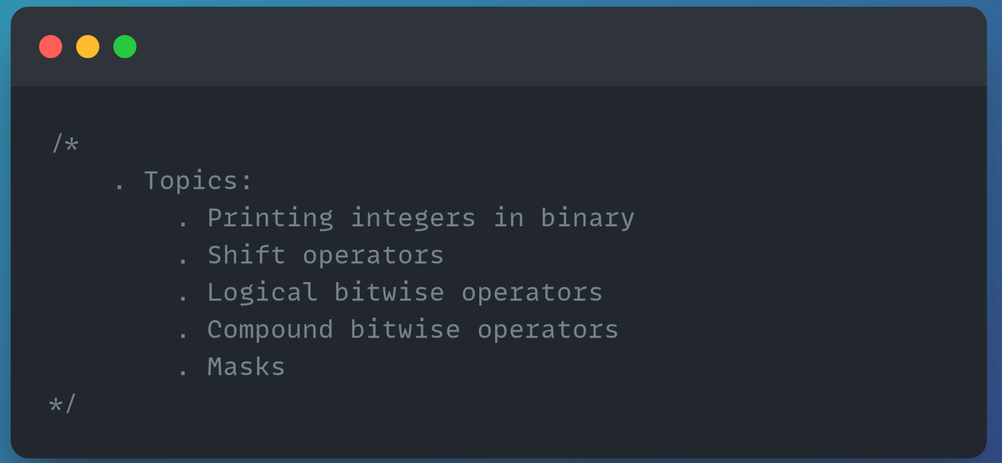
Bitwise operators



```
/*
    . Topics:
        . Printing integers in binary
        . Shift operators
        . Logical bitwise operators
        . Compound bitwise operators
        . Masks
*/
```

Printing integers in binary

```cpp
unsigned short int data {15};

fmt::println("data (dec): {0:d}", data);
fmt::println("data (oct): {0:o}", data);
fmt::println("data (hex): {0:x}", data);
fmt::println("data (bin): {:016b}", std::bitset<16>(data).to_ulong());
```

Shift operators

```
unsigned short int value {0xff0u};

fmt::println("Size of short int {0}", sizeof(short int));//  16 bits
fmt::println("(0)bin value : {:016b}, decimal value: {} ", value,value);

//Shift left by one bit
value = static_cast<unsigned short int>(value << 1);
fmt::println("(1)bin value : {:016b}, decimal value: {} ", value,value);

//Shift left by one bit
value = static_cast<unsigned short int>(value << 1);
fmt::println("(2)bin value : {:016b}, decimal value: {} ", value,value);

//Shift left by one bit
value = static_cast<unsigned short int>(value << 1);
fmt::println("(3)bin value : {:016b}, decimal value: {} ", value,value);

//Shift left by one bit
value = static_cast<unsigned short int>(value << 1);
fmt::println("(4)bin value : {:016b}, decimal value: {} ", value,value);

//Shift left by one bit
value = static_cast<unsigned short int>(value << 1);
fmt::println("(5)bin value : {:016b}, decimal value: {} ", value,value);

//Shift right by one bit
value = static_cast<unsigned short int>(value >> 1);
fmt::println("(6)bin value : {:016b}, decimal value: {} ", value,value);
```

Shift operators (contd)

```
//Shift by multiple bits in one go
//Shift right by four bits
value = static_cast<unsigned short int>(value >> 4);
fmt::println("(7)bin value : {:016b}, decimal value: {} ", value,value);
```

Logical bitwise operators

```
int COLUMN_WIDTH {20};
unsigned int value1 {0x3}; // 0000 0011
unsigned int value2 {0x5}; // 0000 0101

fmt::println("value1: {:032b}", value1);
fmt::println("value2: {:032b}", value2);

//AND
fmt::println("Bitwise AND :");
fmt::println("value1 & value2: {:032b}", value1 & value2);

//OR
fmt::println("Bitwise OR :");
fmt::println("value1 | value2: {:032b}", value1 | value2);

//NOT
fmt::println("Bitwise NOT :");
fmt::println("~value1: {:032b}", ~value1);
fmt::println("~value2: {:032b}", ~value2);

//XOR
fmt::println("Bitwise XOR :");
fmt::println("value1 ^ value2: {:032b}", value1 ^ value2);
```

Compound bitwise operators

```
unsigned int sandbox_var{0b00110100};  // 8 bits : positive numbers only

// Print out initial value
fmt::println("Initial value: ");
fmt::println("sandbox_var: {:032b}", sandbox_var);

// Compound left shift
fmt::println("Shift left 2 bit positions in place:  ");
sandbox_var <<= 2;
fmt::println("sandbox_var: {:032b}", sandbox_var);

// Compound right shift
fmt::println("Shift right 4 bit positions in place:  ");
sandbox_var >>= 4;
fmt::println("sandbox_var: {:032b}", sandbox_var);

// Compound OR with 0000 0010 to have all lower 4 bits turned on
fmt::println("Compound OR with 0000 0010 :  ");
sandbox_var |= 0b00001111;
fmt::println("sandbox_var: {:032b}", sandbox_var);

// Compound AND with 0000 1100 to turn off the 2 lowest bits
fmt::println("Compound AND with 0000 1100 :  ");
sandbox_var &= 0b000001100;
fmt::println("sandbox_var: {:032b}", sandbox_var);

//XOR with 00000011 to turn on the 4 lowest bits again
fmt::println("Compound XOR with 0000 0011 :  ");
sandbox_var ^= 0b00000011;
fmt::println("sandbox_var: {:032b}", sandbox_var);
```

Masks

```
//Masks
//Highlight position for bit of interest with a 1
//Mask other positions with 0

constexpr unsigned int mask_bit_0 {0b00000001} ;//Bit0
constexpr unsigned int mask_bit_1 {0b00000010} ;//Bit1
constexpr unsigned int mask_bit_2 {0b00000100} ;//Bit2
constexpr unsigned int mask_bit_3 {0b00001000} ;//Bit3
constexpr unsigned int mask_bit_4 {0b00010000} ;//Bit4
constexpr unsigned int mask_bit_5 {0b00100000} ;//Bit5
constexpr unsigned int mask_bit_6 {0b01000000} ;//Bit6
constexpr unsigned int mask_bit_7 {0b10000000} ;//Bit7
```

Setting bits

```
//Setting : ⊨ with mask of the bit

//Set bit 1
fmt::println("Setting bit in position 1:");
var ⊨ mask_bit_1;
fmt::println("var: {:032b}", var);


//Set bit 5
fmt::println("Setting bit in position 5: ");
var ⊨ mask_bit_5;
fmt::println("var: {:032b}", var);
```

Resetting bits

```
//RESETTING BITS : set to 0
//Resetting : &= (~mask)

//Reset bit 1
fmt::println("Resetting bit in position 1: ");
var &= (~mask_bit_1);
fmt::println("var: {:032b}", var);

//Reset bit 5
fmt::println("Resetting bit in position: 1");
var &= (~mask_bit_5);
fmt::println("var: {:032b}", var);
```

Check state of a bit

```cpp
//Check state of a bit: & with mask
fmt::println("Checking the state of each bit position (on/off): ");
fmt::println("bit0 is {}", static_cast<bool>(var & mask_bit_0));
fmt::println("bit1 is {}", static_cast<bool>(var & mask_bit_1));
fmt::println("bit2 is {}", static_cast<bool>(var & mask_bit_2));
fmt::println("bit3 is {}", static_cast<bool>(var & mask_bit_3));
fmt::println("bit4 is {}", static_cast<bool>(var & mask_bit_4));
fmt::println("bit5 is {}", static_cast<bool>(var & mask_bit_5));

fmt::println("bit6 is {}", static_cast<bool>(var & mask_bit_6));
fmt::println("bit6 is {}", (var & mask_bit_6) >> 6 );

fmt::println("bit7 is {}", static_cast<bool>(var & mask_bit_7));
fmt::println("bit7 is {}", (var & mask_bit_7) >> 7 );
```

Toggle bits

```rust
//Toggle : var ^ mask

//Toggle bit 0
fmt::println("Toggle bit 0: ");
var ^= mask_bit_0;
fmt::println("var: {:032b}", var);


//Toggle bit7
fmt::println("Toggle bit 7: ");
var ^= mask_bit_7;
fmt::println("var: {:032b}", var);

//Toggle multiple bits in one go : the 4 higher bits
fmt::println("Toggle multiple bits in one go : the 4 higher bits: ");
var ^= (mask_bit_7 | mask_bit_6 | mask_bit_5 | mask_bit_4);
fmt::println("var: {:032b}", var);
```