Outline

- In this presentation, we will:
  - Introduce bitwise logical operations
  - Contrast these with Boolean logical operations
  - Describe
    - The bitwise EXCLUSIVE OR operator
      in addition to bitwise AND and OR
    - The unary bitwise NOT or complement operator
  - Describe left and right shift operators

Logical operators

- We have seen two logical operators:
  - The binary logical AND operator and the binary logical OR operator
  - Their behavior is defined by the values of the operands:

x	y	x && y	x || y
false	false	false	false
false	true	false	true
true	true	true	true
true	false	false	true

  - Recall that any zero value is `false`, while any non-zero value is `true`
    - `true` and `false` have the values `1` and `0`, respectively

Primitive types

- Recall that primitive types are a fixed number of bits
  - Given any two bits, we could define

b_3	c_3	b_3 AND c_3	b_3 OR b_3
0	0	0	0
0	1	0	1
1	1	1	1
1	0	0	1

Bitwise AND operator

- There are three binary bitwise operators in C++
  - Given any two operands of the same type, the bitwise AND operator & compares the corresponding pairs of bits
  - Each result is 1 only if both bits are also 1

```
  00100100101010010100101001010100
& 01001010101011110100111101000001
  --------------------------------
  00000000101010010100101001000000
```


Bitwise AND operator

- Like arithmetic operations,
  the bitwise AND of any pair of bits does not affect the operands

Output:

```
  m   = 615074388
  n   = 1253003073
m & n = 11094592
```

```
0b00000000101010010100101001000000
```

```
#include <iostream>

int main();

int main() {
    unsigned int m{0b00100100101010010100101001010100};
    unsigned int n{0b01001010101011110100111101000001};
    std::cout << "  m   = " << m << std::endl;
    std::cout << "  n   = " << n << std::endl;
    std::cout << "m & n = " << (m & n) << std::endl;

    return 0;
}
```


Bitwise OR operator

- The second is bitwise OR operator |
  - Given any two operands of the same type a logical OR to each corresponding pair of bits
  - Each result is 0 only if both bits are also 0

```
  00100100101010010100101001010100
| 01001010101011110100111101000001
  --------------------------------
  01101110101011110100111101010101
```


Bitwise OR operator

- Like arithmetic operations,
  the bitwise OR of any pair of bits does not affect the operands

Output:

```
  m   = 615074388
  n   = 1253003073
m | n = 1856982869
```

```
0b01101110101011110100111101010101
```

```
#include <iostream>

int main();

int main() {
    unsigned int m{0b00100100101010010100101001010100};
    unsigned int n{0b01001010101011110100111101000001};
    std::cout << "  m   = " << m << std::endl;
    std::cout << "  n   = " << n << std::endl;
    std::cout << "m | n = " << (m | n) << std::endl;

    return 0;
}
```

Bitwise EXCLUSIVE-OR operator

- The third is bitwise XOR operator
  - This has no equivalent binary logical operator
  - For this result to be `true`, one but not both operands must be `true`

b_1	b_2	b_1 AND b_2	b_1 OR b_2	b_1 XOR b_2
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Bitwise XOR operator

- The third is bitwise XOR operator ^
  - This has no equivalent binary logical operator
  - If both bits have the same value, the result is `0`, otherwise it is `1`

```
  00100100101010010100101001010100
^ 01001010101011110100111101000001
  01101110000001100000010100010101
```


Bitwise XOR operator

- Like arithmetic operations,
  the bitwise XOR of any pair of bits does not affect the operands

```
#include <iostream>

int main();

int main() {
    unsigned int m{0b00100100101010010100101001010100};
    unsigned int n{0b01001010101011110100111101000001};
    std::cout << "  m   = " << m << std::endl;
    std::cout << "  n   = " << n << std::endl;
    std::cout << "m ^ n = " << (m ^ n) << std::endl;

    return 0;
}
```

Output:

```
  m   = 615074388
  n   = 1253003073
m ^ n = 1845888277

0b01101110000001100000010100010101
```


Automatic bitwise assignment

- For each binary bitwise operator, there is an automatic assignment operator:

Assignment	Automatic assignment	Name
`a = a & 32`	`a &= 32`	auto bitwise AND
`b = b | 41`	`b |= 41`	auto bitwise OR
`c = 2 ^ c`	`c ^= 2`	auto bitwise XOR

- Note: there are no Boolean automatic assignment operators
  - The operators &&= and ||= do not exist in C++

Unary bitwise NOT operator

- A unary bitwise operator is the NOT operator ~
  - It is equivalent to applying the logical NOT operator ~ to each bit

```
~ 01001010101011110100111101000001
  10110101010100001011000010111110
```


Bitwise AND operator

- Like arithmetic operations,
  the bitwise AND of any pair of bits does not affect the operands

```
#include <iostream>

int main();

int main() {
    unsigned int m{0b00100100101010010100101001010100};
    std::cout << " m = " <<   m  << std::endl;
    std::cout << "~m = " << (~m) << std::endl;

    return 0;
}
```

Output:
```
 m = 615074388
~m = 3679892907
```

`0b11011011010101101011010110101011`

Application of bitwise operators

- Bitwise operators allow the manipulation of individual bits
  - Suppose this local variable has exactly one 1 bit
    `unsigned int MASK{256}; // 0b00000000000000000000000100000000`
  - Suppose n is any unsigned integer value:
    ```
    unsigned int n{};
    std::cin >> n;
    ```
    Bit 0
    - We can set the 8th bit of n to 1:
      `n |= MASK;`
    - We can set the 8th bit of n to 0:
      `n &= ~MASK;`
    - We can flip the 8th bit of n between 0 and 1:
      `n ^= MASK;`
    - We can have a condition that is true if the 8th bit is 1:
      ```
      if ( n & MASK ) {
          // Do something if the 8th bit is 1
      }
      ```


Application of bitwise operators

```
#include <iostream>
int main();

int main() {
    unsigned short MASK_0{1} // 0b0000000000000001
    unsigned short MASK_1{2} // 0b0000000000000010
    unsigned short MASK_2{4} // 0b0000000000000100
    unsigned short MASK_3{8} // 0b0000000000001000

    unsigned short n{};
    std::cout << "Enter a positive integer: ";
    std::cin >> n;

    n |=  MASK_0;      // Set bit 0 to 1
    n &= ~MASK_1;      // Set bit 1 to 0
    n ^=  MASK_2;      // Flip the value of bit 2

    if ( n & MASK_3 ) {
       std::cout << "Bit 3 is '1'" << std::endl;
    } else {
        std::cout << "Bit 3 is '0'" << std::endl;
    }

    return 0;
}
```

`000···0001000`

`Bit 3 is '1'`

Bit-shift operators

- There are two operators that literally shift bits left or right:
  - The left-shift operator << evaluates to
    the bits of the operand `op` shifted to the left by `n` bits

```
unsigned int op{17};
unsigned int q1{ op << n };
                    // n is any non-negative integer
```

  - The right-shift operator >> evaluates to
    the bits of the operand `op` shifted to the right by `n` bits

```
unsigned int q2{ op >> n };
                    // n is any non-negative integer
```

- Any bits shifted beyond the last position are lost

Bit-shift operators

- Examples:
  - If `op` is four bytes and has the value
    `00100100111110010100111001010100`
  - The result of `op >> 5` is
    `00000001001001111100101001110010`
  - The result of `op >> 12` is
    `00000000000000100100111110010100`
  - The result of `op << 8` is
    `11111001010011100101010000000000`
  - The result of `op << 13` is
    `00101001110010101000000000000000`

Automatic bit-shift assignment

- There are two automatic bit-shift operators
  - Shift the bits in the operand `op` to the left by `n` bits
    `op <<= n;`
  - Shift the bits in the operand `op` to the right by `n` bits
    `op >>= n;`

Application of bit-shift operators

- Bit-shift operators can be used to precisely read or place bits
  - In our next example, we will use bit shifting and bitwise AND to print a number to the screen in binary

Application of bit-shift operators

```
#include <iostream>
int main();

int main() {
    unsigned int n;
    std::cout << "Enter a positive integer: ";
    std::cin >> n;

    for ( unsigned int k{1 << 31}; k > 0; k >>= 1 ) {
        if ( n & k ) {
            std::cout << "1";
        } else {
            std::cout << "0";
        }
    }

    std::cout << std::endl;

    return 0;
}
```

00000000000000000000000000000000

00111011100110101100100111111111

Summary of operators

- To summarize our knowledge of operators

Operator	Binary	Unary
Arithmetic	`+ - * / %`	`+ -`
Comparison	`< <= == != >= >`	
Logical	`&& ||`	`!`
bitwise	`& | ^`	`~`
Bit shift	`<< >>`	
Assignment	`=`	
Arithmetic auto-assignment	`+= -= *= /= %=`	`++ --`
Bitwise auto-assignment	`&= |= ^=`	
Bit-shift auto-assignment	`<<= >>=`	

Summary

- In this presentation, you now
  - Are aware of bitwise and bit-shifting operators
  - Understand the behavior of these operators
  - Understand the automatic operators corresponding to these
    - There are no &&= or ||= operators

References

[1] No references?

Colophon

These slides were prepared using the Georgia typeface. Mathematical equations use Times New Roman, and source code is presented using `Consolas`.

The photographs of lilacs in bloom appearing on the title slide and accenting the top of each other slide were taken at the Royal Botanical Gardens on May 27, 2018 by Douglas Wilhelm Harder. Please see

https://www.rbg.ca/

for more information.

Disclaimer

These slides are provided for the ECE 150 *Fundamentals of Programming* course taught at the University of Waterloo. The material in it reflects the authors' best judgment in light of the information available to them at the time of preparation. Any reliance on these course slides by any party for any other purpose are the responsibility of such parties. The authors accept no responsibility for damages, if any, suffered by any party as a result of decisions made or actions based on these course slides for any other purpose than that for which it was intended.