Programming Languages and Tools: Programming with C++ CS:3210:0003

Lecture/Lab #6

Conditional Operator

Operator	Description	Fixedness	Arity	Input Type	Return Type
=	Assignment	Infix	Binary	Any Type *T*	*T*
+, -, *, /, %	Arithmetic	Infix	Unary, Binary	ints, floats	ints, floats
++, --	Increment, Decrement	Prefix/Postfix	Unary	int	int
==, !=, <, >, <=, >=	Relational	Infix	Binary	ints, floats	bool
!	NOT	Prefix	Unary	bool	bool
&&, ||, ^	AND, OR, XOR	Infix	Binary	bools	bool
?:	Conditional	Infix	Ternary	bool, T, T	T

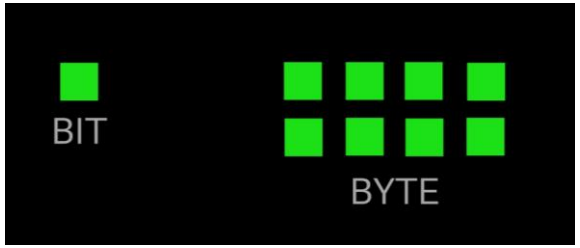
Simplify int2bin.cpp using ?:

sizeof Operator

- Amount of memory reserved for a type:

```
sizeof(t);          //t is a type, or
sizeof(x);          //x has type t
```

- Returns size in bytes
- Fixed-width integer types from <cstdint>:



Width	Unsigned Type	Signed Type
8	int8_t	uint8_t
16	int16_t	uint16_t
32	int32_t	uint32_t
64	int64_t	uint64_t

Narrowing Errors

- What happens when we try to fit an out-of-range value into a variable of a machine integer type? Integer overflow
- For unsigned integers, value *wraps around*
- For signed integers, *undefined behavior* (could wrap around)
- To avoid narrowing during initialization, use *list initialization* syntax:

```
varType varName {value};          // to initialize varName with value
```

Activity

Update int2bin.cpp so that

- For non-negative integers > 15, it wraps around
- For negative integers, it prints an out-of-range error message (as before)

Type Conversion

- Converting value from type to type
- Produces a new value, doesn't actually convert
- Compiler can do implicit type conversion
- For types where data is lost, some compilers display warnings
- Write type-safe programs:
  - either use explicit type conversion, or
  - correctly type your values
- auto declaration for automatic type inference

  auto varName = value;
- Compiler error if compiler can't infer type

Bitwise Operators

Operator	Description	Fixedness	Arity	Input Type	Return Type
=	Assignment	Infix	Binary	Any Type *T*	*T*
+, -, *, /, %	Arithmetic	Infix	Unary, Binary	ints, floats	ints, floats
++, --	Increment, Decrement	Prefix/Postfix	Unary	int	int
==, !=, <, >, <=, >=	Relational	Infix	Binary	ints, floats	bool
!	NOT	Prefix	Unary	bool	bool
&&, ||, ^	AND, OR, XOR	Infix	Binary	bools	bool
?:	Conditional	Infix	Ternary	bool, T, T	T
~	Bitwise NOT	Prefix	Unary	int	int
&, |, ^	Bitwise AND, OR, XOR	Infix	Binary	ints	int

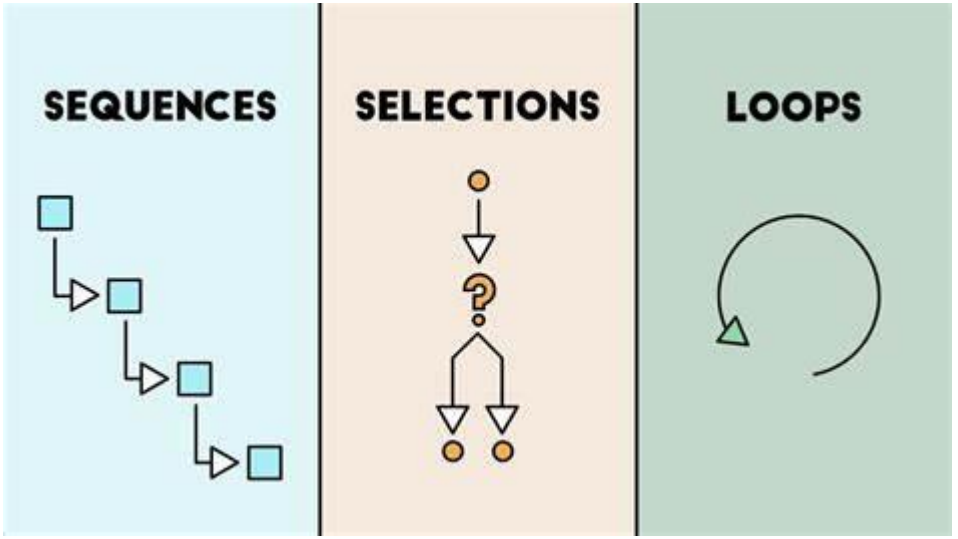
Bit Shifts

Operator	Description	Fixedness	Arity	Input Type	Return Type
=	Assignment	Infix	Binary	Any Type *T*	*T*
+, -, *, /, %	Arithmetic	Infix	Unary, Binary	ints, floats	ints, floats
++, --	Increment, Decrement	Prefix/Postfix	Unary	int	int
==, !=, <, >, <=, >=	Relational	Infix	Binary	ints, floats	bool
!	NOT	Prefix	Unary	bool	bool
&&, ||, ^	AND, OR, XOR	Infix	Binary	bools	bool
?:	Conditional	Infix	Ternary	bool, T, T	T
~	Bitwise NOT	Prefix	Unary	int	int
&, |, ^	Bitwise AND, OR, XOR	Infix	Binary	ints	int
<<, >>	Bit shifts	Infix	Binary	ints	int

Bit Shifts

- Shift all bits to right/left.
  - x << 1 is equivalent to x * 2
  - x >> 1 is equivalent to x / 2
- In general,
  - x << n is equivalent to $x * 2^n$
  - x >> n is equivalent to $x / 2^n$
- For left shift, 0 bits are shifted in
- For right shifts,
  - 0 bits are shifted in for positive integers
  - 1 bits are shifted in for negative integers

while Loops



- For repeated execution
  - Use functions
  - Use loops when there is a common pattern of change in each execution
- Syntax:

```
while(expression) {
   //if expression evaluates to true
   StatementBlock;
}
```

- Repeats as long as expression evaluates to true
- Need to make expression false at some point within the body to avoid infinite loop

Practice with while Loops