Integer division
Type conversion
Casting
Modular arithmetic
More operators
Increment and decrement
Other assignment operators
Relational operators
Logical operators
Precedence and associativity

Operators, integer division, casting, modular arithmetic

Comp Sci 1570 Introduction to C++

Outline

1. Integer division
2. Type conversion
   - Casting
3. Modular arithmetic
4. More operators
   - Increment and decrement
   - Other assignment operators
   - Relational operators
   - Logical operators
   - Precedence and associativity

Integer division

Integer division
Type conversion
Casting
Modular arithmetic
More operators
Increment and decrement
Other assignment operators
Relational operators
Logical operators
Precedence and associativity

```
#include <iostream>
using namespace std;

int main() {
  float celc;
  int fahr;

  celc = (5/9)*(fahr-32);
  cout << celc << endl;

  celc = (5.0/9)*(fahr-32);
  cout << celc << endl;

  return 0;
}
```

Integer division

Integer division
Type conversion
Casting
Modular arithmetic
More operators
Increment and decrement
Other assignment operators
Relational operators
Logical operators
Precedence and associativity

- When an int (int-type) is divided by another int, the result is an int.
- Program to convert Fahrenheit to Celcius (see code)
- Regardless of the value of fahr that is used, celc will be assigned 0.
- Both of the literal constants 5 and 9 are stored by the compiler as integers.
- Integer division will give you 0 (9 goes into 5 zero times), and 0 times anything is 0.

Integer division
Type conversion
Casting
Modular arithmetic
More operators
Increment and decrement
Other assignment operators
Relational operators
Logical operators
Precedence and associativity

Outline

1. Integer division
2. Type conversion
   - Casting
3. Modular arithmetic
4. More operators
   - Increment and decrement
   - Other assignment operators
   - Relational operators
   - Logical operators
   - Precedence and associativity

Type conversion

Integer division
Type conversion
Casting
Modular arithmetic
More operators
Increment and decrement
Other assignment operators
Relational operators
Logical operators
Precedence and associativity

Implicit conversions are automatically performed when a value is copied to a compatible type.

```
short a=2000;
int b;
b=a;
```

- The value of a is promoted from short to int without the need of any explicit operator.
- This is known as a standard conversion.
- Standard conversions affect fundamental data types, and allow the conversions between numerical types (short to int, int to float, double to int...), to or from bool, and more
- Converting to int from some smaller integer type, or to double from float is known as promotion, and is guaranteed to produce the exact same value in the destination type.

Conversions

```
int integer1;
float float1;
float1 + integer1 // gives a float
float1 − integer1 // gives a float
float1 * integer1 // gives a float
float1 / integer1 // gives a float
integer1 / float1 // gives a float
float1 % integer1 // can't be done
integer1 % float1 // can't be done
```

Alternatively, if conversion is from a floating-point type to an integer type, the value is truncated (the decimal part is removed). If the result lies outside the range of representable values by the type, the conversion causes undefined behavior.

Integer division
Type conversion
Casting
Modular arithmetic
More operators
Increment and decrement
Other assignment operators
Relational operators
Logical operators
Precedence and associativity

Outline

1 Integer division

2 Type conversion
   Casting

3 Modular arithmetic

4 More operators
   Increment and decrement
   Other assignment operators
   Relational operators
   Logical operators
   Precedence and associativity

Integer division
Type conversion
Casting
Modular arithmetic
More operators
Increment and decrement
Other assignment operators
Relational operators
Logical operators
Precedence and associativity

Casting

- C++ is a strong-typed language.
- Many conversions, specially those that imply a different interpretation of the value, require an explicit conversion, known in C++ as type-casting.
- Several forms for generic type-casting:

```
double x = 10.3;
int y;
y = int (x);      // functional notation
y = (int) x;      // c-like cast notation

// static casting
int someValue;
double Num1, Num2;
someValue = Num1 + Num2; // warning issued
someValue = static_cast<int>(Num1 + Num2);
```

Integer division
Type conversion
Casting
Modular arithmetic
More operators
Increment and decrement
Other assignment operators
Relational operators
Logical operators
Precedence and associativity

Static casting

We want average_age to have a decimal point, but the result of the quotient on the rhs will be an int.

```
int total_of_ages;
int numPeople;
average_age = total_of_ages / numPeople;
average_age = static_cast<float>(total_of_ages) / numPeople;
```

- We cast either the numerator or the denominator or both, but NOT the quotient.
- Now, a float divided by an int will give you a float as desired.
- Note: you have not changed the nature of either total_of_ages or numPeople, and they are both still ints after this line of code is executed.

Outline

1. Integer division
2. Type conversion
   - Casting
3. Modular arithmetic
4. More operators
   - Increment and decrement
   - Other assignment operators
   - Relational operators
   - Logical operators
   - Precedence and associativity

mod

The mod operator, %, works this way:

- a mod b is the remainder after a is divided by b.
- 4%7 is 4 (since 4/7 is 0 with remainder 4)
- 7%3 is 1 (since 7/3 is 2 with remainder 1)
- 27%3 is 0 (since 27/3 is 9 with remainder 0)

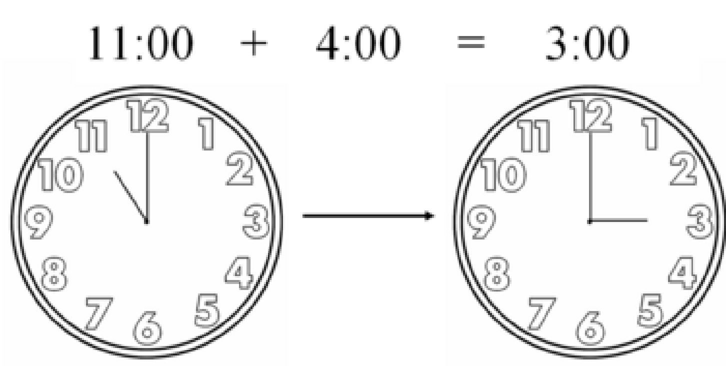
mod to choose place

1. Suppose you read in an integer from a user into a variable named x.
2. Assume x is 5 digits long, and let's represent it as x=abcde.
3. So, e is the "ones" digit, d is the "tens" digit, etc.
4. Thus, we don't know any of these digits at compile time.
5. But suppose that we need to know, say, the tens digit, d, at run-time.
6. How can we extract that from the value x, entered by the user at run-time?
7. Well, $x\%100$ is the integer de.
8. This is because 100 goes into x abc times with a remainder of de.
9. Now, de/10 is d. That is, 10 goes into de d times.

```
int tens_digit;
tens_digit = (x%100)/10; //assigns tens digit
```

Modular arithmetic

Integer division
Type conversion
Casting
Modular arithmetic
More operators
Increment and decrement
Other assignment operators
Relational operators
Logical operators
Precedence and associativity



$$(11 + 4)\%12 = 3$$

Integer division
Type conversion
Casting
Modular arithmetic
More operators
Increment and decrement
Other assignment operators
Relational operators
Logical operators
Precedence and associativity

Outline

1. Integer division
2. Type conversion
   - Casting
3. Modular arithmetic
4. More operators
   - Increment and decrement
   - Other assignment operators
   - Relational operators
   - Logical operators
   - Precedence and associativity

Outline

1 Integer division

2 Type conversion
Casting

3 Modular arithmetic

4 More operators
Increment and decrement
Other assignment operators
Relational operators
Logical operators
Precedence and associativity

Increment and decrement operators

Integer division
Type conversion
Casting
Modular arithmetic
More operators
Increment and decrement
Other assignment operators
Relational operators
Logical operators
Precedence and associativity

A=10; B=20;

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiplies both operands	A * B will give 200
/	Divides numerator by de-numerator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	**Increment operator** , increases integer value by one	A++ will give 11
--	**Decrement operator** , decreases integer value by one	A-- will give 9

Increment and decrement operators

Integer division
Type conversion
Casting
Modular arithmetic
More operators
Increment and decrement
Other assignment operators
Relational operators
Logical operators
Precedence and associativity

Operator	Symbol	Form	Operation
Prefix increment (pre-increment)	++	++x	Increment x, then evaluate x
Prefix decrement (pre-decrement)	—	—x	Decrement x, then evaluate x
Postfix increment (post-increment)	++	x++	Evaluate x, then increment x
Postfix decrement (post-decrement)	—	x—	Evaluate x, then decrement x

Syntax

Integer division
Type conversion
Casting
Modular arithmetic
More operators
Increment and decrement
Other assignment operators
Relational operators
Logical operators
Precedence and associativity

To increment (or decrement) an integer variable's value.
This is common in looping structures (repeated operations).

```
val = val + 1; // retrieve val, add one, replac
```

```
val = val + 1; // increment
val++;      or      ++val;
```

```
val = val - 1; // decrement
val--;      or      --val;
```

Order matters

Integer division
Type conversion
Casting
Modular arithmetic
More operators
Increment and decrement
Other assignment operators
Relational operators
Logical operators
Precedence and associativity

Equivalent to val = val + 1; but are much faster such that if this operation is repeated millions of times, time savings is significant.

```
int val = 6, num;
num = ++val;
// num is now 7, val is now 7

int val = 6, num;
num = val++;
// num is now 6, val is now 7
```

- The two versions of increment (or decrement) are NOT the same.
- val++ is a post-increment while ++val is a pre-increment.
- It is when these statements are inserted into bigger statements that the difference becomes apparent.

Order matters

Integer division
Type conversion
Casting
Modular arithmetic
More operators
Increment and decrement
Other assignment operators
Relational operators
Logical operators
Precedence and associativity

Example 1	Example 2

`x = 3;`
`y = ++x;`
`// x contains 4, y contains 4`
	`x = 3;`
	`y = x++;`
	`// x contains 4, y contains 3`

In any C++ statement, a pre-increment is executed before anything else, while a post-increment is executed last.

Integer division
Type conversion
Casting
Modular arithmetic
More operators
Increment and decrement
Other assignment operators
Relational operators
Logical operators
Precedence and associativity

Outline

1. Integer division
2. Type conversion
   - Casting
3. Modular arithmetic
4. More operators
   - Increment and decrement
   - Other assignment operators
   - Relational operators
   - Logical operators
   - Precedence and associativity

Other fast operators

Integer division
Type conversion
Casting
Modular arithmetic
More operators
Increment and decrement
Other assignment operators
Relational operators
Logical operators
Precedence and associativity

```
x += y; // equivalent to x = x + y;
x -= y; // equivalent to x = x - y;
x /= y; // equivalent to x = x / y;
x *= y; // equivalent to x = x * y;
x %= y; // equivalent to x = x % y;
```

expression	equivalent to...
`y += x;`	`y = y + x;`
`x -= 5;`	`x = x - 5;`
`x /= y;`	`x = x / y;`
`price *= units + 1;`	`price = price * (units+1);`

Integer division
Type conversion
Casting
Modular arithmetic
More operators
Increment and decrement
Other assignment operators
Relational operators
Logical operators
Precedence and associativity

Fast operators

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand.	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand.	C %= A is equivalent to C = C % A

Outline

Integer division
Type conversion
Casting
Modular arithmetic
More operators
Increment and decrement
Other assignment operators
Relational operators
Logical operators
Precedence and associativity

1. Integer division
2. Type conversion
   - Casting
3. Modular arithmetic
4. More operators
   - Increment and decrement
   - Other assignment operators
   - Relational operators
   - Logical operators
   - Precedence and associativity

Relational operators

- What are expressions?
- C++ statements that will evaluate to either true or false
- false is interpreted as a 0 and 0 is interpreted as false
- true is interpreted as 1 and any number other than 0 is interpreted as true

Operator	Symbol	Form	Operation
Greater than	>	x > y	true if x is greater than y, false otherwise
Less than	<	x < y	true if x is less than y, false otherwise
Greater than or equals	>=	x >= y	true if x is greater than or equal to y, false otherwise
Less than or equals	<=	x <= y	true if x is less than or equal to y, false otherwise
Equality	==	x == y	true if x equals y, false otherwise
Inequality	!=	x != y	true if x does not equal y, false otherwise

Integer division
Type conversion
Casting
Modular arithmetic
More operators
Increment and decrement
Other assignment operators
Relational operators
Logical operators
Precedence and associativity

Assume variable A holds 10 and variable B holds 20

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Relational operators

The relational operators are:
”¡”, ”¡=”, ”¿”, ”¿=”, ”==”, ”!=”.

```
short val = 5, num = 8, bob = 0;
(val <= num); // evals to true (or 1)
(num % val > bob); // evals to true
(val == num); // true
(num != (num/val)); // true
```

Integer division
Type conversion
Casting
Modular arithmetic
More operators
Increment and decrement
Other assignment operators
Relational operators
Logical operators
Precedence and associativity

== vs. =

- The == operator is the "is equal" operator and "!=" is the "is not equal" operator.
- Many times, those learning C++ for the first time will make a mistake when trying to use this operator that the compiler will NOT catch.
- The code will compile and run, but incorrectly!
- They will use the = operator instead of the == operator.
- Thus, val = num will compile and run but will NOT compare the two values.
- It will set the value of the variable val to that of num and will return true...always.
- This is not the desired result.
- BE CAREFUL.

Examples

Integer division
Type conversion
Casting
Modular arithmetic
More operators
Increment and decrement
Other assignment operators
Relational operators
Logical operators
Precedence and associativity

Suppose that a=2, b=3, and c=6, then:

```
(7 == 5)        // evaluates to false
(5 > 4)         // evaluates to true
(3 != 2)        // evaluates to true
(6 >= 6)        // evaluates to true
(5 < 5)         // evaluates to false
(a == 5)        // evaluates to false
(a*b >= c)      // evaluates to true
(b+4 > a*c)     // evaluates to false
((b=2) == a)    // evaluates to true
```

Outline

1 Integer division

2 Type conversion
  Casting

3 Modular arithmetic

4 More operators
  Increment and decrement
  Other assignment operators
  Relational operators
  Logical operators
  Precedence and associativity

Integer division
Type conversion
Casting
Modular arithmetic
More operators
Increment and decrement
Other assignment operators
Relational operators
Logical operators
Precedence and associativity

Integer division
Type conversion
Casting
Modular arithmetic
More operators
Increment and decrement
Other assignment operators
Relational operators
Logical operators
Precedence and associativity

AND, OR

Assume variable A holds 1 and variable B holds 0

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true, then Logical NOT operator will make false.	!(A && B) is true.

Operator	Symbol	Form	Operation
Logical NOT	!	!x	true if x is false, or false if x is true
Logical AND	&&	x && y	true if both x and y are true, false otherwise
Logical OR	||	x || y	true if either x or y are true, false otherwise

Integer division
Type conversion
Casting
Modular arithmetic
More operators
Increment and decrement
Other assignment operators
Relational operators
Logical operators
Precedence and associativity

AND, OR

&& OPERATOR (and)		
a	b	a && b
true	true	true
true	false	false
false	true	false
false	false	false

|| OPERATOR (or)		
a	b	a || b
true	true	true
true	false	true
false	true	true
false	false	false

Logical AND (operator &&)		
Left operand	Right operand	Result
false	false	false
false	true	false
true	false	false
true	true	true

Logical OR (operator ||)		
Left operand	Right operand	Result
false	false	false
false	true	true
true	false	true
true	true	true

Integer division
Type conversion
Casting
Modular arithmetic
More operators
Increment and decrement
Other assignment operators
Relational operators
Logical operators
Precedence and associativity

NOT

Logical NOT (operator !)	
Right operand	**Result**
true	false
false	true

- The operator ! is the C++ operator for the Boolean operation NOT.
- It has only one operand, to its right, and inverts it, producing false if its operand is true, and true if its operand is false.
- Returns the opposite Boolean value of evaluating its operand.

```
!(5 == 5)    // evaluates to false because the
!(6 <= 4)    // evaluates to true because (6 <=
!true        // evaluates to false
!false       // evaluates to true
```

short-circuit

Integer division
Type conversion
Casting
Modular arithmetic
More operators
Increment and decrement
Other assignment operators
Relational operators
Logical operators
Precedence and associativity

- When using the logical operators, C++ only evaluates what is necessary from left to right to come up with the combined relational result, ignoring the rest.
- Therefore, in the last example $((5 == 5)||(3 > 6))$, C++ evaluates first whether 5==5 is true, and if so, it never checks whether $3 > 6$ is true or not.
- This is known as short-circuit evaluation, and works like this for these operators:
- && if the left-hand side expression is false, the combined result is false (the right-hand side expression is never evaluated).
- || if the left-hand side expression is true, the combined result is true (the right-hand side expression is never evaluated).

Integer division
Type conversion
Casting
Modular arithmetic
More operators
Increment and decrement
Other assignment operators
Relational operators
Logical operators
Precedence and associativity

Which short-circuit?

```
short val = 5;
short num = 8;
short bob = 0;

((val == num) || (!val));
/*
false, since val is not equal to num (F),
val is true (5 same as true),
so not true is false,
and F || F is false
*/

( (5 == 5) && (3 > 6) )   // evaluates to false
( (5 == 5) || (3 > 6) )   // evaluates to true
```

Integer division
Type conversion
Casting
Modular arithmetic
More operators
Increment and decrement
Other assignment operators
Relational operators
Logical operators
Precedence and associativity

XOR

Logical XOR		
Left operand	**Right operand**	**Result**
false	false	false
false	true	true
true	false	true
true	true	false

```
if (a != b) ... // a XOR b, assuming bool
if (a != b != c != d)
```

- C++ doesn't provide a logical XOR operator.
- Unlike logical OR or logical AND, XOR cannot be short circuit evaluated.
- Because of this, making an XOR operator out of logical OR and logical AND operators is challenging.
- However, you can easily mimic logical XOR using the not equals operator (!=):

Integer division
Type conversion
Casting
Modular arithmetic
More operators
Increment and decrement
Other assignment operators
Relational operators
Logical operators
Precedence and associativity

Outline

1. Integer division
2. Type conversion
   - Casting
3. Modular arithmetic
4. More operators
   - Increment and decrement
   - Other assignment operators
   - Relational operators
   - Logical operators
   - Precedence and associativity

Integer division
Type conversion
Casting
Modular arithmetic
More operators
Increment and decrement
Other assignment operators
Relational operators
Logical operators
Precedence and associativity

Precedence and associativity

<table>
<tr><th>Category</th><th>Operator</th><th>Associativity</th></tr>
<tr><td>Postfix</td><td>() [] -> . ++ - -</td><td>Left to right</td></tr>
<tr><td>Unary</td><td>+ - ! ~ ++ - - (type)* & sizeof</td><td>Right to left</td></tr>
<tr><td>Multiplicative</td><td>* / %</td><td>Left to right</td></tr>
<tr><td>Additive</td><td>+ -</td><td>Left to right</td></tr>
<tr><td>Shift</td><td><< >></td><td>Left to right</td></tr>
<tr><td>Relational</td><td>< <= > >=</td><td>Left to right</td></tr>
<tr><td>Equality</td><td>== !=</td><td>Left to right</td></tr>
<tr><td>Bitwise AND</td><td>&</td><td>Left to right</td></tr>
<tr><td>Bitwise XOR</td><td>^</td><td>Left to right</td></tr>
<tr><td>Bitwise OR</td><td>|</td><td>Left to right</td></tr>
<tr><td>Logical AND</td><td>&&</td><td>Left to right</td></tr>
<tr><td>Logical OR</td><td>||</td><td>Left to right</td></tr>
<tr><td>Conditional</td><td>?:</td><td>Right to left</td></tr>
<tr><td>Assignment</td><td>= += -= *= /= %=>>= <<= &= ^= |=</td><td>Right to left</td></tr>
<tr><td>Comma</td><td>,</td><td>Left to right</td></tr>
</table>

Not required to know this, just for anyone curious