Compound Assignment Operators

(variable on left MUST occur on right in order to use!)

In C++, you can combine arithmetic and assignments.

For example, the statement

```
      total += cans * CAN_VOLUME;
```

is a shortcut for

```
      total = total + cans * CAN_VOLUME;
```

Similarly,

```
      total *= 2;
```

is another way of writing

```
      total = total * 2;
```

also available:

/=

-=

Many programmers *prefer* using this form of coding.

Arithmetic Operators (2.3)

C++ has the same arithmetic operators as a calculator:

* for multiplication: a * b
(not a · b or ab as in math)

/ for division: a / b
(not ÷ or a fraction bar as in math)

+ for addition: a + b

- for subtraction: a - b

put a space before and a space after each operator

Arithmetic Operators – Precedence

PEMDAS

Just as in regular algebraic notation,
* and / have higher precedence
than + and –.

In `a + b / 2`,
the `b / 2` happens first.

parentheses are used to override precedence rules

(a+b)/2 ← add parentheses if want addition done first

Arithmetic Operators – Two Kinds of Division

- If both arguments of / are integers, the remainder is discarded:
  `7 / 3` is 2, not `2.5`

you get an integer answer!

- but

```
      7.0 / 4.0
      7 / 4.0
      7.0 / 4
```

- all yield `1.75`.

at least one argument must be real (decimal pt) to get a real answer

not considered efficient b/c the computer must coerce the integer argument into a real argument

Arithmetic Operators – Getting the Remainder

- The % operator computes the remainder of an integer division.
- It is called the ***modulus operator***
  (also modulo and mod)

EX: 5 % 3 would be 2 ← the remainder

- It has nothing to do with the % key on a calculator

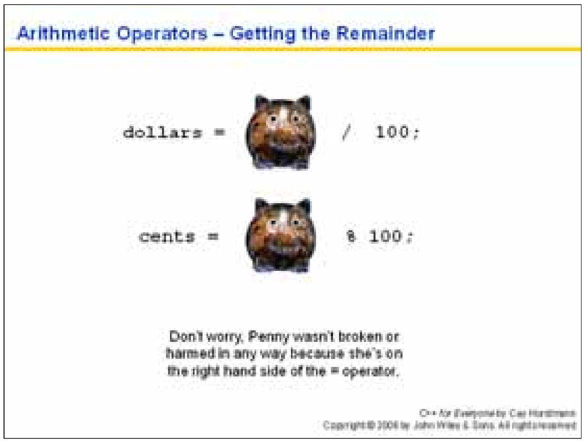
Arithmetic Operators – Getting the Remainder

EX: Time to break open the piggy bank. ← contains only pennies

You want to determine the value in dollars and cents stored in the piggy bank.
You obtain the dollars through an integer division by 100.
The integer division discards the remainder.
To obtain the remainder (cents), use the % operator:

```
int pennies = 1729;
int dollars = pennies / 100; // Sets dollars to 17
int cents = pennies % 100; // Sets cents to 29
```

(yes, 100 is a magic number)

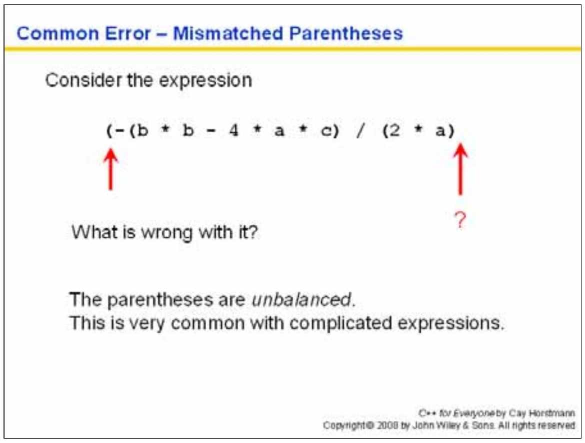


Arithmetic Operators – Getting the Remainder

```
dollars =         /  100;

 cents =          % 100;
```

Don't worry, Penny wasn't broken or harmed in any way because she's on the right hand side of the = operator.



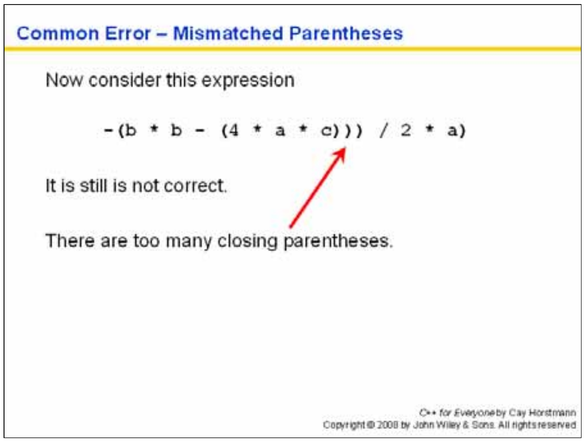
Common Error – Mismatched Parentheses

Consider the expression

```
(-(b * b - 4 * a * c) / (2 * a)
```

What is wrong with it?

The parentheses are *unbalanced*.
This is very common with complicated expressions.



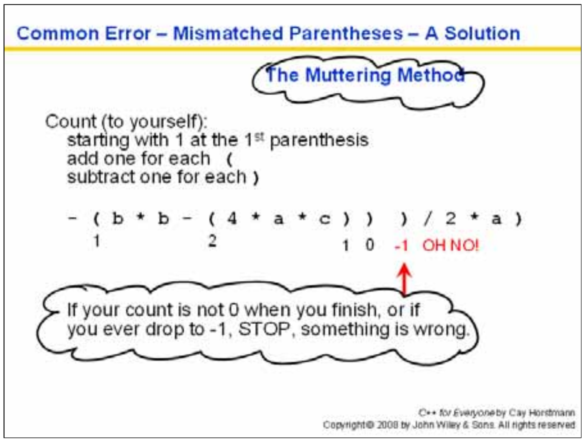
Common Error – Mismatched Parentheses

Now consider this expression

```
-(b * b - (4 * a * c))) / 2 * a)
```

It is still is not correct.

There are too many closing parentheses.



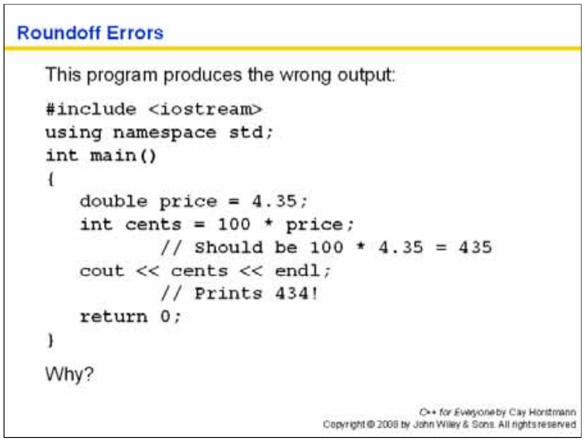
Common Error – Mismatched Parentheses – A Solution

The Muttering Method

Count (to yourself):
starting with 1 at the 1st parenthesis
add one for each (
subtract one for each)

```
- ( b * b - ( 4 * a * c ) )  ) / 2 * a )
  1         2             1 0 -1  OH NO!
```

If your count is not 0 when you finish, or if you ever drop to -1, STOP, something is wrong.

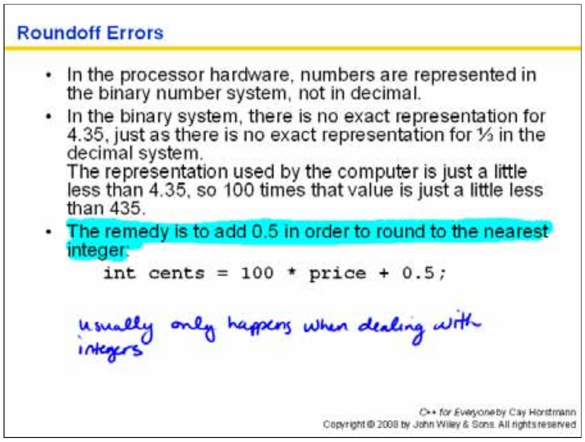


Roundoff Errors

This program produces the wrong output:

```
#include <iostream>
using namespace std;
int main()
{
   double price = 4.35;
   int cents = 100 * price;
          // Should be 100 * 4.35 = 435
   cout << cents << endl;
          // Prints 434!
   return 0;
}
```

Why?

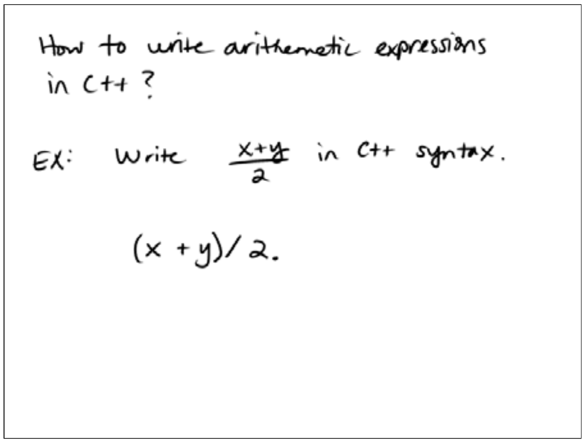


Roundoff Errors

- In the processor hardware, numbers are represented in the binary number system, not in decimal.
- In the binary system, there is no exact representation for 4.35, just as there is no exact representation for ⅓ in the decimal system.
  The representation used by the computer is just a little less than 4.35, so 100 times that value is just a little less than 435.
- The remedy is to add 0.5 in order to round to the nearest integer:

  ```
  int cents = 100 * price + 0.5;
  ```

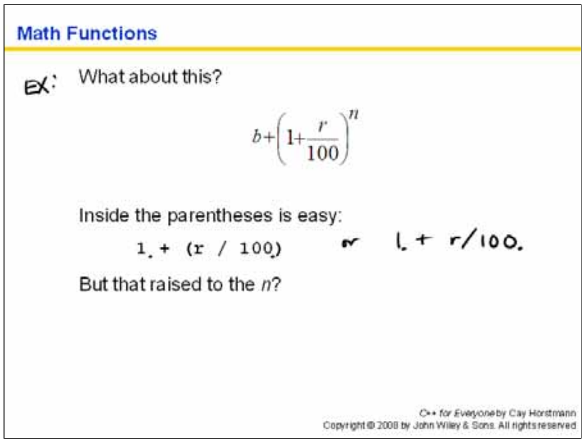
usually only happens when dealing with integers



How to write arithmetic expressions in C++?

EX: Write $\frac{x+y}{2}$ in C++ syntax.

(x + y)/2.



Math Functions

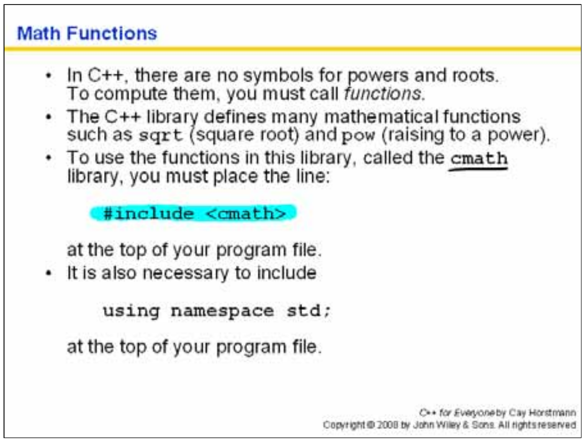
EX: What about this?

$$b+\left(1+\frac{r}{100}\right)^n$$

Inside the parentheses is easy:

`1. + (r / 100.)` or 1. + r/100.

But that raised to the *n*?



Math Functions

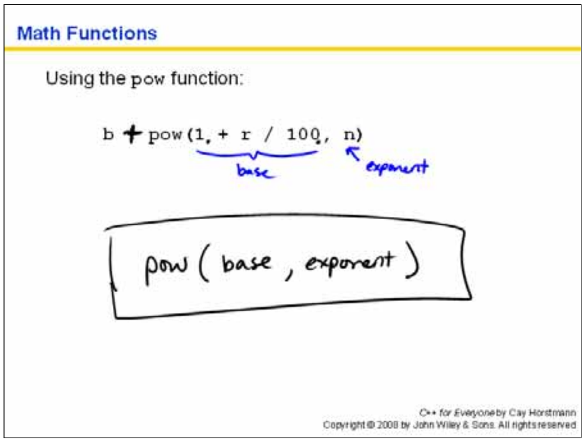
- In C++, there are no symbols for powers and roots. To compute them, you must call *functions*.
- The C++ library defines many mathematical functions such as `sqrt` (square root) and `pow` (raising to a power).
- To use the functions in this library, called the `cmath` library, you must place the line:

  ```
  #include <cmath>
  ```

  at the top of your program file.
- It is also necessary to include

  ```
  using namespace std;
  ```

  at the top of your program file.

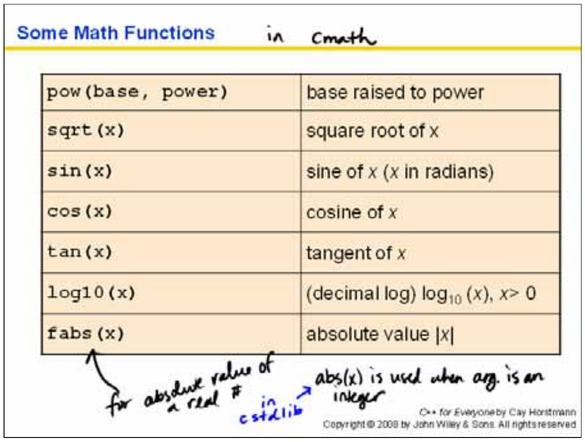


Math Functions

Using the `pow` function:

$$\texttt{b + pow(}\underbrace{\texttt{1. + r / 100.}}_{\text{base}}\texttt{, n)}$$

$$\boxed{\text{pow (base , exponent)}}$$



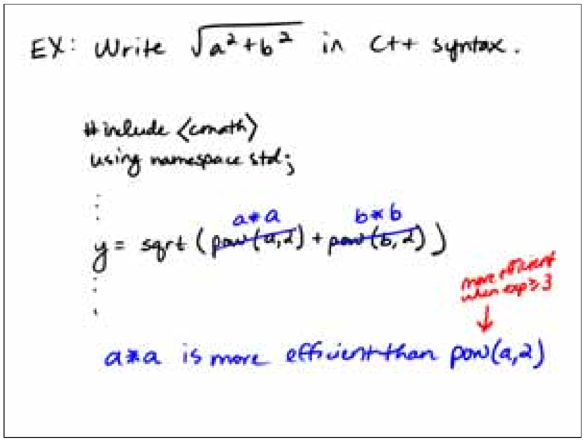
Some Math Functions in cmath

`pow(base, power)`	base raised to power
`sqrt(x)`	square root of x
`sin(x)`	sine of *x* (*x* in radians)
`cos(x)`	cosine of *x*
`tan(x)`	tangent of *x*
`log10(x)`	(decimal log) $\log_{10}(x)$, $x > 0$
`fabs(x)`	absolute value $\lvert x\rvert$

for absolute value of a real #

abs(x) is used when arg. is an integer

in cstdlib



EX: Write $\sqrt{a^2+b^2}$ in C++ syntax.

#include <cmath>
using namespace std;

⋮

y = sqrt (~~pow(a,2)~~ a*a + ~~pow(b,2)~~ b*b)

⋮

more efficient when exp>3

a*a is more efficient than pow(a,2)

Converting Floating-Point Numbers to Integers

- When a floating-point value is assigned to an integer variable, the fractional part is discarded:

```
double price = 2.55;
int dollars = price;
      // Sets dollars to 2
```

- You probably want to round to the *nearest* integer. To round a positive floating-point value to the nearest integer, add 0.5 and then convert to an integer:

```
int dollars = price + 0.5;
         // Rounds to the nearest integer
```

Formatting Output

Formatting Output

Which do you think the user prefers to see on her gas bill:

```
Price per liter: $1.22
```

or

```
Price per liter: $1.21997
```

how do we make sure only 2 decimal places print out? and rounded

Formatting Output

- When you print an amount in dollars and cents, you usually want it to be *rounded* to two significant digits.
- You learned how to actually round off and store a value but, for output, we want to round off *only* for display.
- A ***manipulator*** is something that is sent to `cout` to specify how values should be formatted.
- To use manipulators, you must include the `iomanip` header in your program:

```
#include <iomanip>
```

and

```
using namespace std;
```

is also needed

Formatting Output (section 8.3 has more details)

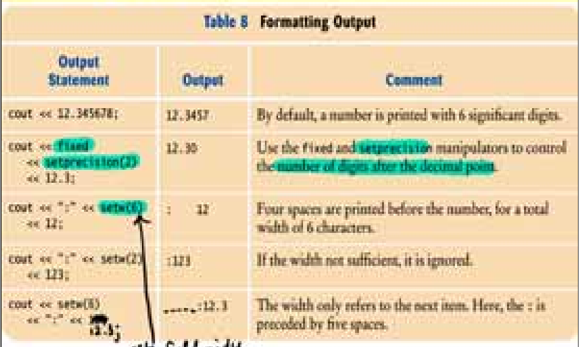


Table 8 Formatting Output

Output Statement	Output	Comment
`cout << 12.345678;`	12.3457	By default, a number is printed with 6 significant digits.
`cout << fixed << setprecision(2) << 12.3;`	12.30	Use the `fixed` and `setprecision` manipulators to control the number of digits after the decimal point.
`cout << ":" << setw(6) << 12;`	`:    12`	Four spaces are printed before the number, for a total width of 6 characters.
`cout << ":" << setw(2) << 123;`	:123	If the width not sufficient, it is ignored.
`cout << setw(6) << ":" << 12.3;`	␣␣␣␣␣:12.3	The width only refers to the next item. Here, the : is preceded by five spaces.

gets field width for next item only

Formatting Output

- You can combine manipulators and values to be displayed into a single statement:

EX:

```
cout << fixed << setprecision(2)
   << "Price per liter: "
   << price_per_liter << endl;
```