Lecture 5: Introducing Data Types!

Variables, Arithmetic and Input/Output

Announcements and reminders

Submissions:

- HW 2 (pseudocode) -- due Saturday at 6 PM

Back up your work!

- **Use cloud storage; <u>always save your work in more than one place</u>!**
- Dropbox, Google Drive, Github, aggressively long email chain to yourself…

MY COMPUTER DOESN'T WORK! THE HARD DRIVE CRASHED! WHAT DO I DO?!
DID YOU BACK UP?
WHY? IS IT GONNA EXPLODE?
STEINFELD ©96

Last time on *Starting Computing…*

Your first program!

```
#include <iostream>
using namespace std;
int main()
{
        cout << “Hello world!” << endl;
        return 0;
}
```

Other notes:

- Every program includes one or more headers for required services (e.g., input/output)
- Every program using standard services needs the namespace std directive
- **Every** program has a `main` function
- The statements of a function are **always** enclosed in braces (curly brackets { })
- This line is the “meat and cheese” of your program.
  To do other things, replace it with other codes!
- Every statement ends in a semicolon ;
  (So compiler knows where lines begin/end)

Last time on *Starting Computing…*

Output statements -- printing multiple items

Can display more than one thing by chaining or *streaming* multiple copies of the << operator into the same statement.

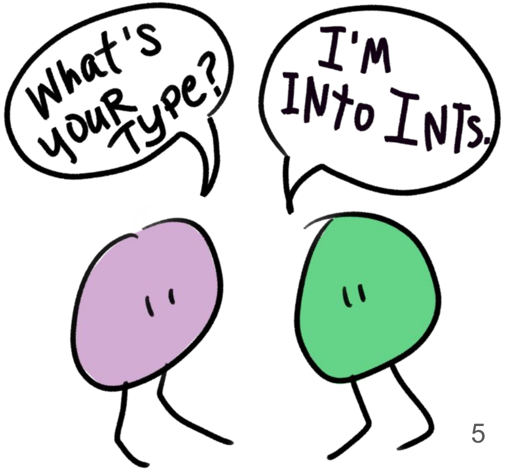
Example: S’pose we want to print the message “A big number is [37*41]” to the screen, where [37*41] is replaced by us actually computing the product of 37 and 41.

```
cout << “A big number is ” << 37*41 << endl;
```

Chapter 2: Fundamental Data Types

Chapter topics:

- Variables
- Arithmetic
- Input and output
- Problem solving: first do it by hand
- Strings



Arithmetic Operations

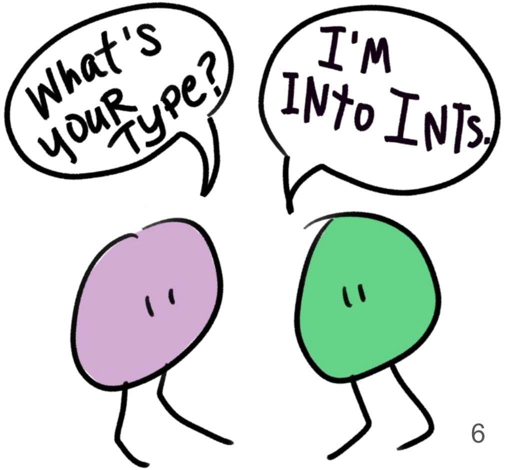
C++ has same arithmetic operations as a calculator

- Multiplication: a * b
- Division: a / b
- Addition: a + b
- Subtraction: a - b

Operator precedence

Just like in regular math, * and / have higher precedence than + and -

Example: What will 4 + 6 / 2 yield?



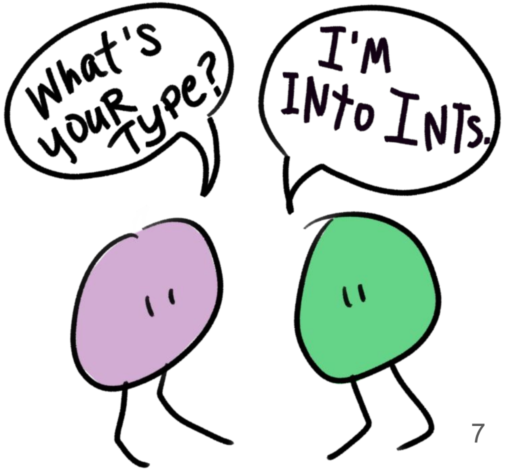
Increment and Decrement

Changing a variable by adding or subtracting 1 is so common that there is a special shorthand:

- Increment (add 1): `count++;    // add 1 to count`
- Decrement (subtract 1): `count--;    // subtract 1 from count`

Example: What is the value of `count` after the code below?

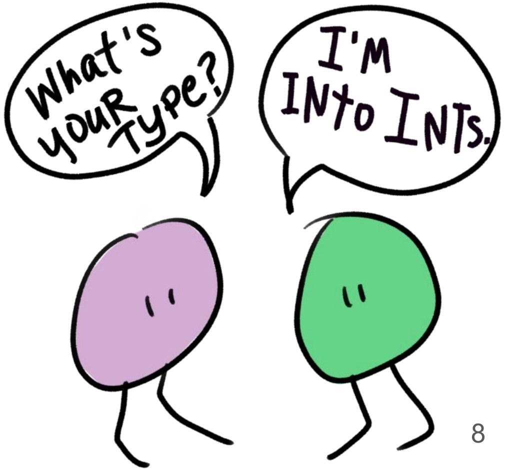
```
int count = 3;

count--;

count = count + 2;

count++;
```



Integer division and Remainder

Definition: The % operator is called the **<u>modulus operator</u>** (or **modulo**, or **mod**). It computes the remainder of an integer division (like 10/4 has a remainder of 2).

- **Example:** 10/4 has a remainder of 2, so 10%4 = 2
- Has nothing to do with the % symbol on a calculator



Integer division and Remainder

Example: You want to determine the value in dollars and cents stored in a piggy bank.

```
int pennies = 1729;

int dollars =

int cents =
```

Integer division and Remainder

Example: You want to determine the value in dollars and cents stored in a piggy bank.

```
int pennies = 1729;

int dollars = pennies / 100;    // sets dollars to 17

int cents = pennies % 100;      // sets cents to 29
```

- You obtain the dollars through an integer division by 100 (discards the remainder)
- You obtain the cents (the remainder) using the modulus (%) operator

Integer division and Remainder

More Examples: What are the results from each of the following divisions or mods?

____________________ 27 / 4

____________________ 27.0 / 4

____________________ 27 % 4

____________________ -27 % 4

____________________ 27 % 10

____________________ 27 % 2

Converting floating-point numbers to integers

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

Example:

```
double price = 2.55;

int dollars = price;

cout << dollars << endl;     ← what is printed to the screen?
```

Question: We probably want to round the decimal number to the ***nearest*** integer. How can we modify the above code to do this?

Converting floating-point numbers to integers

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

Example:

```
double price = 2.55;

int dollars = price;

cout << dollars << endl;
```

← what is printed to the screen?

Question: We probably want to round the decimal number to the ***nearest*** integer. How can we modify the above code to do this?

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

Powers and Roots

What if we want to calculate some compound interest using the following equation?

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

The part inside the parentheses is easy:

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

But what about raising to the *n* power?

Powers and Roots

What if we want to calculate some compound interest using the following equation?

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

The part inside the parentheses is easy:

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

But what about raising to the *n* power?

→ There are no native C++ functions for powers and roots.

→ So load the C++ library with handy functions like `sqrt(...)` (square root) and `pow(...)` (raising to a power, and lots, lots more!

→ Need to include this library at the top of our program: `#include <cmath>`
and if you didn't already: `using namespace std;`

Powers and Roots

Example: The power function pow(...) is has two **arguments** (the inputs):

1) The base
2) The exponent

→ pow(base, exponent)

So for our money example, we would have:

```
double balance = b * pow( 1+r / 100, n)
```

Powers and Roots

Example: The power function pow(...) is has two **arguments** (the inputs):

1) The base
2) The exponent

→ pow(base, exponent)

So for our money example, we would have:

```
double balance = b * pow( 1+r / 100, n)
```

THINK FAST! What did we need to *include* at the top of our code?

Powers and Roots -- Examples (Table 5)

Mathematical expression	**C++ expression**	**Comment**
	(x + y) / 2	Parentheses required; x + y / 2 would compute x + (y / 2)
	x * y / 2	Parentheses not required; operators with same precedence are evaluated from left to right.
	pow(1 + r / 100, n)	Need #include <cmath> at the top of our program
	sqrt(a * a + b * b)	a * a is simpler than pow(a, 2)
	(i + j + k) / 3.0	If i, j and k are integers, using a denominator of 3.0 forces floating-point division. Results in a double

More Math Function Examples

What are the results from each of the following calculations?

____________________ pow(10, 3)

____________________ sqrt(100)

____________________ abs(3 - 10)

____________________ log10(1000)

____________________ max(3, -10)

____________________ cos(3.1415926535)

____________________ tan(M_PI / 4)

Fun fact: M_PI is a constant defined in the <cmath> library

More Math Function Examples (Table 6)

Function	**Description**
sin(x)	Sine of x
cos(x)	Cosine of x
tan(x)	Tangent of x
log10(x)	Decimal log: $\log_{10}(x)$, $x > 0$
abs(x)	Absolute value $\lvert x \rvert$

Common Error -- Unintended Integer Division

If both arguments of / are integers, then the remainder is **discarded**.

Example:

7 / 3 = 2, ***not*** 2.5

but… 7.0 / 3.0, 7 / 3.0, and 7.0 / 3 all yield 2.5

Example: Will this work? Why or why not?

```
int score1 = 2
int score2 = 3
int score3 = 5

double average = (score1 + score2 + score3) / 3;

cout << “Your average score is ” << average << endl;
```

Common Error -- Unintended Integer Division

Example: ACK!! This doesn’t work!! **How can we fix it?**

```
int score1 = 2
int score2 = 3
int score3 = 5

double average = (score1 + score2 + score3) / 3;

cout << “Your average score is ” << average << endl;
```

Common Error -- Unbalanced Parentheses

Example: Consider the expression:

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

What’s wrong with this picture?

Common Error -- Unbalanced Parentheses

Example: Consider the expression:

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

What’s wrong with this picture?

→ the parentheses are **unbalanced** - there are 3 open-parens (, but only 2 close-parens)

→ common bug in complicated expressions

Common Error -- Unbalanced Parentheses

Example: Consider the expression:

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

What’s wrong with this picture?

→ the parentheses are **<u>unbalanced</u>** - there are 3 open-parens (, but only 2 close-parens)

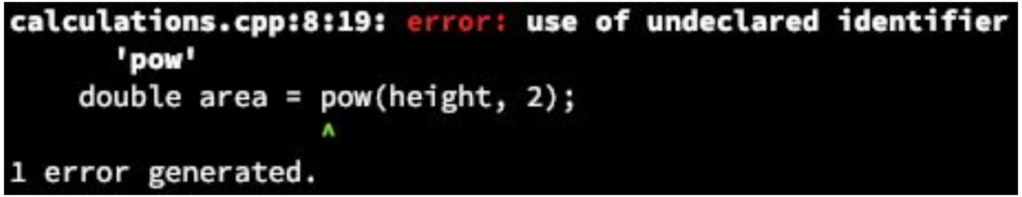
→ common bug in complicated expressions

Solution: The Muttering Method

- Count starting with 1 at the 1st parenthesis.
- Add 1 for each open-paren (left paren), and subtract 1 for each close-paren (right paren)
- If your final count is not 0, or if you ever drop to -1, then **STOP - something is wrong!**

Common Error -- Forgetting Header Files

- Every program that carries out input or output needs the <iostream> header.
- If you use mathematical functions (`sqrt`, `pow`, ...) you need to include <cmath>
- If you forget to include the appropriate header file, the compiler will complain about unfamiliar symbols like `cout` or `sqrt`



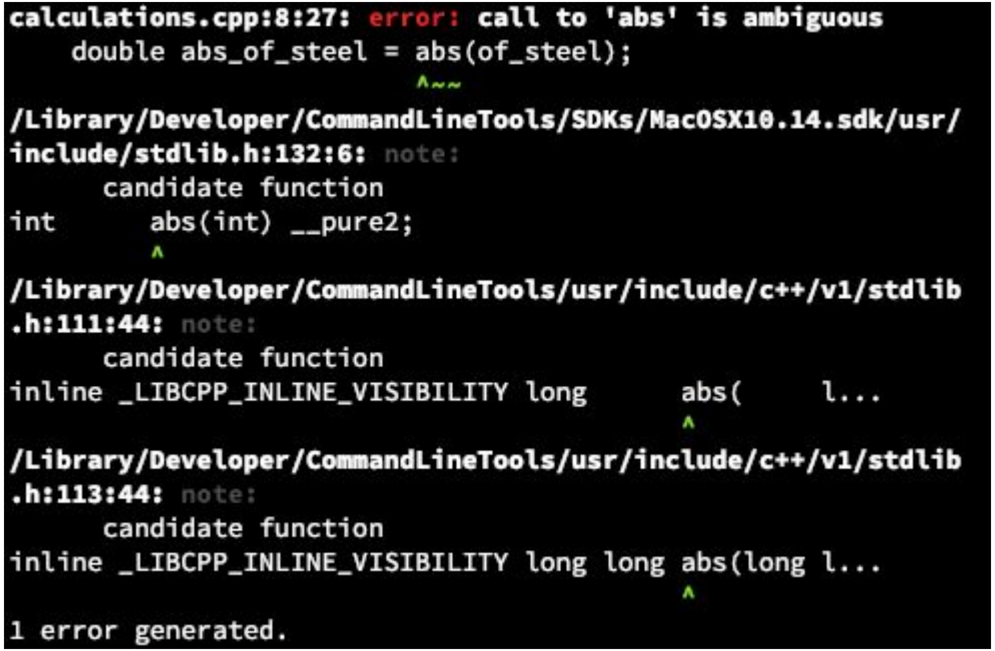
```
calculations.cpp:8:19: error: use of undeclared identifier
      'pow'
    double area = pow(height, 2);
                  ^
1 error generated.
```

- If that happens, check your header files!

Including the *Right* Header Files

- Sometimes you may not know which header files to include
- S’pose you want to compute the absolute value of an integer using the `abs` function

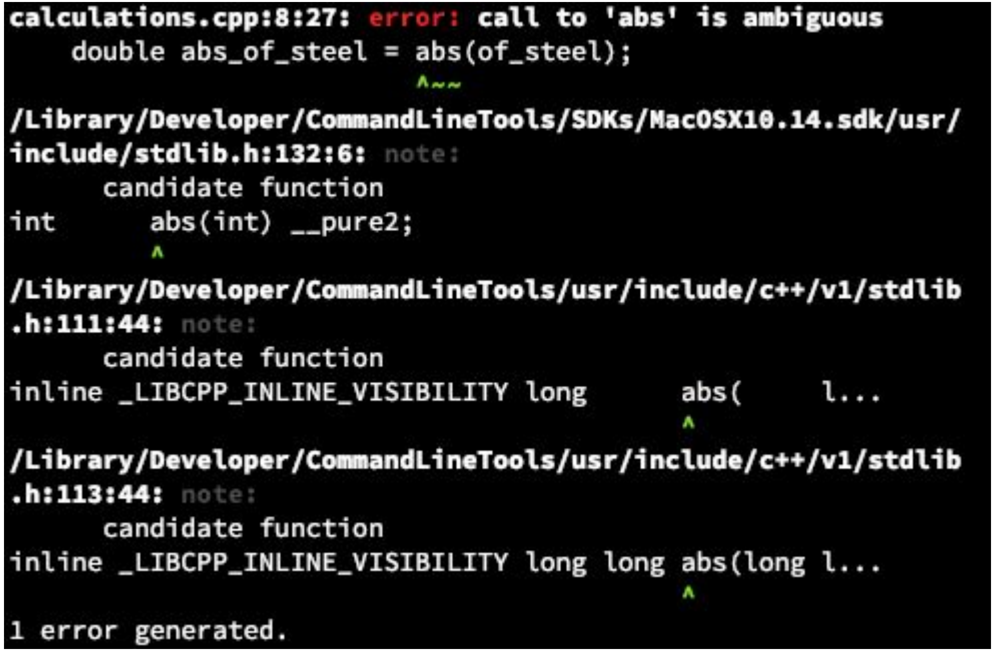
- That’s terrifying! So many error messages. What should we do?!



```
calculations.cpp:8:27: error: call to 'abs' is ambiguous
    double abs_of_steel = abs(of_steel);
                          ^~~
/Library/Developer/CommandLineTools/SDKs/MacOSX10.14.sdk/usr/
include/stdlib.h:132:6: note:
      candidate function
int      abs(int) __pure2;
         ^
/Library/Developer/CommandLineTools/usr/include/c++/v1/stdlib
.h:111:44: note:
      candidate function
inline _LIBCPP_INLINE_VISIBILITY long      abs(     l...
                                           ^
/Library/Developer/CommandLineTools/usr/include/c++/v1/stdlib
.h:113:44: note:
      candidate function
inline _LIBCPP_INLINE_VISIBILITY long long abs(long l...
                                           ^
1 error generated.
```

Including the *Right* Header Files

- Sometimes you may not know which header files to include
- S’pose you want to compute the absolute value of an integer using the abs function



```
calculations.cpp:8:27: error: call to 'abs' is ambiguous
    double abs_of_steel = abs(of_steel);
                          ^~~
/Library/Developer/CommandLineTools/SDKs/MacOSX10.14.sdk/usr/
include/stdlib.h:132:6: note:
      candidate function
int      abs(int) __pure2;
         ^
/Library/Developer/CommandLineTools/usr/include/c++/v1/stdlib
.h:111:44: note:
      candidate function
inline _LIBCPP_INLINE_VISIBILITY long      abs(     l...
                                           ^
/Library/Developer/CommandLineTools/usr/include/c++/v1/stdlib
.h:113:44: note:
      candidate function
inline _LIBCPP_INLINE_VISIBILITY long long abs(long l...
                                           ^
1 error generated.
```

- That’s terrifying! So many error messages. What should we do?!
- We take to the Internet!!
  and find out that abs is defined in <cstdlib> (int)

and in <cmath> (int and double)

Spaces in Expressions

- It is usually easier to read

```
x1 = (-b + sqrt(b * b - 4 * a * c)) / (2 * a);
```

than

```
x1 = (-b+sqrt(b*b-4*a*c))/(2*a);
```

← somucheasiertoreadwithspacesright?

- To make your codes easier for others to read, we put spaces around all operators:

```
+ - * / % =
```

Spaces in Expressions

- It is usually easier to read

```
x1 = (-b + sqrt(b * b - 4 * a * c)) / (2 * a);
```

than

```
x1 = (-b+sqrt(b*b-4*a*c))/(2*a);
```

← somucheasiertoreadwithspacesright?

- To make your codes easier for others to read, we put spaces around all operators:

```
+ - * / % =
```

- It is customary to ***not*** put a space between function names and the parentheses:
  - Good: `sqrt(x)`
  - Bad: `sqrt (x)`

Spaces in Expressions -- Unary vs Binary Minus

- **<u>Unary minus</u>**: A minus sign - used to negate a single quantity like: `-b`

- **<u>Binary minus</u>**: A minus sign taking the difference between *two* quantities: `a - b`

- We do ***not*** put a space after a unary minus.

- Helps distinguish it from a binary one.

Casts

- Occasionally, you need to store a value into a variable of a different type, or print it in a different way
- A **cast** is a conversion from one type (e.g., `int`) to another type (e.g., `double`)

Example: How can we print or capture the exact quotient from two `int` variables?

```
int x = 25;
int y = 10;

cout << “The quotient is ” << x / y << endl;        ← what will happen here?
```

Casts

- Cast conversion syntax:

```
static_cast<newtype>(data_to_convert)
```

- Older version is discouraged, but works: `(newtype)data_to_convert`

Example: How can we print or capture the exact quotient from two `int` variables?

```
int x = 25;
int y = 10;

cout << "The quotient is " <<                                << endl;
```

Casts

- Cast conversion syntax:

```
static_cast<newtype>(data_to_convert)
```

- Older version is discouraged, but works: `(newtype)data_to_convert`

Example: How can we print or capture the exact quotient from two `int` variables?

```
int x = 25;
int y = 10;

cout << “The quotient is ” << x / static_cast<double>(y) << endl;
```

or using the deprecated old version:

```
cout << “The quotient is ” << x / (double)y << endl;
```

Casts

THINK FAST!! Which of these will **not** give the mathematically correct quotient?

```
int num = 70;
int den = 20;
```

a) `cout << “The quotient is ” << num / static_cast<double>(den) << endl;`

b) `cout << “The quotient is ” << static_cast<double>(num) / den << endl;`

c) `cout << “The quotient is ” << static_cast<double>(num/den) << endl;`

Combining Assignment and Arithmetic

… YOU CAN

Examples: How can we make the follow computations more compact?

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

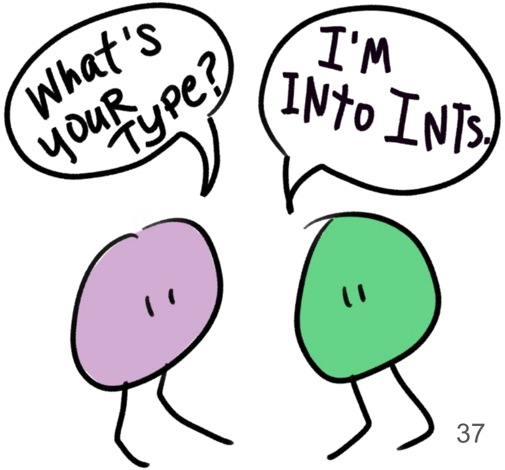
```
total = total * 2.0;
```

What just happened…?

We saw how **variables** work!

We saw how to **represent** different types of **numbers**!

We saw some **mathematical** functions and **arithmetic**!



What just happened…?

We saw how **variables** work!

- Variable naming conventions
- Assignments

We saw how to **represent** different types of **numbers**!

- Floating point (double)
- Integer (int)
- Constants (const)

We saw some **mathematical** functions and **arithmetic**!

- `+ - * / % pow() abs() sqrt()` etc...
- Need to include the right header files -- Google is your friend!