C++ Basics

Announcements

Lab 1 this week!

Homework posted Wednesday (late)

Avoid errors

To remove your program of bugs,
you should try to test your program on
a wide range of inputs

Typically it is useful to start with a small
piece of code that works and build up
rather than trying to program everything
and then debug for hours

Variables

Variables are objects in program

To use variables two things must be done:
- Declaration
- Initialization

See: uninitialized.cpp

Example if you forget to initialize:

I am 0 inches tall.

I am -1094369310 inches tall.

Variables

```
int x, y, z;
x = 2;
y = 3;
z = 4;
```

int x, y, z; ← Declaration

x = 2; y = 3; z = 4; } Initialization

Same as:

```
int x=2, y=3, z=4;
```

Variables can be declared anywhere (preferably at start)

Assignment operator

= is the assignment operator

The object to the right of the equals sign is stored into the object in the left

```
int x, y;
y = 2;
x = y+2;
```

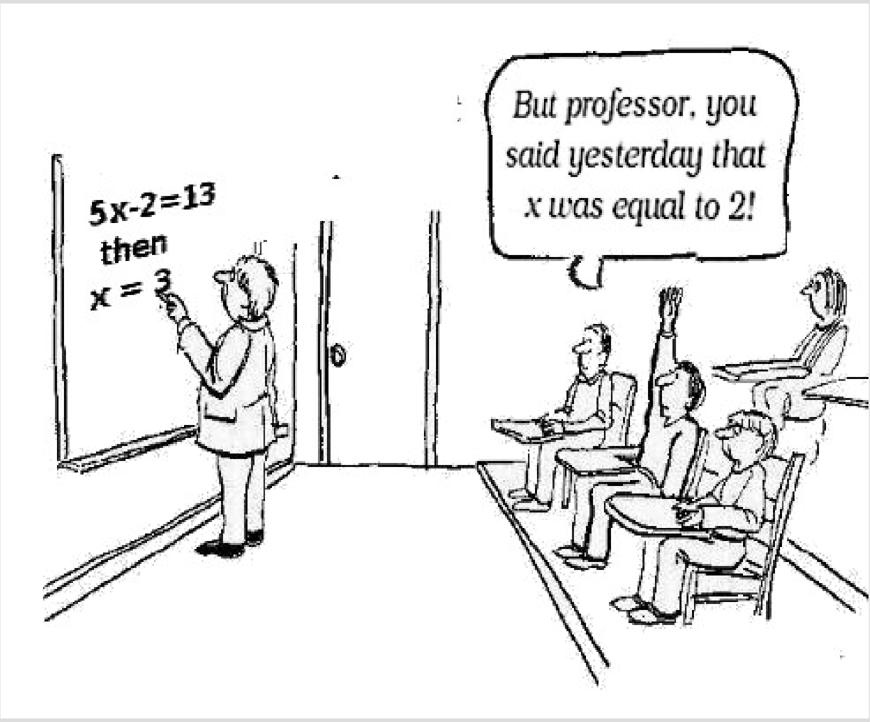
See: assignmentOp.cpp

Assignment operator

= is NOT a mathematic equals

x=3;
x=4; // computer is happy!

This does not mean 3=4



Assignment operator

To the left of = needs to be a valid object that can store the type of data on the right

```
int x;
x=2.6; // unhappy, 2.6 is not an integer

x+2 = 6; // x+2 not an object

2 = x; // 2 is a constant, cannot store x
```

Assignment operator

What does this code do?

```
int x = 2, y = 3;
y=x;
x=y;
```

What was the intention of this code?

Increment operators

What does this code do?

```
int x = 2;
x=x+1;
```

Increment operators

What does this code do?

```
int x = 2;
x=x+1;
```

Same as:

```
x+=1;
```

or

```
x++;
```

Increment operators

Two types of increment operators:

```
x++;  // increments after command
    vs
++x;  // increments before command
```

Complex assignments

The following format is general for common operations:

variable (operator)= expression
variable = variable (operator) expression

Examples:

x+=2		x = x + 2
x*=y+2		x = x * (y + 2)

Order of operations

Order of precedence (higher operations first):
-, +, ++, -- and ! (unary operators)
*, / and % (binary operators)
+ and - (binary operators)

% is remainder operator
(example later in simpleDivision.cpp)

Order of operations

Binary operators need two arguments
Examples:
2+3, 5/2 and 6%2

Unary operators require only one argument:
Examples: (see binaryVsUnaryOps.cpp)
+x, x++, !x

(! is the logical inversion operator for bool)

Identifiers

Identifiers

An identifier is the name of a variable (or object, class, method, etc.)

```
int sum;
```

type

identifier

- Case sensitive
- Must use only letters, numbers or _
- Cannot start with a number
- (Some reserved identifiers, like main)

Identifiers

Already did this in week 1!
See: RuntimeError.cpp

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

int main()
{
    int number;

    cout << "What is your lucky number?" << endl;
    cin >> number;
    cout << "I like " << 10/number << "!\n";

    return 0;
}

```

Identifiers

Which identifiers are valid?

1) james parker
2) BoByBoY
3) x3
4) 3x
5) x_________
6) _________x
7) Home.Class
8) Five%
9) x-1

Identifiers

Which identifiers are valid?

1) ~~james parker~~
2) BoByBoY
3) x3
4) ~~3x~~
5) x__________
6) __________x
7) ~~Home.Class~~
8) ~~Five%~~
9) ~~x-1~~

Identifiers

(See: float.cpp)

```
int main()
{
    float Float, fLoat, fl0at, FLOAt, FLOAT;
    Float = 1;
    fLoat = 2;
    fl0at = -3;
    FLOAT = 2;
    FLOAt = 4;
    cout << (-fLoat + floAT(fLoat*fLoat - FLOAt * Float * fl0at))/(FLOAT*Fl
    cout << (-fLoat - floAT(fLoat*fLoat - FLOAt * Float * fl0at))/(FLOAT*Fl

    return 0;
}
```

Identifiers

Variables

We (hopefully) know that if you say:

```
int x;
```

You ask the computer for a variable called x

Each variable actually has an associated type describing what information it holds (i.e. what can you put in the box, how big is it, etc.)

Fundamental Types

bool - true or false
char - (character) A letter or number
int - (integer) Whole numbers
long - (long integers) Larger whole numbers
float - Decimal numbers
double - Larger decimal numbers

See: intVSlong.cpp

int vs long?

int - Whole numbers in the approximate range:
-2.14 billion to 2.14 billions (10^9)

long - Whole numbers in the approximate range:
-9.22 quintillion to 9.22 quintillion (10^{18})

Using int is standard (unless you really need more space, for example scientific computing)

float vs double?

float vs double?

float is now pretty much obsolete.

double takes twice as much space in the computer and 1) has wider range and 2) is more precise

Bottom line: use double (unless for a joke)

float and double

Both stored in scientific notation

```
double x = 2858291;
```

Computer's perspective:

x = 2.858291e6

or

$x = 2.858291 * 10^6$

Welcome to binary

Decimal:	Binary:
1/2 = 0.5	0.1
1/3 = 0.3333333	0.010101010101
1/10 = 0.1	0.0001100110011

double is often just an approximation!

Numerical analysis

Field of study for (reducing) computer error

See: subtractionError.cpp

Can happen frequently when solving system of linear equations

bool

You can use integers to represent bool also.

false = 0
true = anything else

(You probably won't need to do this)

int or double?

If you are counting something (money),
use int

If you are dealing with abstract concepts (physics),
use double

int doesn't make “rounding” mistakes

Primitive type hierarchy

bool < int < long < float < double

If multiple primitive types are mixed together in a statement, it will convert to the largest type present

Otherwise it will not convert type

Primitive type hierarchy

```
int x;
double y;

x+y
```

Converted to double

```
int x;
int y;

x/y
```

Not converted (still int)

Integer division

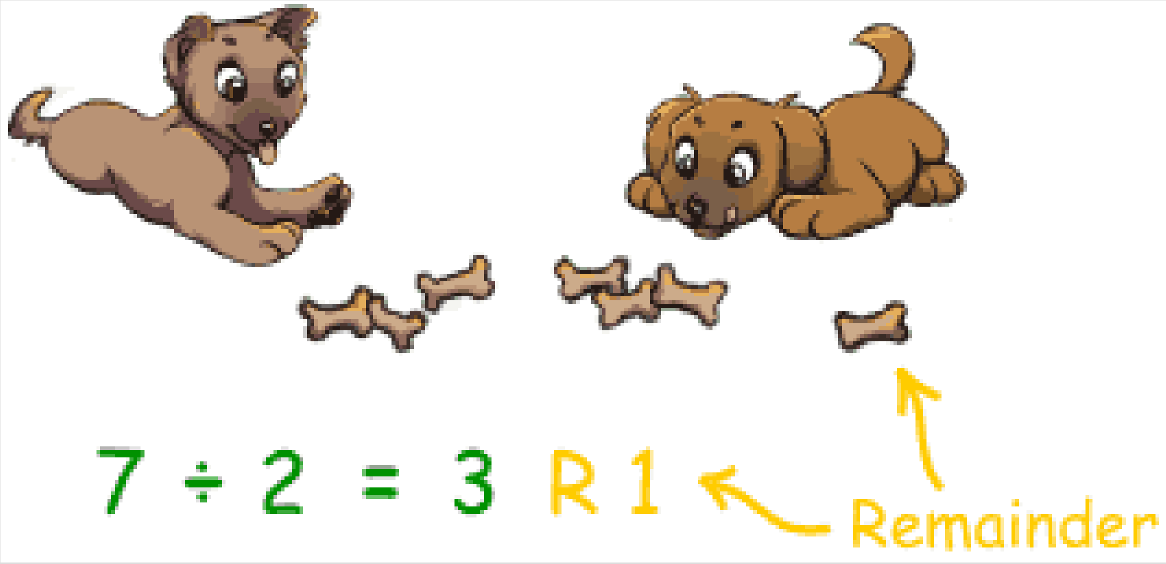
See: simpleDivision.cpp

Can be fixed by making one a double:

```
1/2.0
```

or

```
static_cast<double>(1)/2
```



Constants

You can also make a “constant” by adding const before the type

This will only let you set the value once

```
const double myPI = 3.14;
myPI = 7.23;  // unhappy computer!
```

Functions

Functions allow you to reuse pieces of code (either your own or someone else's)

Every function has a return type, specifically the type of object returned

sqrt(2) returns a double, as the number will probably have a fractional part

The “2” is an argument to the sqrt function

Functions

Functions can return void, to imply they return nothing (you should not use this in an assignment operation)

The return type is found right before the functions name/identifier.

int main() { ... means main returns an int type, which is why we always write return 0 and not return 'a' (there is no char main())

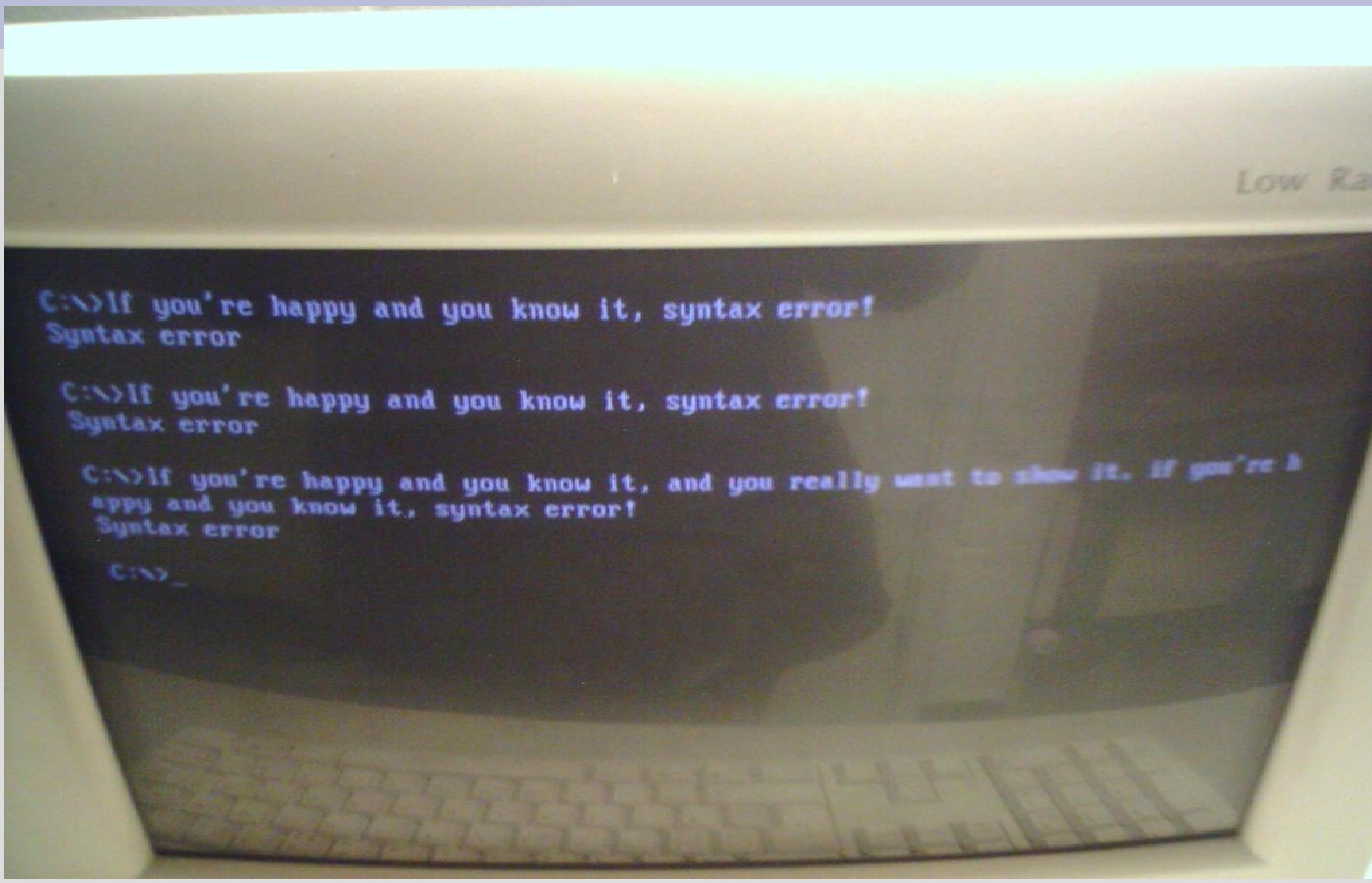
Functions

A wide range of math functions are inside <cmath> (get it by #include <cmath>; at top)

We can use these functions to compute Snell's Law for refraction angle

(See: math.cpp)

Input and output



Strings and input

char can only hold a single letter/number, but one way to hold multiple is a string

```
string str;
cin >> str;
```

The above will only pull one word, to get all words (until enter key) use:

`getline(cin, str);` (See: stringInput.cpp)

More Output

When showing doubles with cout, you can change how they are shown

For example, to show a number as dollars and cents, you would type (before cout):

```
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
```

More Output

There are two ways to get output to move down a line: endl and “\n”

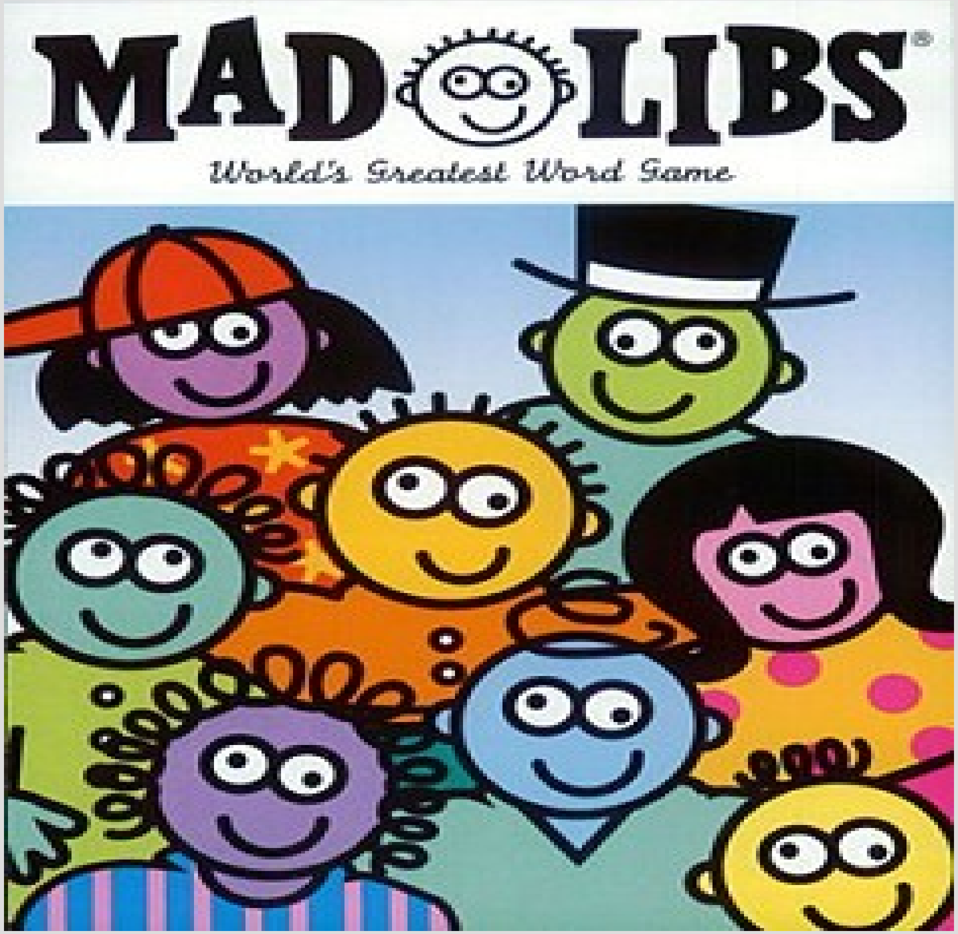
```
cout << endl;
```

... is the same as...

```
cout << “\n”
```

I will use both when coding

Madlibs



(see: madlibs.cpp)

bool

bool - either true or false

You have the common math comparisons:

- > (greater than), e.g. 7 > 2.5 is true
- == (equals), e.g. 5 == 4 is false
- <= (less than or eq), e.g. 1 <= 1 is true

If you cout this, “false” will be 0
and “true” will be 1 (anything non-zero is T)

Double trouble!

(See: doubleCompare.cpp)

Double trouble!

When comparing doubles, you should use check to see if relative error is small:

fabs((x-y)/x) < 10E-10
(double has about 16 digits of accuracy so you could go to 10E-15 if you want)

For comparing Strings, use: (0 if same)
string1.compare(string2)