C/C++ Programming
Session 2

Dr Jim Lupo
LSU/CCT HPC Enablement
jalupo@cct.lsu.edu

Review

Each line represents a concept we touched on last session:

```
1.  int main ( ... )
2.  ;
3.  { ... }
4.  int
5.  float
6.  cout
7.  cin
8.  <<
9.  >>
10.  “blah blah”
```

Did you succeed with this?

```
1 #include <iostream>
2 using namespace std;
3 int main () {
4    cout << "Hello World!\n";
5    return (0);
6 }
```

Continuing The Basics

- Additional syntax elements
  - comments
- More about variables:
  - types
  - creation
  - memory usage

Comments

Comments are text that are meant for humans and are ignored by the compiler:

`//` .. **Ignore everything to the end-of-line.**
`/*` .. **Begin a comment, typically multiline.**
`*/` .. **End a comment.**

```
/*
    Transform cubic feet of water to gallons
*/
gallons = cuft / 8.0; // just divide by 8
```

Alternate Forms

```
//
// Transform cubic feet of water to gallons
//
gallons = cuft / 8.0; /* just divide by 8 */
```

```
/**************************************
**
** Transform cubic feet of water to gallons
**
**************************************/
/* 1 cu ft water equals 8 gallons */
gallons = cuft / 8.0;
```

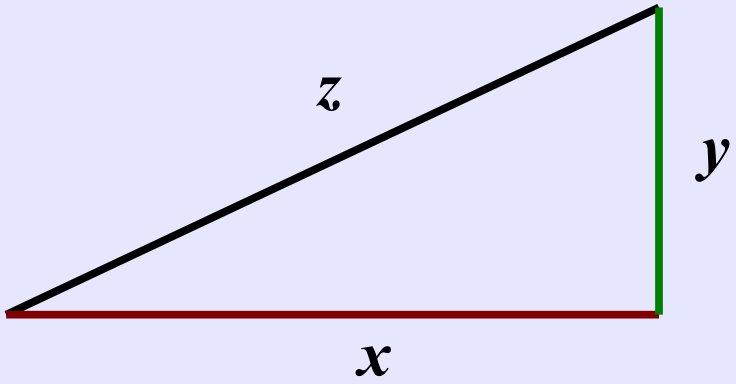
It helps the code reader (i.e. yourself in the future??) to keep comments open and consistent.

What is a Variable?

From algebra, you may (or may not) recall that a variable represents a specific quantity of some kind. For instance:

$$x^2 + y^2 = z^2$$

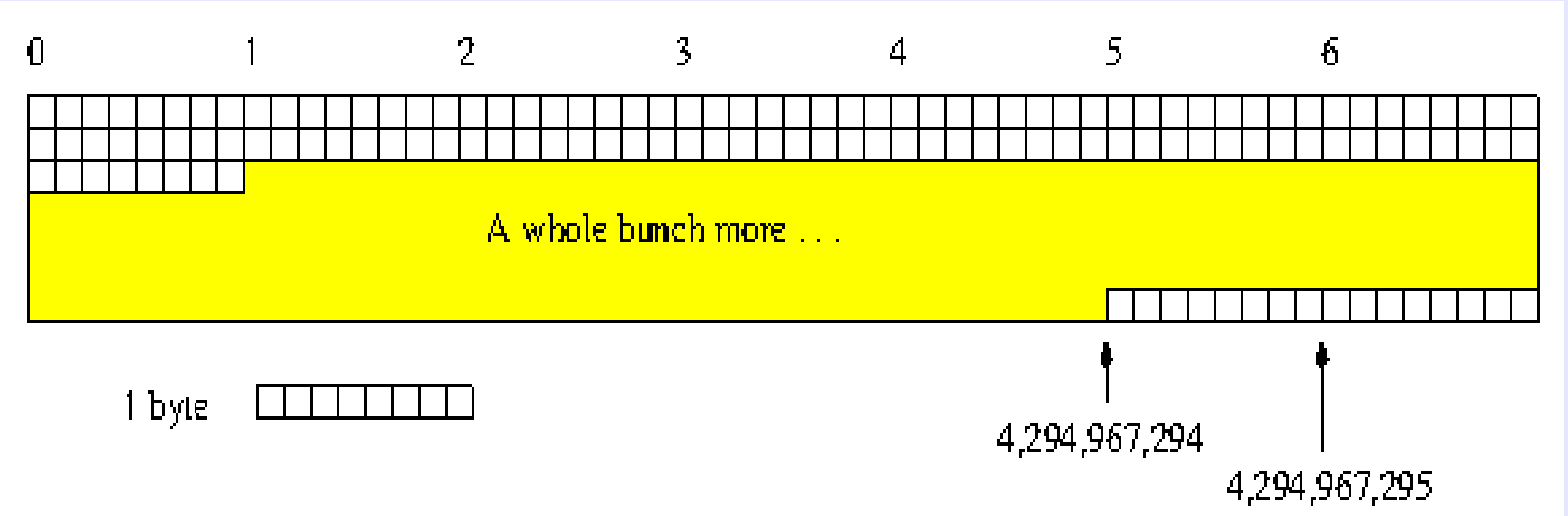
It gives the relationship between the lengths of the sides of a right triangle via the Pythagorean Theorem.



Program Variables

- Variable – named symbol to which a value is attached.
  - Identifies chunk of memory to store value in.
  - Critical characteristics include:
    - size in memory
    - precision supported

Computer Memory



This is what 4GiB of memory looks like. Each byte can be located by its sequence number or *address*.

Address of first byte: 0
Address of last byte: 4,294,967,295

GB = gigabyte = 10^9 = 1,000,000,000
GiB = gibibyte = 2^{30} = 1,073,741,824

Integer Values

Everything stored in a computer must be reduced to some expression that can be expressed in terms of 1s and 0s – *binary data*.

Integers are whole numbers: 42 (b'00101010), 7 (b'00000111)

Different memory units accommodate different value ranges:

`bit` . . . . . **single bit: 0 – 1**
`nibble` . . **4-bit unit: 0 – 15, or -8 – 7**
`byte` . . . . **8-bit unit: 0 – 255, -128 – 127**
`word` . . . . **2/4/8/16 byte unit. 65,536 values to several million billion.**

Available Integer Types

`char`	1 byte (signed)
`unsigned char`	1 byte (unsigned)
`short`	2 bytes (signed)
`unsigned short`	2 bytes (unsigned)
`int`	4 bytes (signed)
`unsigned int`	4 bytes (unsigned)
`long`	4/8 bytes (signed)*
`unsigned long`	4/8 bytes (unsigned)*
`long long`	8/16 bytes (signed)*
`unsigned long long`	8/16 bytes (unsigned)*

Compiler dependent!

CHAR Values

The CHAR type is most frequently used to manipulate individual letters, or characters. Single quotes are used to specify a character value:

```
char ans;
ans = 'y'; // or 121, or 0x79
ans = 'N'; /* or 78, or 0x4E */
```

Control characters can be specified using a "\" escape:

```
tab ..... '\t' (or 9)
Ctrl-C .. '\c' (or 3)
```

Declare vs. Define

Creating a variable involves associating a name with a location in memory, so we should be crystal clear about the terminology used:

`declare` . . **specify the data type of an object.**
`define` . . . **specify the data type and memory for an object.**
`address` . . **the location in memory.**

Everything we do in the next couple of sessions will involve definition – we will revisit declaration later.

Examples

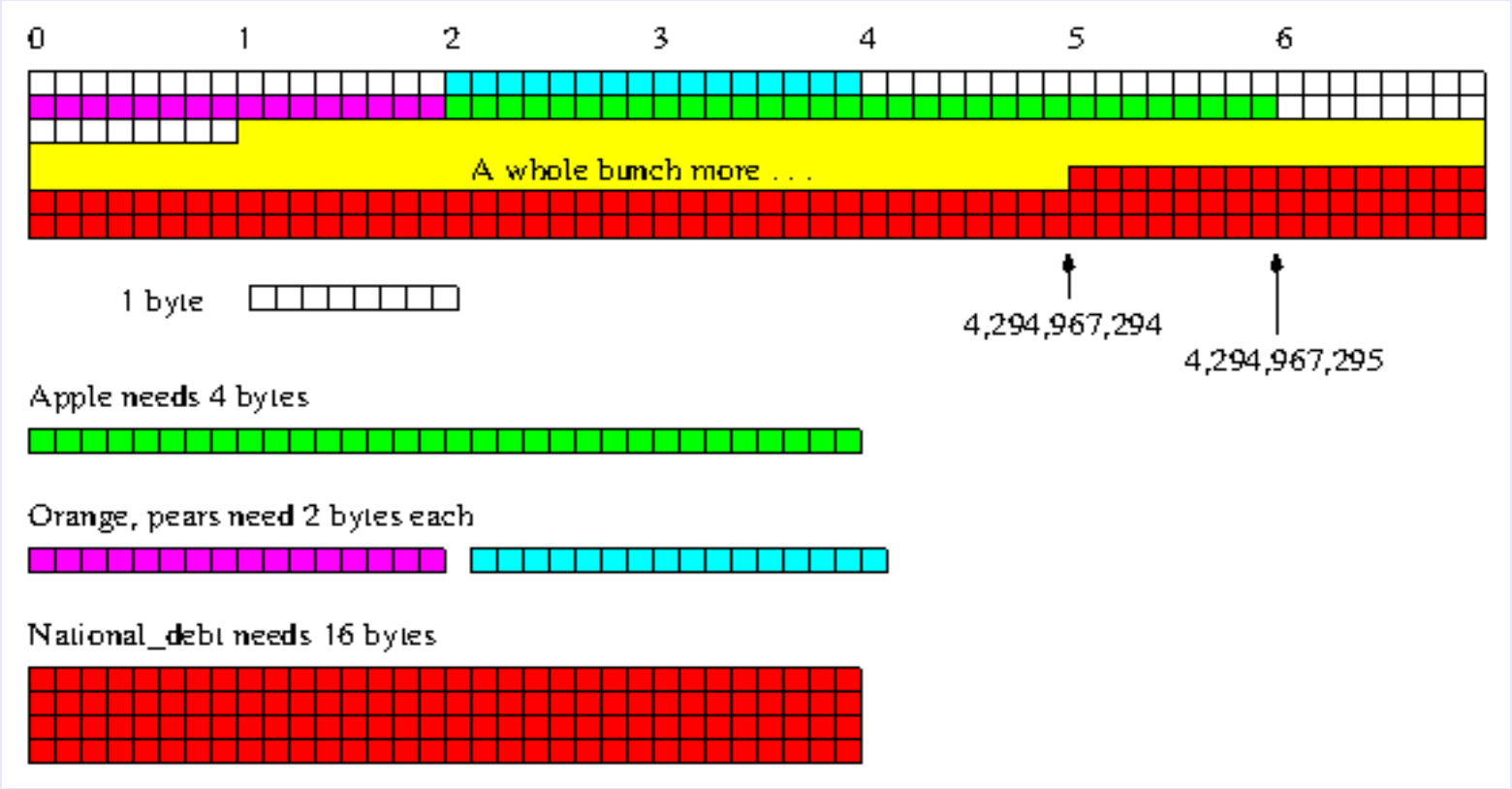
The example program gave us one example, defining the variable "gallon" as an integer value. Using the other types is straight-forward:

```
int apples;
short oranges, pears;
long long national_debt;
```

The statement syntax should be obvious:

```
type varname [, varname [, ...]];
```

Allocation in Memory



Floating Point Numbers

Floating point numbers are used whenever fractional values are required. For example:

$$1/137 = 7.29927007299 \times 10^{-3}$$

To store such numbers, they have to be *encoded.* As such, they can not store exact values unless they can be expressed as a sum of various powers of 2. Exactly how this is done depends on the type of floating point number you use.

Available Floating Point Types

`float`	4 bytes
`double`	8 bytes
`long double`	10 bytes (IEEE754)

Compilers on different machine architectures may also provide more exotic floating point types, like 2-byte or 16-byte floats.

Encoding Floating Point?

32 bit:

Sign	Exponent	Mantissa
1-bit	8-bits (signed)	23-bits (unsigned)

64 bit:

Sign	Exponent	Mantissa
1-bit	11-bits (signed)	52-bits (unsigned)

Range and Precision

How big or small (magnitude) is your data?
How well must you approximate the answer?

Type	Range	Digits of Precision
`float`	3.4×10^{-38} to 3.4×10^{38}	About 7
`double`	1.07×10^{-308} to 1.07×10^{308}	About 15
`long double`	1.2×10^{-4932} to 1.2×10^{4932}	About 19

Numerical Analysis is the field that studies how algorithms behave and guides choice of precision.

Binary Fractions

Binary decimal: summed powers of 2 – any integer

Binary fraction: summed inverse powers of 2 ??

0.1 is less than 1/8 but greater than 1/16:

0.1 ≈ 1/16 + 1/32 + 1/256 + 1/512 + 1/4096 + 1/8192 + …

0.1 (decimal) = .0001100110011001 0011 . . . (binary)

What we have here is a rational decimal fraction (1/10) but an irrational binary fraction.

Exercise

```
1.   #include <iostream>
2.   #include <iomanip>
3.   using namespace std;
4.   #define PI 3.14159265358979323846264338327950288
5.   #define str_helper(x) #x
6.   #define str(x) str_helper(x)
7.   int main ( void ) {
8.     float pi4;
9.     double pi8;
10.    pi4 = PI;
11.    pi8 = PI;
12.    cout << "pi  = " << str(PI) << endl;
13.    cout << "pi4 = " << setprecision(20) << pi4 << endl;
14.    cout << "pi8 = " << setprecision(20) << pi8 << endl;
15.    return ( 0 );
16.  }
```

Cats and Dogs

`bool` ... a type that accepts only true (not zero) or false (0). It is most often implement with byte storage.

`const` .. a modifier of any type indicating the value is fixed. The compiler or runtime will throw an error if anything attempts to change the value. May help compiler optimize the code.

```
bool result;
const double pi;
```

Automatic Conversion

There is some compiler magic going on that could trip up the inattentive. C/C++ tends to treat explicit floating constants as double precision unless told otherwise. This example isn't too surprising as C converts what it thinks is a double to a float in one case, and directly assigns a double in the second:

```
float pi4;
double pi8;
pi4 = 3.14159265358979323846264338327950288;
pi8 = 3.14159265358979323846264338327950288;
```

You can specify a constant's type:

- `n.mmmF` . . **n.mmm is treated as a float.**
- `n.mmmD` . . **n.mmm is treated as a double.**
- `n.mmmL` . . **n.mmm is treated as a long double.**

Machine ε (Epsilon)

For a given machine and variable type, machine epsilon is defined as the smallest number, ε, such that:

$$1 - \varepsilon \neq 1$$

This captures the accuracy and precision issues of floating point numbers.

We can write an example program to illustrate the finite precision of floats that also makes use of several of the other features we've talked about.

Machine ε Calculation

```
1.   #include <iostream>
2.   #include <iomanip>
3.   using namespace std;
4.   int main ( void ) {
5.      float epsilon;
6.      epsilon = 1.0F;
7.      cout << setprecision(20) << 1.0F - epsilon/2.0F
8.           << " : " << epsilon << endl;
9.      epsilon = epsilon / 2.0F;
10.     ...
11.     ... repeat 7-9 till no change shown by cout
12.  }
```

Note: "/" is the division operator, "=" is the assignment operator, and we are forcing all numbers and operations to be floats.

How many times must lines 7-9 be repeated before output change stops?

Summary

Variables

- Types: Integer, Floating Point, Logical
- Storage: units, address, define, declare
- Precision: approximate numbers
- Match data needs to variable type

Questions?