Data Types and Mathematical Expressions

Lecture 3

Outline

1. Data Types
2. Mixing Types
3. Operators
4. Variable Assignments
5. Integer Division
6. Operator Precedence
7. Complex Expressions
8. The `cmath` Library

Data Types: Review

- Data stored in memory is in bits (i.e. 0/1)
- A program uses a **data type** to tell C++
  - how much memory is required
  - how to interpret the bits

Data Types: Numbers

`int`

- Integer (whole numbers)
- 4 bytes of memory
- Range: -2^{31} to ($2^{31} - 1$)
- Examples: 0, 15, -10042, 21792, 1, -1

`double`

- Continuous values (15 digit precision)
- 8 bytes of memory
- Range: 10^{-308} to 10^{308}, +/-

Data Types: Alphanumeric

`char`

- Single character or symbol
- 1 byte of memory
- Always put in single quotes
- Examples: ‘a’, ‘C’, ‘3’, ‘.’, ‘$’

`string`

- A sequence of characters/numbers/symbols
- Always put in double quotes
- Examples: “Hello World”, “475!”, “eh?”

Strings

The `string` type is actually a C++ *class*

- Others we've discussed are *primitive* types

To use string variables, you need to include the string library at the top of your program (above/below `#include <iostream>`)

```
#include <string>
```

Data Types: Boolean

`bool`

- Boolean valued (only true/false)
- At least 1 byte of memory

Mixing Types (1)

In general, you can not assign a value of one type to a variable of another type

- But, there are many exceptions with many caveats

Rule of thumb: don't mix types except when necessary, and always be careful when you do

Mixing Types (2)

When assigning a double value to an integer, the fractional part will be discarded

- `int sum = 1.99; // sum will be 1, not 1.99, not 2`
- Same when assigning from a double variable

Strings and characters don't mix in either direction (depends upon the compiler)

- `string name = 'A'; // compiler error`
- `char letter = "a"; // compiler error`

Mixing Types (3)

Characters and integers are inter-changeable using ASCII character codes

- http://asciitable.com
- Examples:
  - `char letter = 33; // letter will be '!'`
  - `int letter = 'A'; // letter will be 65`

Mathematical Operators

Used with numeric types (`int`, `double`)

Addition	`+`	`total = part1 + part2;`
Subtraction	`-`	`left_over = total - used;`
Multiplication	`*`	`force = mass * acceleration;`
Division	`/`	`avg_weight = total_weight / num_items;`

- When both operands are of type `int` the result is also of type `int`
- When one or both operands are of type `double`, the result is also of type `double`

Assignment Statements (1)

- Notice in all the previous examples the math statements look like: VARIABLE = FORMULA;

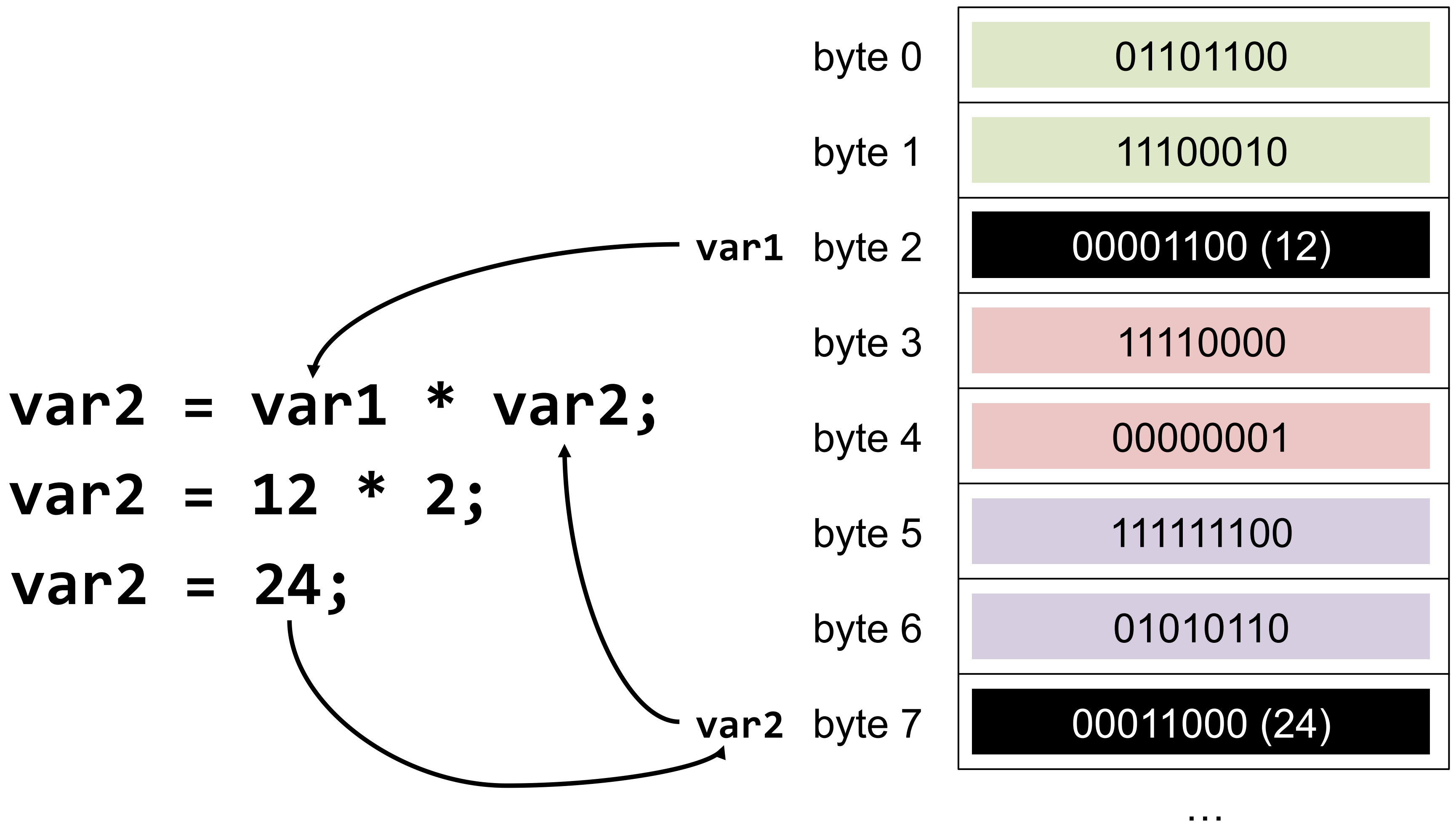
- This is because they are NOT formulas!
  - In other words, they are NOT statements of fact like in normal mathematical equations

- Every "math" statement in C++ is used to calculate a one time result when that line executes and then the "equation" is no longer remembered
  - Sequential execution!

Assignment Statements (2)

- The result of one of these one-time math calculations can be stored in a variable
- The variable name must go on the left side of the expression
- Example: **`total_inches = yards * 36;`**
  - When this statement is executed (and ONLY then), C++ plugs in the current value of the **`yards`** variable, multiplies by 36, and updates the value of **`total_inches`** to be the result

Assignment Statements (3)



```
var2 = var1 * var2;
var2 = 12 * 2;
var2 = 24;
```

	byte 0	01101100
	byte 1	11100010
var1	byte 2	00001100 (12)
	byte 3	11110000
	byte 4	00000001
	byte 5	111111100
	byte 6	01010110
var2	byte 7	00011000 (24)

…

Common Mistake (1)

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

int main()
{
    int input_value;
    int squared_value;

    squared_value = input_value * input_value;

    cout << "Enter the value: ";
    cin >> input_value;

    cout << input_value << " squared is ";
    cout << squared_value << endl;

    return 0;
}
```

Uninitialized Variable!!!

Common Mistake (2)

- The previous program fails because the programmer forgot about **sequential execution**
- The mathetmatical expression comes before the **`input_value`** variable is initialized (given a value)
  - Before the **`cin`** in this case
- So, C++ tries to execute the math statement before it has a [non-garbage] value for **`input_value`**
- To fix this mistake, move the mathematical expression after **`input_value`** has been initialized (but before you print out the result!)

Corrected!

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

int main()
{
    int input_value;
    int squared_value;

    squared_value = input_value * input_value;

    cout << "Enter the value: ";
    cin >> input_value;

    cout << input_value << " squared is ";
    cout << squared_value << endl;

    return 0;
}
```

Integer Division

- When dividing two integers, the result is an integer
  - The same rules for converting a `double` to an `int` are used (fractional value is thrown away)

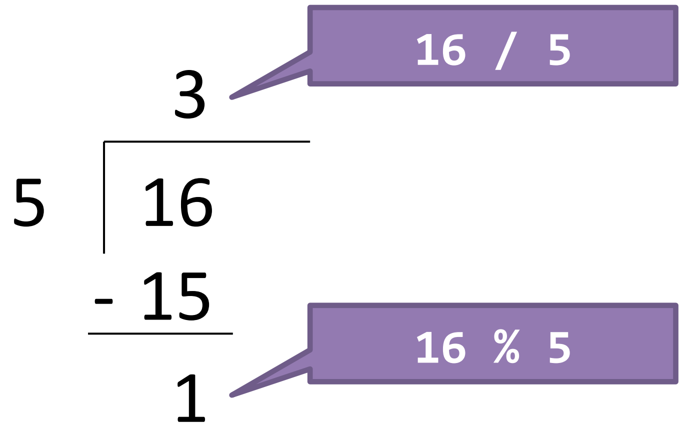
```
int answer = 7 / 2;  // answer = 3
```

- The remainder of an integer division can be accessed with the % (mod, modulus) operator

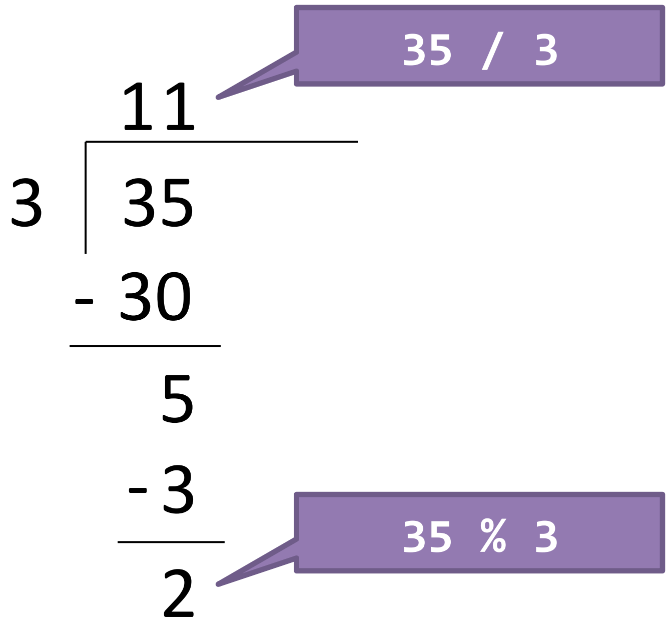
```
int remainder = 7 % 2;  // remainder = 1
```

Long Division Review

16 divided by 5:



35 divided by 3:



Operator Precedence

Evaluated First

Evaluate First

`( )`	Parentheses
`* / %`	Multiplication, Division, Modulus
`+ -`	Addition, Subtraction
`=`	Assignment

Multiple operators at the same level will be evaluated left-to-right (in the code).

Complex Expressions

Many operations can be combined in a single expression

- Use parentheses to specify order of evaluation
- Otherwise, default precedence rules are followed
- In general, use parentheses to be sure it is right

Examples

```
double ans = (b*b) – 4*a*c; // b² – 4ac
int result = x*(y + z);    // x(y+z)
```

Exercise

Write a C++ program that reads exactly three integers from the user, calculates the average of the three numbers, and prints out the average.

Answer

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

int main()
{
     int x,y,z;
     double average;

     cout << "Enter three integers:" << endl;
     cin >> x;
     cin >> y;
     cin >> z;

     average = (x + y + z) / 3.0;

     cout << "Average is " << average << endl;

     return 0;
}
```

The ".0" after 3 is necessary to get a double result!

The **`cmath`** Library

- C++ also has additional libraries that contain *functions* for doing more complex calculations
  - Square root, power, exponent, sine, cosine, tangent, etc.

- To use them, include the **`cmath`** library by putting the following at the top of your program

```
#include <cmath>
```

Useful Functions

- Square Root
  - Syntax: `RESULT = sqrt( VALUE );`
  - `double v1 = sqrt( 289.0 );`
- Power Function
  - Syntax: RESULT = pow(VALUE, POWER);
  - `double c = pow( 5, 3 ); // c = 125 (5^3)`
- Note: parentheses are required

Example

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

int main()
{
      double input;
      double squared;
      double square_root;

      cout << "Enter a value: ";
      cin >> input;

      squared = pow( input, 2 );
      square_root = sqrt( input );

      cout << endl;
      cout << input << "^2=" << squared << endl;
      cout << input << "^(1/2)=" << square_root << endl;

      return 0;
}
```

Exercise

Write a program that reads two values (x and y) from the user, calculates x^y, and prints the answer

Answer

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

int main()
{
	double x;
	double y;
	double ans;

	cout << "Enter x: ";
	cin >> x;
	cout << "Enter y: ";
	cin >> y;

	ans = pow( x, y );

	cout << endl;
	cout << x << "^" << y << "=" << ans << endl;

	return 0;
}
```

Wrap Up

- Mathematical statements in C++ are NOT like "normal" math formulas
  - They are used only once to calculate a new value, when the statement is executed in sequential order

- Operator precedence is used just like in your calculator, but it's always best to use parentheses for complex expressions anyway

- When dividing two `int` values, the result is an `int` (use long division and throw away the remainder)