CSCI 1380

JANUARY 30, 2016

ADMINISTRATIVIA

QUIZZES

...will be taken at the **end** (instead of at the start) of lectures.

LATE WORK

Any Labs or Exercises completed before the midterm will receive credit.

For credit on late submissions, you must email the TA (and cc me) with a bulleted list of the items by their repl.it name (e.g., Lab1d: Hello World) **on the day of the following quarter exam**.

LATE WORK (CONT.)

Reminder: Labs/Exercises are worth 13.5 points (out of 100) towards your grade.
The purpose of the activities is not their point value, but the understanding gained by you of the material.

In any case, you will do best if/when you do the work on-schedule.

REVIEW

ASSIGNMENT

Another handy operator: =

This is the **assignment operator**. This tells the computer to store some data somewhere. For example:

```
int x;
x = 5;
```

ASSIGNMENT (SYNTAX INTERLUDE)

Another handy operator: =

This is the **assignment operator**. This tells the computer to store some data somewhere. For example:

```
int x = 5;
```

OPERATORS (TERMINOLOGY INTERLUDE)

Another handy operator `=`

This is the **assignment operator**. This tells the computer to store some data somewhere. For example:

```
int x;
x = 5;
```

left hand side (LHS)

right hand side (RHS)

arguments/operands

ASSIGNMENT (CONT.)

An **operand** can be any expression that evaluates to the right type of data. For example, the RHS of an assignment can be an expression:

```
int x;
x = 5 + 10;
```

So what gets saved in memory at x's location? 15

A SIMPLE PROGRAM

```
int x;
cout << “Hello! Please enter a number”;
cin >> x;
cout << “The value of x is: ” << x;
```

USING STORED DATA

Note: **variables are reusable**. Each assignment stores a new value and overwrites any prior data. For example:

```
int x;
x = 5;
cout << x;
```

```
x = 1;
cout << x;
```

```
x = x + 10;
cout << x;
```

ANOTHER SIMPLE PROGRAM

```
int x;
x = 5;
```

```
cout << “The value of x is: ” << x;
```

EXERCISE

With a partner, determine what gets printed to the console when the following code is run.

```
int x;
int y;
cout << “Hello”;
x = 1 + 4 – 3;
cout << x;
y = x – 1;
y = y + 5;
cout << y;
```

EXERCISE (FORMATTING INTERLUDE)

With a partner, determine what gets printed to the console when the following code is run.

```
int x;
int y;
cout << “Hello\n”;
x = 1 + 4 - 3;
cout << x << endl;
y = x - 1;
y = y + 5;
cout << y;
```

EXERCISE

With a partner, write C++ statements to do the following and determine what happens when they are run.

Declare an integer variable called `banana`.
Declare an integer variable called `hamster`.
Assign banana the value `29`.
Assign `hamster` the value of: `17` added to the value of `banana`.
Print out the value of `hamster`.
On a new line, print out the value of: `banana - hamster`.

DATA TYPES

The **type** assigned to a variable tells the computer how much space in memory to reserve (in response to a variable declaration) and subsequently, how it is/should be *encoded* (in retrieval).

COMPUTER MEMORY

A **bit** is the smallest piece of memory. It is short for **b**inary dig**it** (as it can have values of 0 or 1).

A **byte** is eight consecutive bits (e.g., 0000 0001).

DATA STORAGE

All data is stored as bits in main memory. The meaning of a set of bits depends on the *encoding*.

BINARY REPRESENTATION	INTEGER	CHARACTER
1000 0001	65	’A’
1000 0010	66	‘B’
1000 0011	67	‘C’

DATA TYPE: `int`

Declaration of a variable of type `int` tells the computer to reserve ~**4 bytes** of space. And when using that stored information, it cues the computer to interpret the data as an *integer*.

For example, running the following code –

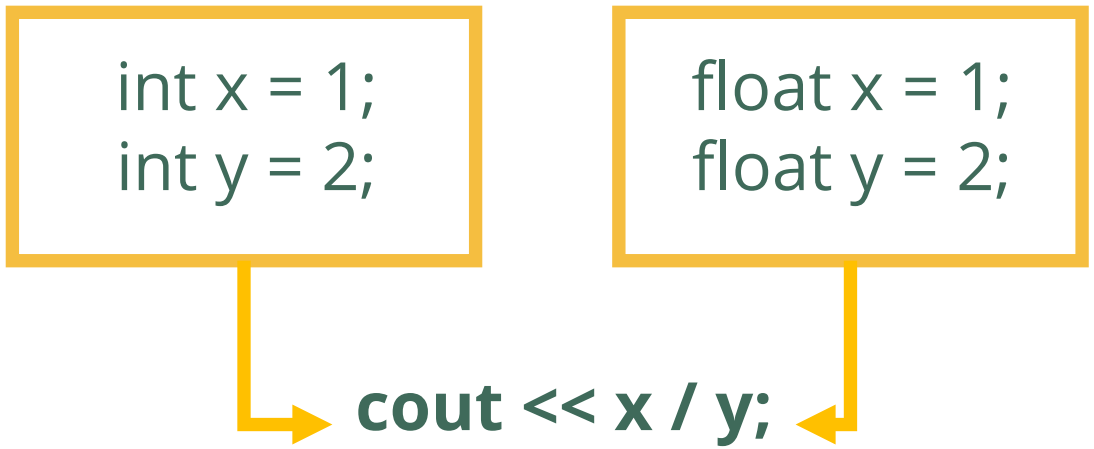
```
int x = 1.0;
cout << x;
```

– prints ‘1’ to the console.

DATA TYPE: `float`

Declaration of a `float` similarly reserves ~**4 bytes** of space. Where it differs from the `int` type is in how the computer interprets the data...

COMPARISON: `int` vs. `float`

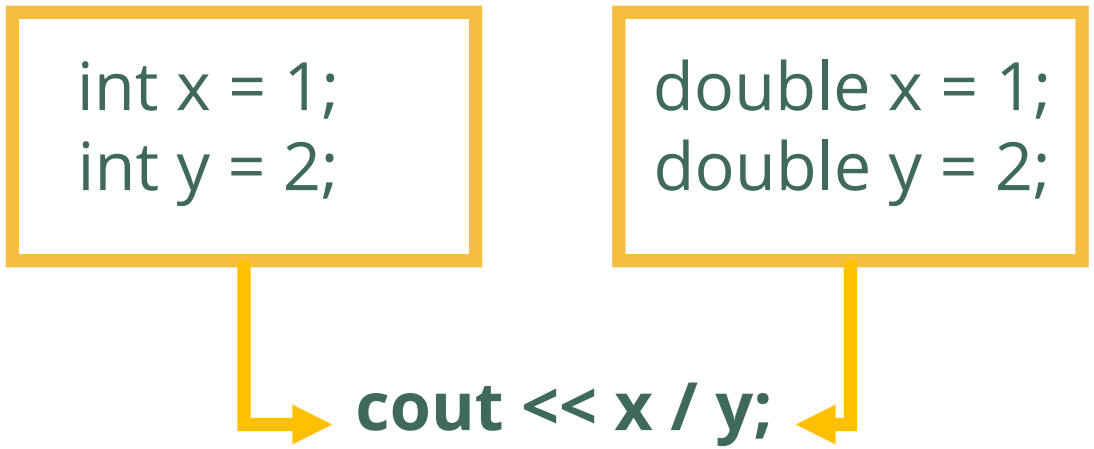


DATA TYPE: `double`

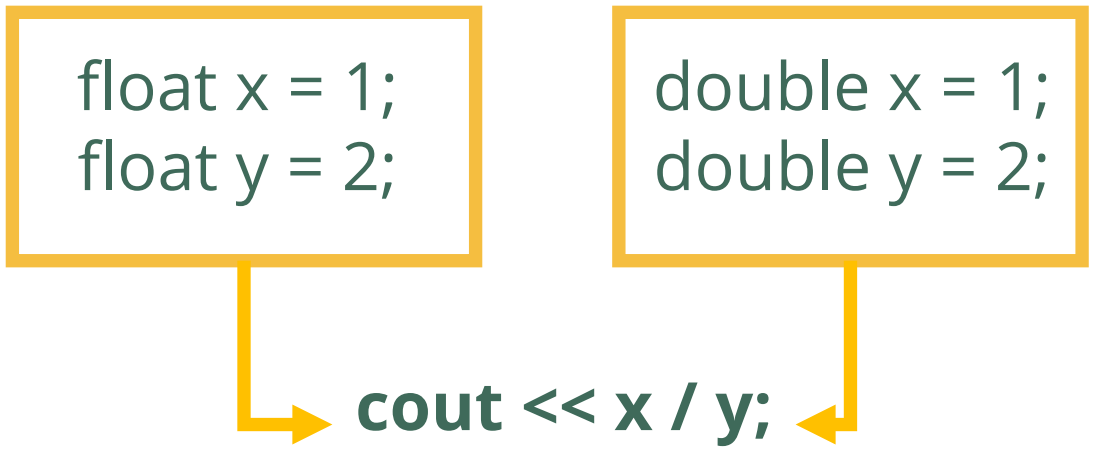
Declaration of a `double` reserves ~**8 bytes** of space.

It similarly differs from the `int` type again in how the computer interprets the data, and it differs from the `float` type in how much precision (“double” that of a float) it offers.

COMPARISON: `int` vs. `double`



COMPARISON: `float` vs. `double`



DATA TYPE: `char`

Declaration of a variable of type `char` tells the computer to reserve **1 byte** of space. This type is used for *characters*: letters, digits, and special symbols.

Note: character data must be enclosed in single quotes. For example:

```
char x = '1';
```

DATA TYPE: `string`

To store a *set of ordered characters*, we need to declare a variable of type string.
String *literals* use as many bytes as there are characters, *plus one*...
String *variables* reserve ~**32 bytes** of space.

Note: string data must be enclosed in double quotes. For example:

```
string x = “1”;
```

GROUP EXERCISE

In computer memory, what does...

...the integer, 0, look like?
...the character, '0', look like?
...the string, "0", look like?

The **type** assigned to a variable tells the computer **how much space in memory to reserve** (in response to a variable declaration) and subsequently, how it is/should be *encoded* (in retrieval).

OVERFLOW

Overflow occurs when we assign a value that is too large to be stored in the respective variable.

As a result, the variable will be assigned a value that is “wrapped around” the set of possible values.

EXAMPLE: OVERFLOW

In the code below, we are declaring an integer of type ***short int*** and ***initializing it to the largest value it can hold***. When we subsequently increment its value, we run into **overflow**.

```
short int num = 32767;
cout << num;
num = num + 1;
cout << num;
```

PARTICIPATION ACTIVITIES

WEEK 3 ACTIVITIES

Wednesday:
labs 2a–e (by 9:25am)
exercises 3a-e (by 10pm)

Thursday:
second group meeting (by 8pm)
labs 2a-f (by 10pm)

Friday:
exercises 3f-i (by 10pm)

QUIZ #3