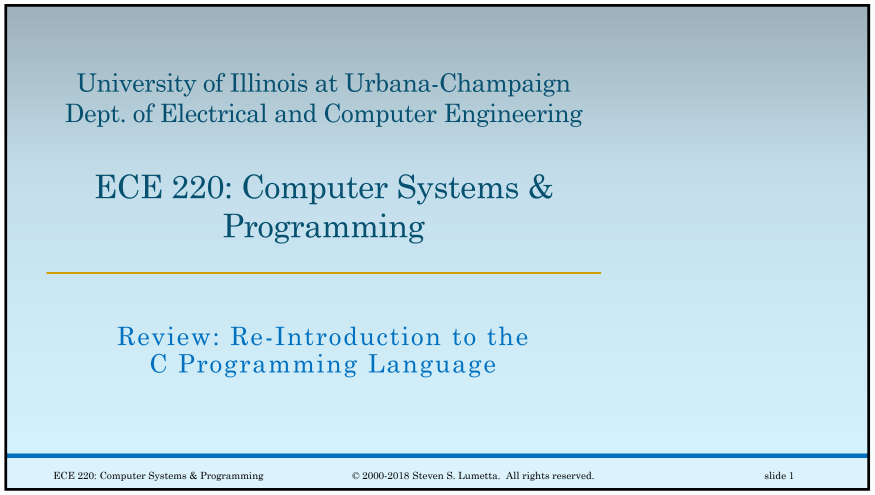
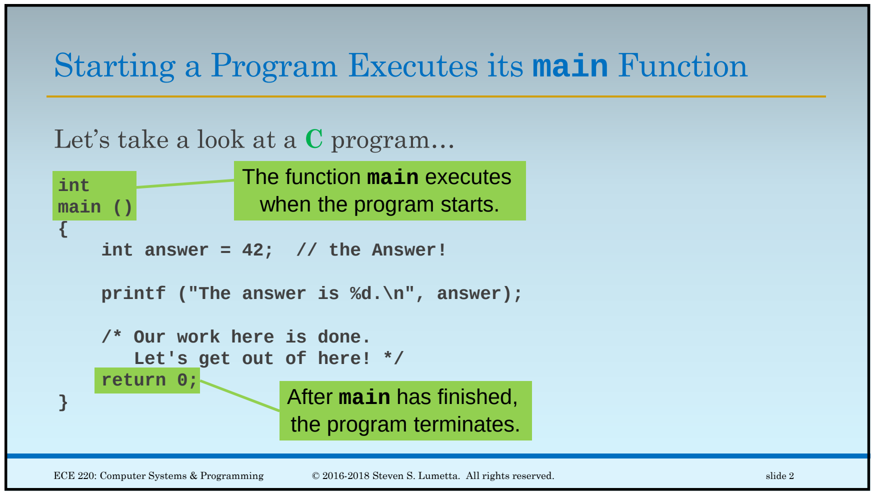


University of Illinois at Urbana-Champaign
Dept. of Electrical and Computer Engineering

ECE 220: Computer Systems & Programming

Review: Re-Introduction to the C Programming Language



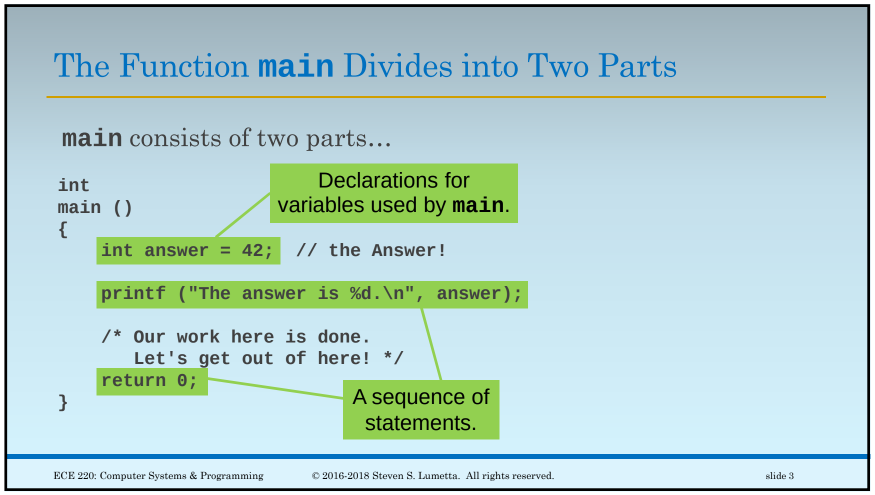
Starting a Program Executes its **`main`** Function

Let's take a look at a C program…

```
int
main ()
{
    int answer = 42;  // the Answer!

    printf ("The answer is %d.\n", answer);

    /* Our work here is done.
       Let's get out of here! */
    return 0;
}
```

The function **`main`** executes when the program starts.

After **`main`** has finished, the program terminates.



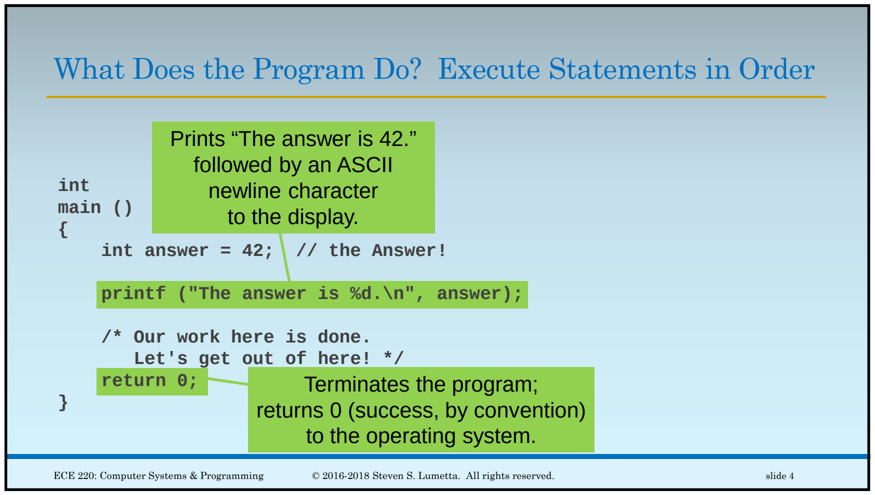
The Function **`main`** Divides into Two Parts

`main` consists of two parts…

```
int
main ()
{
    int answer = 42;  // the Answer!

    printf ("The answer is %d.\n", answer);

    /* Our work here is done.
       Let's get out of here! */
    return 0;
}
```

Declarations for variables used by **`main`**.

A sequence of statements.

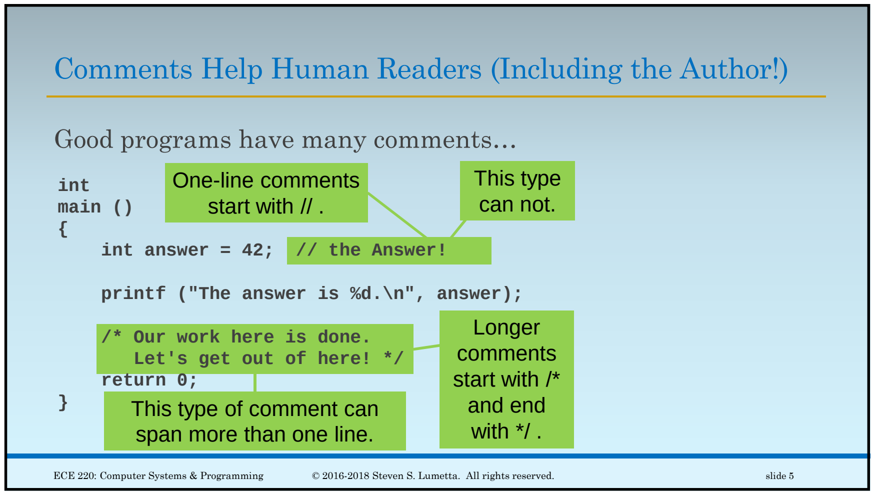


What Does the Program Do? Execute Statements in Order

```
int
main ()
{
    int answer = 42;  // the Answer!

    printf ("The answer is %d.\n", answer);

    /* Our work here is done.
       Let's get out of here! */
    return 0;
}
```

Prints "The answer is 42." followed by an ASCII newline character to the display.

Terminates the program; returns 0 (success, by convention) to the operating system.



Comments Help Human Readers (Including the Author!)

Good programs have many comments…

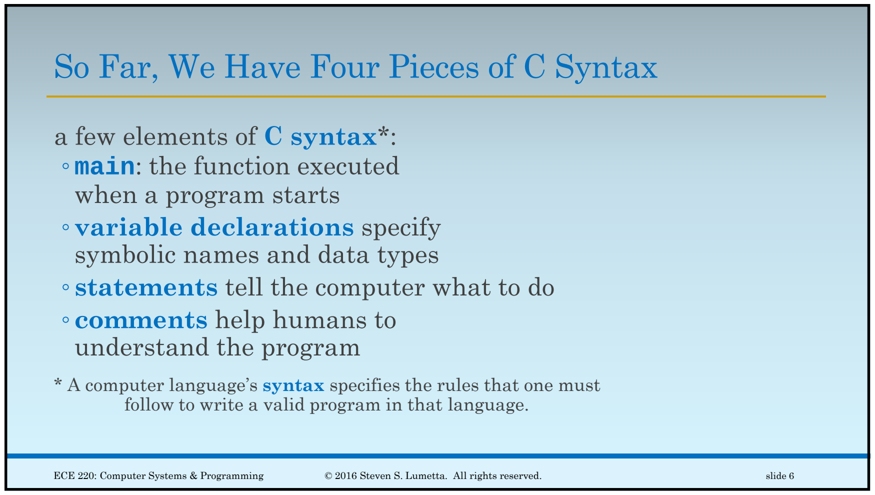
```
int
main ()
{
    int answer = 42;  // the Answer!

    printf ("The answer is %d.\n", answer);

    /* Our work here is done.
       Let's get out of here! */
    return 0;
}
```

One-line comments start with // .

This type can not.

Longer comments start with /* and end with */ .

This type of comment can span more than one line.

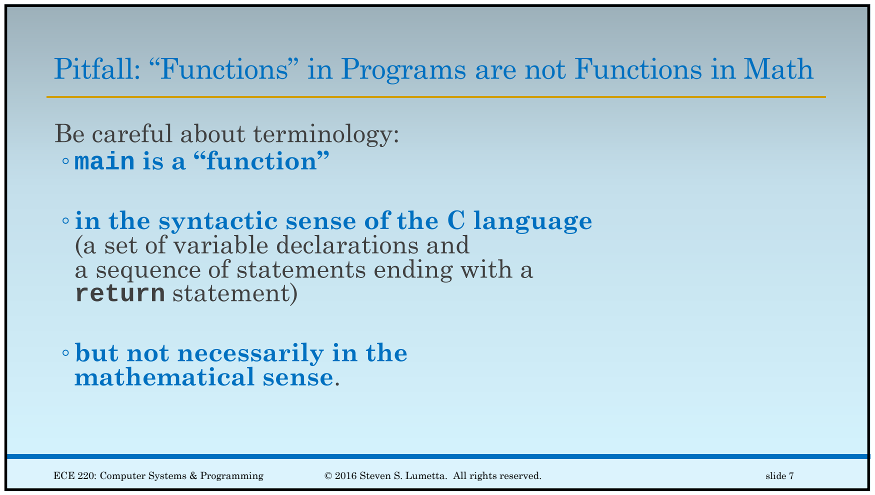


So Far, We Have Four Pieces of C Syntax

a few elements of **C syntax***:

- **`main`**: the function executed when a program starts
- **variable declarations** specify symbolic names and data types
- **statements** tell the computer what to do
- **comments** help humans to understand the program

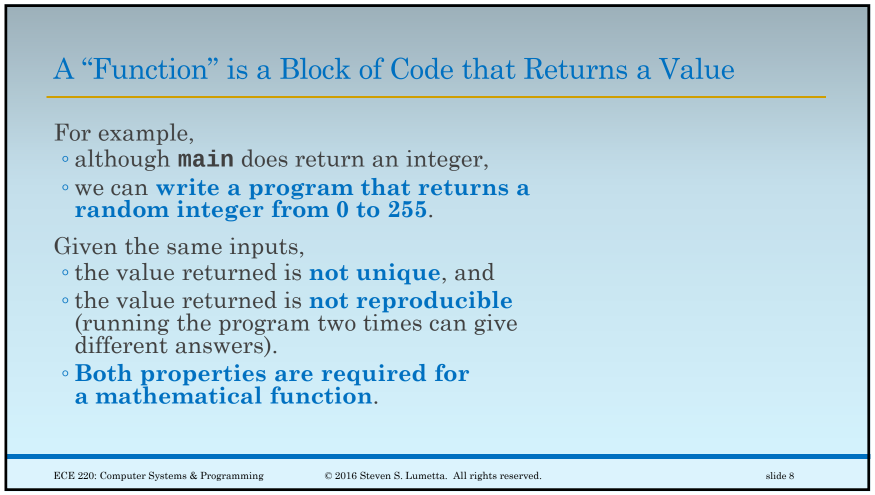
* A computer language's **syntax** specifies the rules that one must follow to write a valid program in that language.



Pitfall: "Functions" in Programs are not Functions in Math

Be careful about terminology:

- **`main` is a "function"**
- **in the syntactic sense of the C language** (a set of variable declarations and a sequence of statements ending with a `return` statement)
- **but not necessarily in the mathematical sense**.



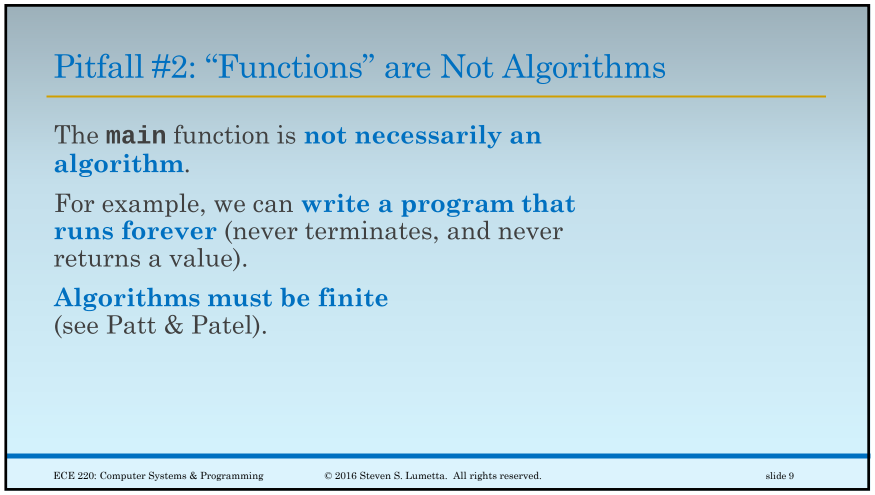
A "Function" is a Block of Code that Returns a Value

For example,

- although `main` does return an integer,
- we can **write a program that returns a random integer from 0 to 255**.

Given the same inputs,

- the value returned is **not unique**, and
- the value returned is **not reproducible** (running the program two times can give different answers).
- **Both properties are required for a mathematical function**.

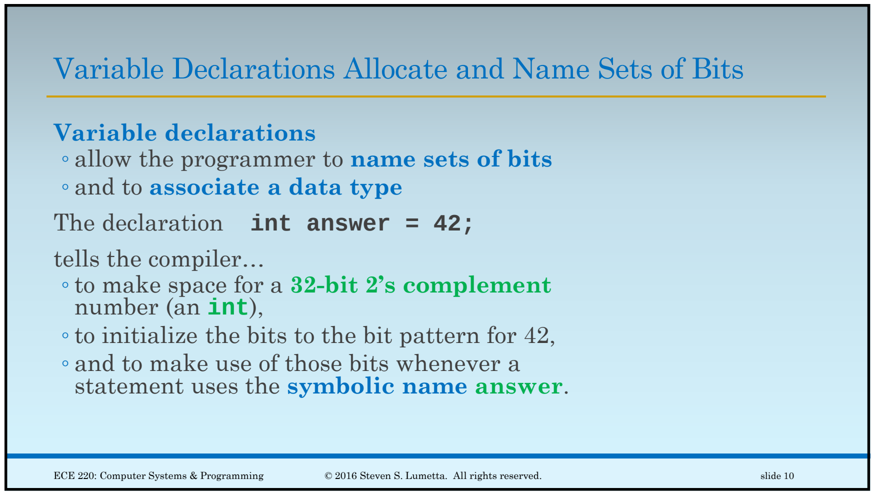


Pitfall #2: “Functions” are Not Algorithms

The **main** function is **not necessarily an algorithm**.

For example, we can **write a program that runs forever** (never terminates, and never returns a value).

Algorithms must be finite
(see Patt & Patel).

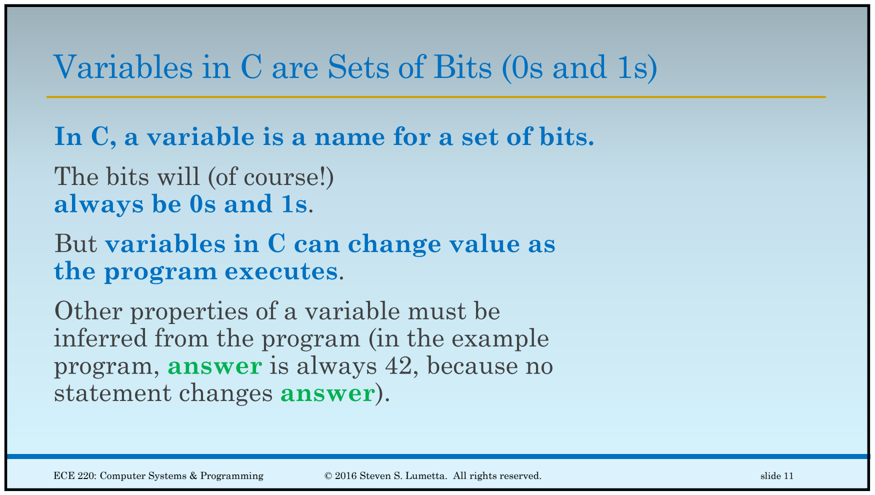


Variable Declarations Allocate and Name Sets of Bits

Variable declarations
- allow the programmer to **name sets of bits**
- and to **associate a data type**

The declaration `int answer = 42;`

tells the compiler…
- to make space for a **32-bit 2’s complement** number (an **int**),
- to initialize the bits to the bit pattern for 42,
- and to make use of those bits whenever a statement uses the **symbolic name answer**.



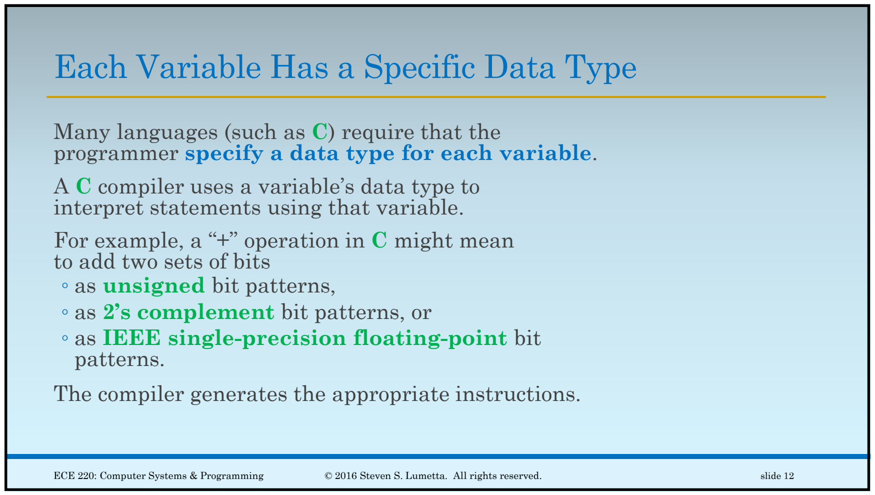
Variables in C are Sets of Bits (0s and 1s)

In C, a variable is a name for a set of bits.

The bits will (of course!)
always be 0s and 1s.

But **variables in C can change value as the program executes**.

Other properties of a variable must be inferred from the program (in the example program, **answer** is always 42, because no statement changes **answer**).



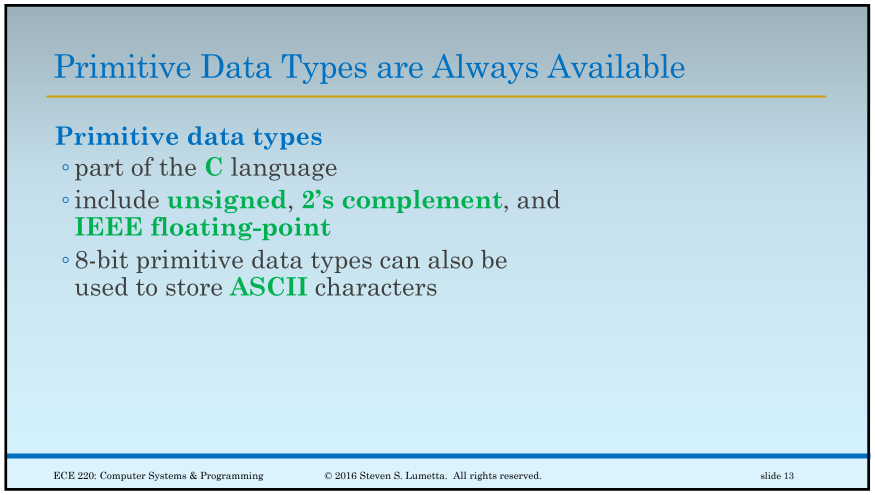
Each Variable Has a Specific Data Type

Many languages (such as **C**) require that the programmer **specify a data type for each variable**.

A **C** compiler uses a variable’s data type to interpret statements using that variable.

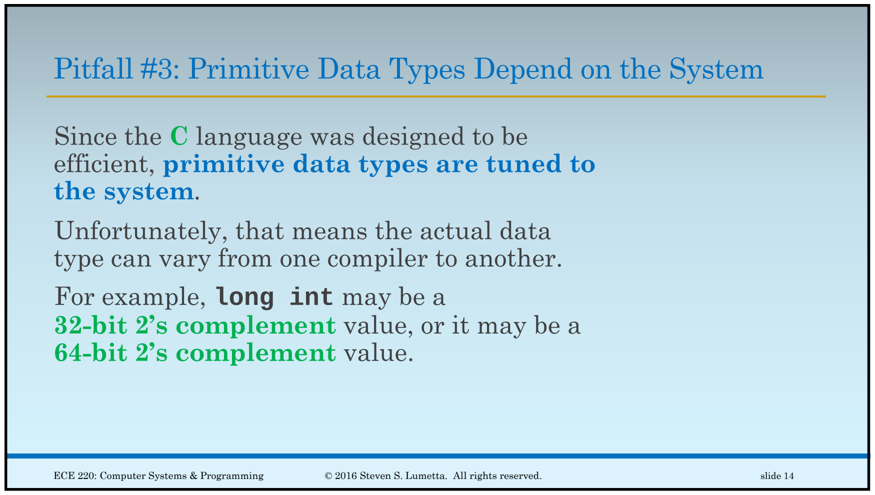
For example, a “+” operation in **C** might mean to add two sets of bits
- as **unsigned** bit patterns,
- as **2’s complement** bit patterns, or
- as **IEEE single-precision floating-point** bit patterns.

The compiler generates the appropriate instructions.



Primitive Data Types are Always Available

Primitive data types
- part of the **C** language
- include **unsigned**, **2's complement**, and **IEEE floating-point**
- 8-bit primitive data types can also be used to store **ASCII** characters

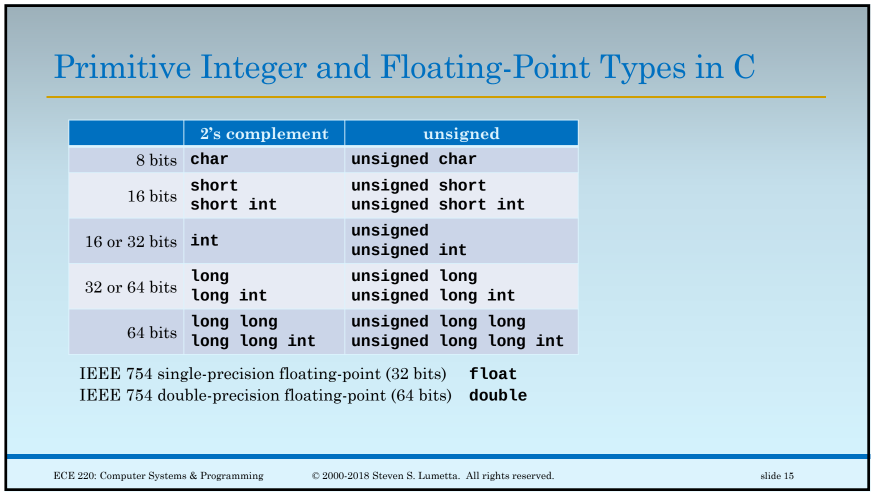


Pitfall #3: Primitive Data Types Depend on the System

Since the **C** language was designed to be efficient, **primitive data types are tuned to the system**.

Unfortunately, that means the actual data type can vary from one compiler to another.

For example, **`long int`** may be a **32-bit 2's complement** value, or it may be a **64-bit 2's complement** value.



Primitive Integer and Floating-Point Types in C

	2's complement	unsigned
8 bits	`char`	`unsigned char`

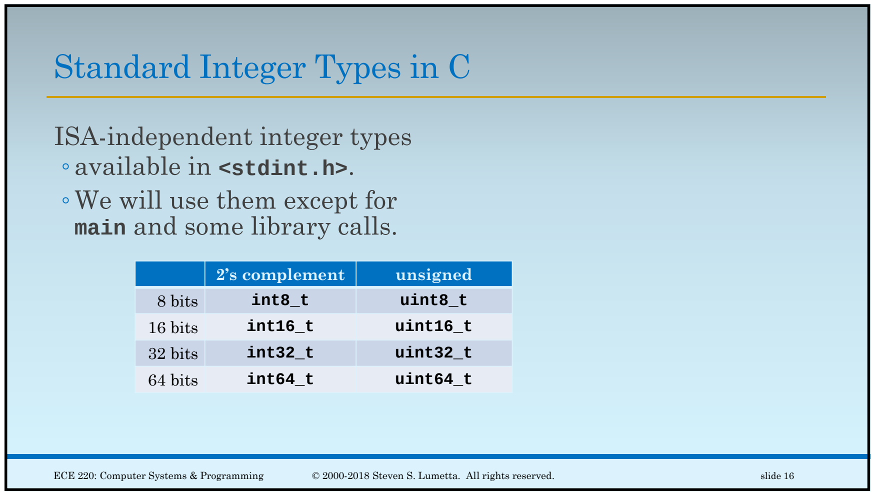
16 bits	`short`
	`short int`
		`unsigned short`
		`unsigned short int`

16 or 32 bits	`int`	`unsigned`
		`unsigned int`

32 or 64 bits	`long`
	`long int`
		`unsigned long`
		`unsigned long int`

64 bits	`long long`
	`long long int`
		`unsigned long long`
		`unsigned long long int`

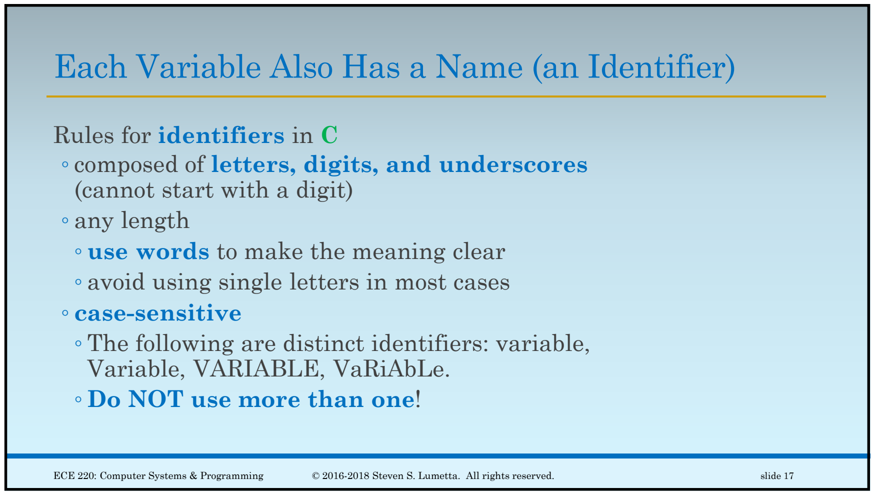
IEEE 754 single-precision floating-point (32 bits)	`float`
IEEE 754 double-precision floating-point (64 bits)	`double`



Standard Integer Types in C

ISA-independent integer types
- available in **`<stdint.h>`**.
- We will use them except for **`main`** and some library calls.

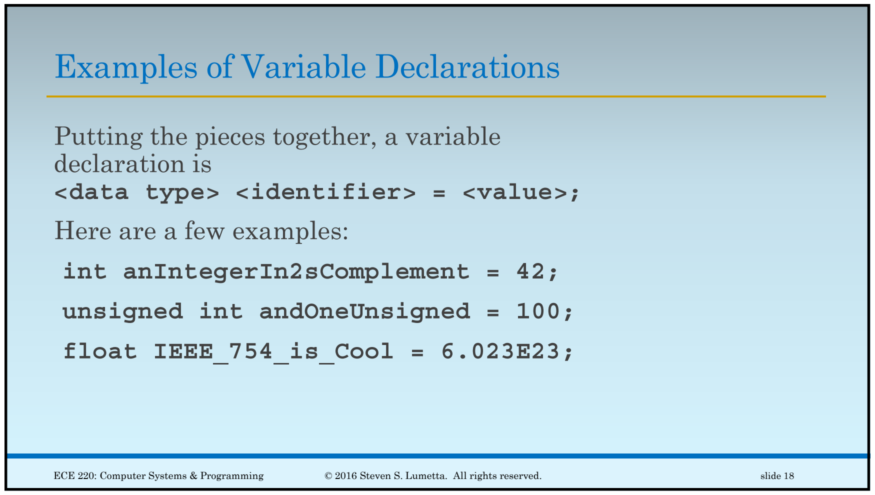
	2's complement	unsigned
8 bits	`int8_t`	`uint8_t`
16 bits	`int16_t`	`uint16_t`
32 bits	`int32_t`	`uint32_t`
64 bits	`int64_t`	`uint64_t`



Each Variable Also Has a Name (an Identifier)

Rules for **identifiers** in **C**

- composed of **letters, digits, and underscores** (cannot start with a digit)
- any length
  - **use words** to make the meaning clear
  - avoid using single letters in most cases
- **case-sensitive**
  - The following are distinct identifiers: variable, Variable, VARIABLE, VaRiAbLe.
  - **Do NOT use more than one**!



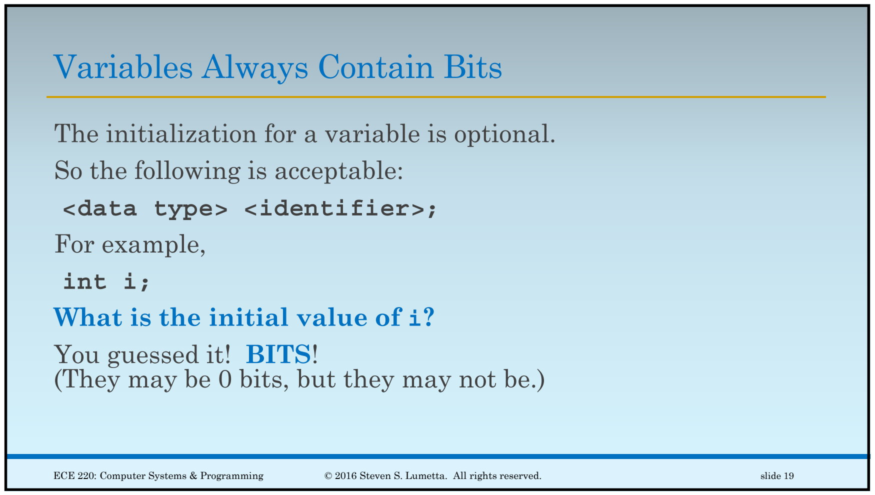
Examples of Variable Declarations

Putting the pieces together, a variable declaration is

```
<data type> <identifier> = <value>;
```

Here are a few examples:

```
int anIntegerIn2sComplement = 42;
unsigned int andOneUnsigned = 100;
float IEEE_754_is_Cool = 6.023E23;
```



Variables Always Contain Bits

The initialization for a variable is optional.

So the following is acceptable:

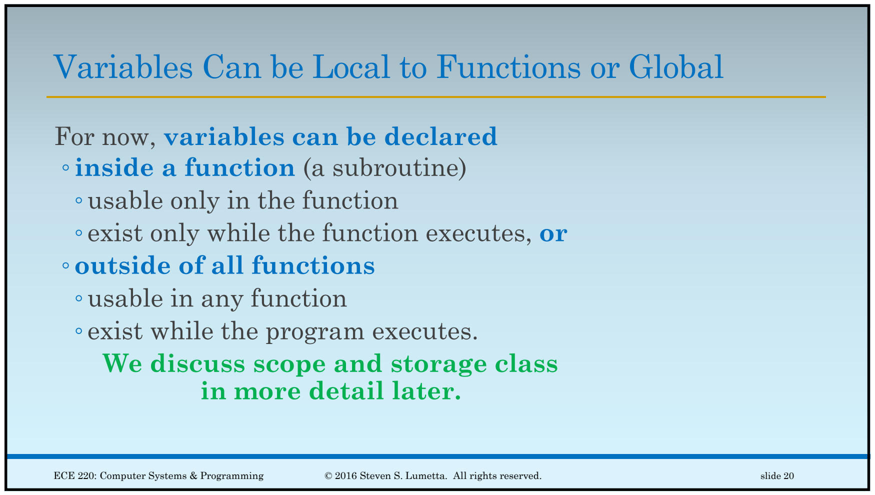
```
<data type> <identifier>;
```

For example,

```
int i;
```

What is the initial value of `i`?

You guessed it! **BITS**!
(They may be 0 bits, but they may not be.)



Variables Can be Local to Functions or Global

For now, **variables can be declared**

- **inside a function** (a subroutine)
  - usable only in the function
  - exist only while the function executes, **or**
- **outside of all functions**
  - usable in any function
  - exist while the program executes.

We discuss scope and storage class in more detail later.

Statements Tell the Computer What to Do

In **C**, a statement specifies a complete operation.

In other words, **a statement tells the computer to do something**.

There are **three types of statements**.

But statements can consist of other statements,

which can consist of other statements,

and so forth.

Many Statements are Quite Simple

Here are two of the three types…

```
;       /* a null statement */

/* A simple statement is often an
   expression and a semicolon. */
A = B;  /* simple statements */
printf ("Hello, ECE220!\n");
```

These two types **end with a semicolon** (;).

Compound Statements Consist of Other Statements

Third type: a **compound statement** consists of
- a **sequence of statements**
- **between braces**.

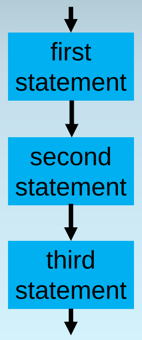
```
{   /* a compound statement */
    radius = 42;
    C = 2 * 3.1416 * radius;
    printf ("C = %f\n", C);
}
```

A compound statement may also contain variable declarations for use inside the statement.

A Program is a Sequence of Statements

The function body of **`main`** is a compound statement.

When program is **started** (or **runs**, or **executes**),
- **the computer executes the statements in `main`**
- in the order that they appear in the program.


first statement
second statement
third statement