Variables and numeric types

Comp Sci 1570 Introduction to C++

Missouri S&T | Computer Science

Outline

1. Introduction
   - Needs
2. Variables
   - Objects
   - Variable creation
   - Declaration vs. definition
   - Initialization vs. assignment
     - Types of initialization
3. Bits and Bytes
4. Variable types
   - Overview charts
   - Type examples
     - short
     - int
     - float
   - Type sizes
5. Variable naming
6. Where to put variables?

Outline

1. Introduction
    - Needs
2. Variables
    - Objects
    - Variable creation
    - Declaration vs. definition
    - Initialization vs. assignment
        - Types of initialization
3. Bits and Bytes
4. Variable types
    - Overview charts
    - Type examples
        - short
        - int
        - float
    - Type sizes
5. Variable naming
6. Where to put variables?

Needs

- Most programs need to manipulate data:
  input values, **output** values, **store** values, **modify** values.
- You accomplish that in large part using variables.
- **Variable** is a modifiable memory address with a name, alias, or identifier.
- Similar to variables in math to represent values that can change, but not entirely the same

```
a = 5;
b = 2;
a = a + 1;
result = a - b;
```

Outline

1. Introduction
   - Needs
2. Variables
   - Objects
   - Variable creation
   - Declaration vs. definition
   - Initialization vs. assignment
     - Types of initialization
3. Bits and Bytes
4. Variable types
   - Overview charts
   - Type examples
     - short
     - int
     - float
   - Type sizes
5. Variable naming
6. Where to put variables?

Variables

- Provides named storage that programs can manipulate
- Each variable in C++ has a specific type, which determines:
    - size and layout of the variable's memory
    - range of values that can be stored within that memory
    - set of operations that can be applied to the variable

Outline

1. Introduction
    - Needs
2. Variables
    - Objects
    - Variable creation
    - Declaration vs. definition
    - Initialization vs. assignment
        - Types of initialization
3. Bits and Bytes
4. Variable types
    - Overview charts
    - Type examples
        - short
        - int
        - float
    - Type sizes
5. Variable naming
6. Where to put variables?

Objects

- C++ programs create, access, manipulate, and destroy objects
- Object is a segment of memory that can be used to store values.
- Objects can store information for later retrieval and manipulation
- When an object is defined, memory is set aside for the object.
- Many objects in C++ come in the form of variables.

Outline

1. Introduction
    - Needs
2. Variables
    - Objects
    - Variable creation
    - Declaration vs. definition
    - Initialization vs. assignment
        - Types of initialization
3. Bits and Bytes
4. Variable types
    - Overview charts
    - Type examples
        - short
        - int
        - float
    - Type sizes
5. Variable naming
6. Where to put variables?

Making new variables

- Statement such as
  $x = 5;$
- We are assigning the value of 5 to x.
- But what exactly is x?
  x is a variable.
- In order to create a variable, we generally use a special kind of declaration statement called a definition (more later)
- Here's an example of defining variable x as an integer variable (one that can hold integer values):

type variable_list;

```
int x;
```

Integers can be written without a fractional component, such as -12, -1, 0, 4, or 27.

Making new variables

- Upon definition memory from RAM will be set aside (called instantiation).
- E.g., variable x is assigned memory location 140.
- One of the most common operations done with variables is assignment.
- Use the assignment operator, more commonly known as the = symbol.
- Later in our program, we could print that value to the screen using std::cout:

```
x = 5;

// prints the value of x (memory location 140)
std::cout << x;
```

Define variables before you use them, or get a compiler error

Outline

1. Introduction
   - Needs
2. Variables
   - Objects
   - Variable creation
   - Declaration vs. definition
   - Initialization vs. assignment
     - Types of initialization
3. Bits and Bytes
4. Variable types
   - Overview charts
   - Type examples
     - short
     - int
     - float
   - Type sizes
5. Variable naming
6. Where to put variables?

Declaration vs. definition

- In C++, you must define the variables you are going to use before you use them.
- A **declaration** is a statement that announces an identifier (variable or function name) and its type.
- A **definition** actually implements or instantiates (causes memory to be allocated for) the identifier.
- Don't worry about this too much until we hit functions

Outline

1. Introduction
   - Needs
2. Variables
   - Objects
   - Variable creation
   - Declaration vs. definition
   - Initialization vs. assignment
     - Types of initialization
3. Bits and Bytes
4. Variable types
   - Overview charts
   - Type examples
     - short
     - int
     - float
   - Type sizes
5. Variable naming
6. Where to put variables?

Initialization vs. assignment

- C++ does not initialize most variables to a given value (such as zero) automatically upon definition
- When variables are defined, they have an undetermined value until they are first assigned a value
- variables without initialization or assignment are called an uninitialized variables
- After a variable is defined, a value may be assigned to it via the assignment operator (the = sign):
- C++ will let you both define a variable AND give it an initial value in the same step.
  This is called initialization.

```
int x; // this is a variable definition
x = 5; // assign the value 5 to variable x
int x = 5; // initialize variable x with 5
```

Uninitialized variables

```
#include <iostream>
int main()
{
    // define an integer variable named x
    // this variable is uninitialized
    int x;

    // print the value of x to the screen
    //dangerous, because x is uninitialized
    std::cout << x;

    return 0;
}
```

Execute

Uninitialized variables

- The computer will assign some unused memory to x.
- It will then send the value residing in that memory location to std::cout, which will print the value.
- But what value will it print?
- The answer is "who knows!", and the answer may change every time you run the program.
- When a variable is assigned a memory location by the compiler, the default value of that variable is whatever (garbage) value happens to already be in that memory location!

Initialize your variables where possible

- A good rule of thumb is to initialize your variables at definition.
- This ensures that your variable will always have a consistent value, making it easier to debug if something goes wrong somewhere else.

Several types of initialization

C-like initialization (because it is inherited from the C language), consists of appending an equal sign followed by the value to which the variable is initialized:

type identifier = *initial_value*;

For example, declare a variable of type int called x and initialize it to a value of zero from the same moment it is declared, we can write:

```
int x = 0;
```

Constructor initialization (introduced by the C++ language), encloses the initial value between parentheses (()):

typeidentifier(*initial_value*);

```
int x (0);
```

Initialization of variables

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

int main ()
{
  int a=5;    // initial value: 5
  int b(3);    // initial value: 3
  int c{2};   // initial value: 2. C++ 11 only!
  int result; // initial value undetermined

  a = a + b;
  result = a - c;
  cout << result;

  return 0;
}
```

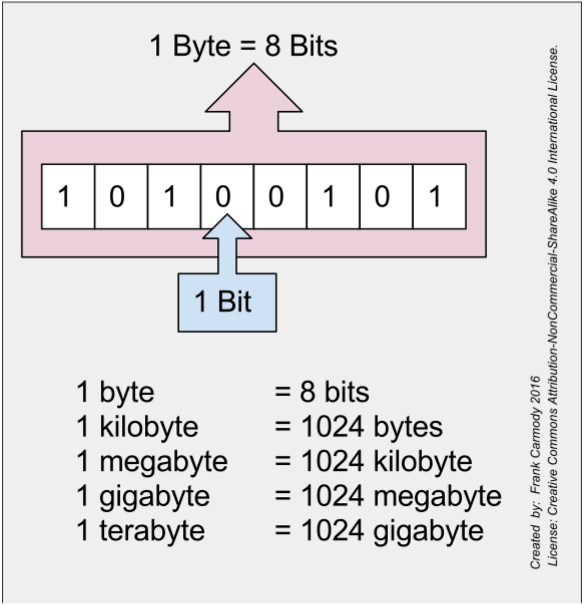
Initialization of variables

```
#include <iostream>
int main ()
{
  // declaring variables:
  int a, b, c;
  int result = 4, uselessProgram=2;

  // Assignment
  a = 5;
  b = 2;

  a = b + uselessProgram;
  std::cout << a << "␣and␣" << result;

  return 0;
}
```

Outline

1. Introduction
   - Needs
2. Variables
   - Objects
   - Variable creation
   - Declaration vs. definition
   - Initialization vs. assignment
     - Types of initialization
3. Bits and Bytes
4. Variable types
   - Overview charts
   - Type examples
     - short
     - int
     - float
   - Type sizes
5. Variable naming
6. Where to put variables?

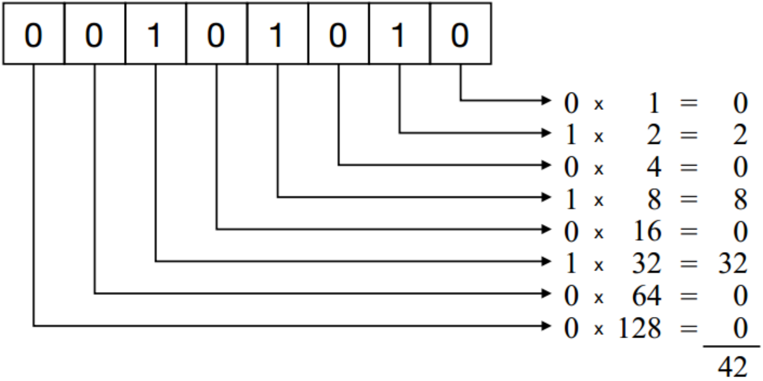
Bits and bytes: how many combinations?



1 byte	= 8 bits
1 kilobyte	= 1024 bytes
1 megabyte	= 1024 kilobyte
1 gigabyte	= 1024 megabyte
1 terabyte	= 1024 gigabyte

Created by: Frank Carmody 2016

Binary and decimal



0	0	1	0	1	0	1	0

$0 \times 1 = 0$
$1 \times 2 = 2$
$0 \times 4 = 0$
$1 \times 8 = 8$
$0 \times 16 = 0$
$1 \times 32 = 32$
$0 \times 64 = 0$
$0 \times 128 = 0$

42

Bits and bytes

- 1 bit is the smallest memory storage unit.
  It can be either 1 or 0.
- 8 bits is one byte.
- 2 bytes is 16 bits.
- Assuming only non-negative numbers,
  2 bytes will store $[0 \text{ to } 2^{16} - 1]$
  (the -1 is because we have to store 0)
- However, we also want to have negative numbers, so the range of a 2-byte value is what?

Outline

1. Introduction
   - Needs
2. Variables
   - Objects
   - Variable creation
   - Declaration vs. definition
   - Initialization vs. assignment
     - Types of initialization
3. Bits and Bytes
4. **Variable types**
   - Overview charts
   - Type examples
     - short
     - int
     - float
   - Type sizes
5. Variable naming
6. Where to put variables?

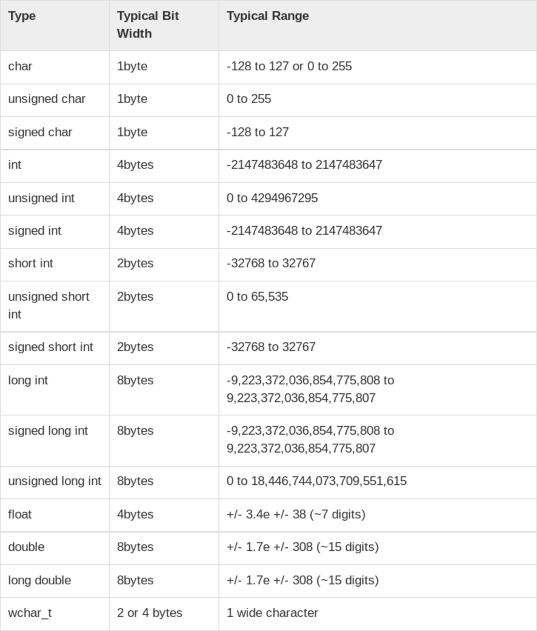
Variable types

- What is different about storing a simple integer vs. large floating-point number?
- Each variable declaration must be given a datatype, on which the memory assigned to the variable depends.
- Values of variables are stored somewhere in an unspecified location in the computer memory as zeros and ones.
- C++ is a strongly-typed language, and requires every variable to be declared with its type before its first use.
- This informs the compiler the size to reserve in memory for the variable and how to interpret its value.
- Syntax for variable type is:
  type variableName (i.e., its identifier)

Outline

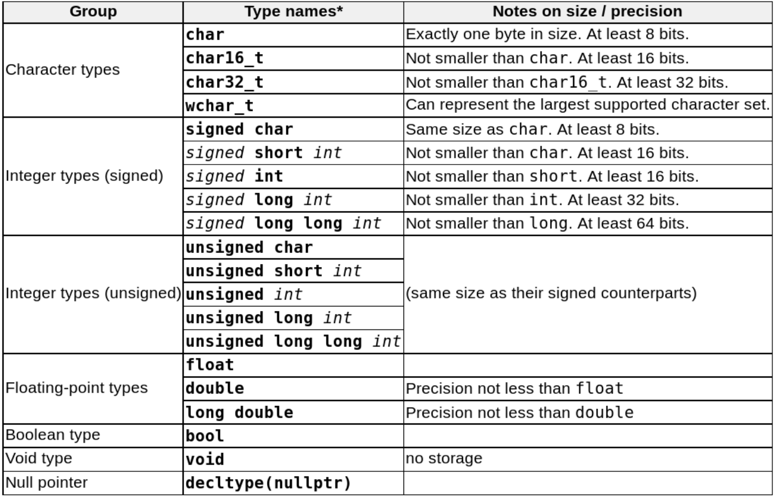
1. Introduction
   - Needs
2. Variables
   - Objects
   - Variable creation
   - Declaration vs. definition
   - Initialization vs. assignment
     - Types of initialization
3. Bits and Bytes
4. **Variable types**
   - **Overview charts**
   - Type examples
     - short
     - int
     - float
   - Type sizes
5. Variable naming
6. Where to put variables?

Overview of types

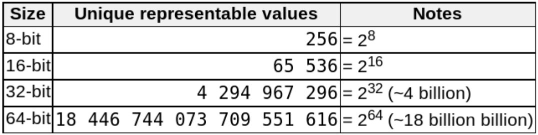


Type	Typical Bit Width	Typical Range
char	1byte	-128 to 127 or 0 to 255
unsigned char	1byte	0 to 255
signed char	1byte	-128 to 127
int	4bytes	-2147483648 to 2147483647
unsigned int	4bytes	0 to 4294967295
signed int	4bytes	-2147483648 to 2147483647
short int	2bytes	-32768 to 32767
unsigned short int	2bytes	0 to 65,535
signed short int	2bytes	-32768 to 32767
long int	8bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
signed long int	8bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
unsigned long int	8bytes	0 to 18,446,744,073,709,551,615
float	4bytes	+/- 3.4e +/- 38 (~7 digits)
double	8bytes	+/- 1.7e +/- 308 (~15 digits)
long double	8bytes	+/- 1.7e +/- 308 (~15 digits)
wchar_t	2 or 4 bytes	1 wide character

Type syntax abbreviations



Group	Type names*	Notes on size / precision
Character types	**char**	Exactly one byte in size. At least 8 bits.
	char16_t	Not smaller than char. At least 16 bits.
	char32_t	Not smaller than char16_t. At least 32 bits.
	wchar_t	Can represent the largest supported character set.
Integer types (signed)	**signed char**	Same size as char. At least 8 bits.
	signed **short** *int*	Not smaller than char. At least 16 bits.
	signed **int**	Not smaller than short. At least 16 bits.
	signed **long** *int*	Not smaller than int. At least 32 bits.
	signed **long long** *int*	Not smaller than long. At least 64 bits.
Integer types (unsigned)	**unsigned char**	(same size as their signed counterparts)
	unsigned short *int*	
	unsigned *int*	
	unsigned long *int*	
	unsigned long long *int*	
Floating-point types	**float**	
	double	Precision not less than float
	long double	Precision not less than double
Boolean type	**bool**	
Void type	**void**	no storage
Null pointer	**decltype(nullptr)**	



Size	Unique representable values	Notes
8-bit	256	= 2^{8}
16-bit	65 536	= 2^{16}
32-bit	4 294 967 296	= 2^{32} (~4 billion)
64-bit	18 446 744 073 709 551 616	= 2^{64} (~18 billion billion)

Outline

1. Introduction
   - Needs
2. Variables
   - Objects
   - Variable creation
   - Declaration vs. definition
   - Initialization vs. assignment
     - Types of initialization
3. Bits and Bytes
4. Variable types
   - Overview charts
   - Type examples
     - short
     - int
     - float
   - Type sizes
5. Variable naming
6. Where to put variables?

signed **short** int

- Memory allocation: 2 Bytes
- Range of values: -2^{15} to $+2^{15}-1$ or about -32 K to 32 K

```
short shoeSize;
short shoeSize = 5;
```

Note: Since ints can vary from 2 to 4 bytes, it's best not to use them. However, they are indeed very popular. But, suppose you declare a variable as an int, Then transfer that program over to another system using a different allocation. Your program may indeed crash!

signed **int**

- Memory allocation: varies. some systems give 2 bytes, some give 4. The system at S&T allocates 4 bytes.
- Range of values: -2^{31} to $+2^{31}-1$ or about -2 billion to 2 billion

```
int numCarsOnHwy;
```

Introduction
Needs
Variables
Objects
Variable creation
Declaration vs. definition
Initialization vs. assignment
Types of initialization
Bits and Bytes
Variable types
Overview charts
Type examples
short
int
float
Type sizes
Variable naming
Where to put variables?

float

type: floating point

- allocation: 4 bytes
- precision: you get 6 significant figures (decimal)

type: double precision

- allocation: 8 bytes
- precision: you get 15 significant figures (decimal)

type: long double

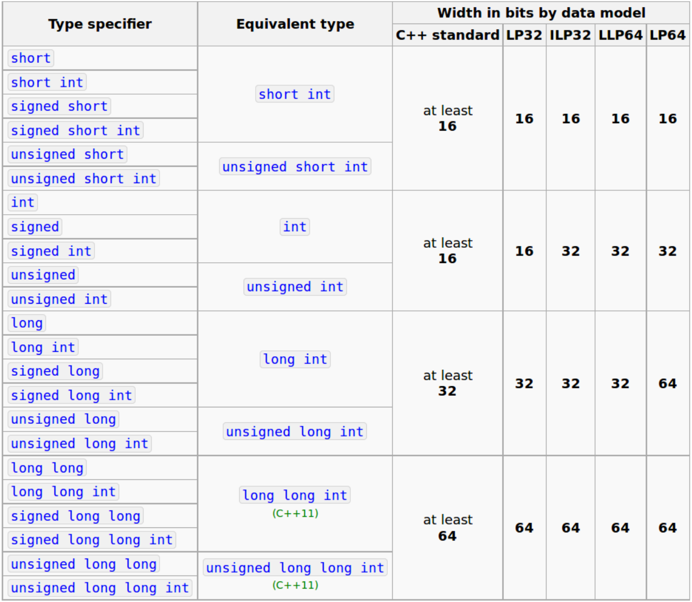
- allocation: non-standard
- precision: lots!

```
float shoeSize = 6.5;
double weight_of_mountain = 746538433.55;
long double electron_wt = 0.00000000000000432;
```

Outline

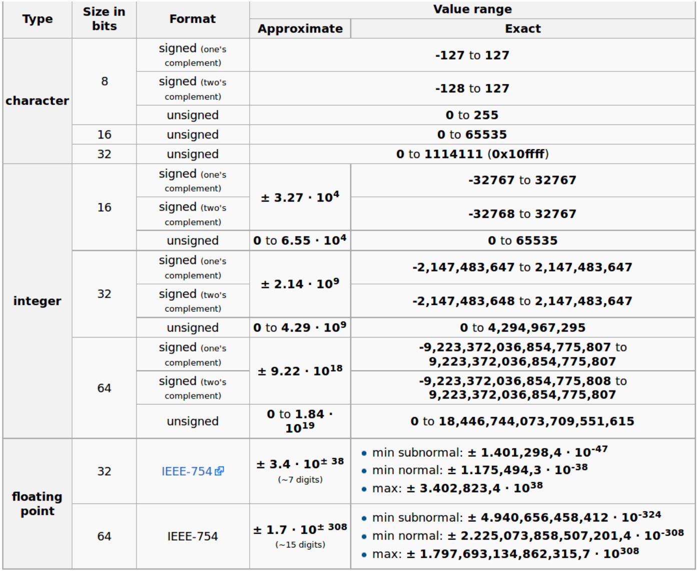
1. Introduction
   - Needs
2. Variables
   - Objects
   - Variable creation
   - Declaration vs. definition
   - Initialization vs. assignment
     - Types of initialization
3. Bits and Bytes
4. Variable types
   - Overview charts
   - Type examples
     - short
     - int
     - float
   - Type sizes
5. Variable naming
6. Where to put variables?

Type syntax abbreviations



Type specifier	Equivalent type	Width in bits by data model				
		C++ standard	LP32	ILP32	LLP64	LP64
`short`	`short int`	at least 16	16	16	16	16
`short int`						
`signed short`						
`signed short int`						
`unsigned short`	`unsigned short int`					
`unsigned short int`						
`int`	`int`	at least 16	16	32	32	32
`signed`						
`signed int`						
`unsigned`	`unsigned int`					
`unsigned int`						
`long`	`long int`	at least 32	32	32	32	64
`long int`						
`signed long`						
`signed long int`						
`unsigned long`	`unsigned long int`					
`unsigned long int`						
`long long`	`long long int` (C++11)	at least 64	64	64	64	64
`long long int`						
`signed long long`						
`signed long long int`						
`unsigned long long`	`unsigned long long int` (C++11)					
`unsigned long long int`						

Type ranges



Type	Size in bits	Format	Value range	
			Approximate	Exact
character	8	signed (one's complement)	**-127** to **127**	
		signed (two's complement)	**-128** to **127**	
		unsigned	**0** to **255**	
	16	unsigned	**0** to **65535**	
	32	unsigned	**0** to **1114111** (**0x10ffff**)	
integer	16	signed (one's complement)	**± 3.27 · 10^4**	**-32767** to **32767**
		signed (two's complement)		**-32768** to **32767**
		unsigned	**0** to **6.55 · 10^4**	**0** to **65535**
	32	signed (one's complement)	**± 2.14 · 10^9**	**-2,147,483,647** to **2,147,483,647**
		signed (two's complement)		**-2,147,483,648** to **2,147,483,647**
		unsigned	**0** to **4.29 · 10^9**	**0** to **4,294,967,295**
	64	signed (one's complement)	**± 9.22 · 10^{18}**	**-9,223,372,036,854,775,807** to **9,223,372,036,854,775,807**
		signed (two's complement)		**-9,223,372,036,854,775,808** to **9,223,372,036,854,775,807**
		unsigned	**0** to **1.84 · 10^{19}**	**0** to **18,446,744,073,709,551,615**
floating point	32	IEEE-754	**± 3.4 · $10^{\pm 38}$** (~7 digits)	• min subnormal: **± 1.401,298,4 · 10^{-47}** • min normal: **± 1.175,494,3 · 10^{-38}** • max: **± 3.402,823,4 · 10^{38}**
	64	IEEE-754	**± 1.7 · $10^{\pm 308}$** (~15 digits)	• min subnormal: **± 4.940,656,458,412 · 10^{-324}** • min normal: **± 2.225,073,858,507,201,4 · 10^{-308}** • max: **± 1.797,693,134,862,315,7 · 10^{308}**

Get the size of a type

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

int main() {
  cout << "Size␣of:";
  cout << "int=" << sizeof(int);
  cout << ",␣short␣int=" << sizeof(short int);
  cout << ",␣long␣int=" << sizeof(long int);
  cout << ",␣float=" << sizeof(float);
  cout << ",␣double=" << sizeof(double);
  cout << ",␣long␣long=" << sizeof(long long);
  return 0;
}
```

Outline

1. Introduction
    - Needs
2. Variables
    - Objects
    - Variable creation
    - Declaration vs. definition
    - Initialization vs. assignment
        - Types of initialization
3. Bits and Bytes
4. Variable types
    - Overview charts
    - Type examples
        - short
        - int
        - float
    - Type sizes
5. Variable naming
6. Where to put variables?

Variable naming

- A valid identifier is a sequence of one or more letters, digits, or underscore characters (_).
- No spaces, punctuation marks, and symbols
- Identifiers usually begin with a letter, and never a numeric character
- Can begin with an (_), but considered reserved for compiler-specific keywords or external identifiers, as well as identifiers containing two successive underscore characters anywhere.
- C++ "case sensitive"
- **RESULT** variable is not the same as the **result** variable or the **Result** variable.
  These are three different identifiers identifying three different variables.

Examples of variable names

- **legal names**: x, y, *xyz_Hello*, bob
  all adhere to the rules
- **illegal names**: 8Hello, hi-there, Go!
  starts with numeric, special character, special character

Naming practice

- Name variables meaningfully for your own benefit (the compiler does not care)
- **Instead of**:
  i and j,
  name variables like:
  quantity age, tirePressure, *lateral_distortion*, etc.
- These examples demonstrate readability; you know immediately what they represent.
- Several ways to include two words in one identifier.

Comments

It is also nice to put comments in your variable declarations to give the reader some idea what the variables will be used for. For example...

```
// count of experiment
short numParticipants;

// user response
char answer;
```

Outline

1. Introduction
   - Needs
2. Variables
   - Objects
   - Variable creation
   - Declaration vs. definition
   - Initialization vs. assignment
     - Types of initialization
3. Bits and Bytes
4. Variable types
   - Overview charts
   - Type examples
     - short
     - int
     - float
   - Type sizes
5. Variable naming
6. Where to put variables?

Put variables in main

- For now, it is easiest to declare variables at the top (inside) of the main function.
- For now, DO NOT declare variables outside of the main function.
- This has the effect of creating dangerous situations which I will discuss later.