VARIABLES AND C++ DATA TYPES

mathExample2.cpp

Same calculations using 10 and 15

```
// math example

#include <iostream>
#include <cmath>

using namespace std;

int main()
{
  cout << "The reciprocal of 10 is " << 1.0/10.0 << endl;
  cout << "The square root of 10 is " << sqrt(10.0) << endl;
  cout << "e^(10.0) = " << exp(10.0) << endl;
  cout << "The reciprocal of 15 is " << 1.0/15.0 << endl;
  cout << "The square root of 15 is " << sqrt(15.0) << endl;
  cout << "e^(15.0) = " << exp(15.0) << endl;

  return 0;      // exit program
}
```

mathExample2.cpp

```
> g++ -o mathExample2.exe mathExample2.cpp
```

Save as this

```
> mathExample2.exe
The reciprocal of 10 is 0.1
The square root of 10 is 3.16228
e^(10.0) = 22026.5
The reciprocal of 15 is 0.0666667
The square root of 15 is 3.87298
e^(15.0) = 3.26902e+06

>
```

Variables

- A *variable* in algebra is a symbol that holds a number
- A *variable* in C++ holds a number or maybe another kind (type) of data
  - The data is held in our computers main memory (RAM)
  - A program can:
    - Read the current value in the variable
    - Write a new value into the variable
  - Thus, as the name "variable" suggests, its value can vary in the program

mathExample3.cpp

```
// math example

#include <iostream>
#include <cmath>

using namespace std;

int main()
{
  double x; // this is a variable declaration statement

  x = 10.0;
  cout << "The reciprocal of 10 is " << 1.0/x << endl;
  cout << "The square root of 10 is " << sqrt(x) << endl;
  cout << "e^(" << x << ") = " << exp(x) << endl;

  x = 15.0;
  cout << "The reciprocal of 15 is " << 1.0/x << endl;
  cout << "The square root of 15 is " << sqrt(x) << endl;
  cout << "e^(" << x << ") = " << exp(x) << endl;

  return 0;     // exit program
}
```

Variable Declaration

- C++ variable declarations are of the form:

```
dataType variableName;
```

dataType: int, float, char, double, unsigned int, ...

variableName: composed of alphanumeric characters or underscore '_'

- Example:

```
double x;
```

Declaration Example

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

int main()
{
   int age;
   float wage;
   char initial;
   double height;

   return 0;     // exit program
}
```

Just allocates mem for variables

- **IMPORTANT: You <u>MUST</u> declare your variable BEFORE you can use it in your program**
  - The computer will set aside the memory for the variable

Variable Names

- Memorize these rules!
- Composed of the characters:
  a, b, c,…, z, A, **B**, C,…, Z, 0, 1, 2,…, 9 and _
- Must begin with:
  a, b, c,…, z, A, B, C,…, Z or _
- Case sensitive
  - Capitalized and lower case letters are different
- Example variable declarations (where int is the data type):

```
int age;
int Age;
int myAge;
int Jacks_age;
```

```
int age1;
int age2;
int age3B;
int _age;
```

Your Turn: Variable Names

Which of these are valid variable names?

A. me2
B. Me2
C. 2L8
D. J-Wilcox
E. He_who_hesitates_is_lost
F. HeWhoHesitatesIsLost
G. Gordon.Gee
H. jason_day

Your Turn: Variable Names

Which of these are valid variable names?

A. me2
B. Me2
C. 2L8
D. J-Wilcox
E. He_who_hesitates_is_lost
F. HeWhoHesitatesIsLost
G. Gordon.Gee
H. jason_day

Variable Assignments

```
x = 10.0;
```

- The equals sign is the assignment operator
  - An operator performs an action, like the mathematical operators + or – for addition and subtraction
- The assignment operator is NOT commutative
  - **Wrong!**: `10.0 = x;`

Syntax Error

What kind of error is this?

- **Rule: The variable must always be on the left side of an equals sign**

Variable Assignments

- C++ variable assignments are of the form:

```
variableName = expression;
```

variableName: valid variable name
expression: formula that evaluates to a single value

- Examples:

```
weight = 160;
totalPay = salary + overtime;
```

```
bool x = (2+2 == 5)
```

Variable Assignments

- C++ variable assignments are of the form:

```
variableName = expression;
```

variableName: valid variable name
expression: formula that evaluates to a single value

- Why is this wrong?

```
phrase = Hello World;
```

No "Quotes" - Syntax Error

Variable Assignments

- Variable assignments MUST be done during or after variable declaration
  - Never before! Why???

- A variable called `dogs` to store how many dogs

```
int dogs;    // declaration
dogs = 2;    // assignment
```

OR

```
int dogs = 2; // assignment during declaration
```

OR

int dogs (2)

Ask the user for variable value

```
// math example

#include <iostream>
#include <cmath>

using namespace std;

int main()
{
  double x; // this is a variable declaration statement

  // Ask the user for the value of variable x here ?????

  cout << "The reciprocal of " << x << " is " << 1.0/x << endl;
  cout << "The square root of " << x << " is " << sqrt(x) << endl;
  cout << "e^(" << x << ") = " << exp(x) << endl;

  return 0;      // exit program
}
```

Input Using `cin`

```
…
  double x;

  cout << "Enter x: ";   // Note: no new line
  cin >> x;              // Note: operator ">>", not operator "<<"
…Arrows point to var that gets input
```

- `cin` (**C**onsole **IN**put) can be used to obtain user input (keyboard)
- **Rule: Unlike cout, use >> with cin, and not <<**
- When the program is run, `cin` will wait indefinitely for user input
- `cin` will input a single value into a variable when it detects a *new line* from the input
- The use of the cout statement before a cin is called *prompting the user*

mathExample4.cpp

```
// math example

#include <iostream>
#include <cmath>

using namespace std;

int main()
{
  double x;

  cout << "Enter x: ";    // Note: no endl (new line)
  cin >> x;               // Note: operator ">>", not operator "<<"

  cout << "The reciprocal of " << x << " is " << 1.0/x << endl;
  cout << "The square root of " << x << " is " << sqrt(x) << endl;
  cout << "e^(" << x << ") = " << exp(x) << endl;

  return 0;      // exit program
}
```

```
…
int main()
{
  double x;

  cout << "Enter x: ";    // Note: no new line
  cin >> x;              // Note: operator ">>", not operator "<<"

  cout << "The reciprocal of " << x << " is " << 1.0/x << endl;
  cout << "The square root of " << x << " is " << sqrt(x) << endl;
  cout << "e^(" << x << ") = " << exp(x) << endl;
…
```

```
> mathExample4.exe
Enter x:
```

```
…
int main()
{
  double x;

  cout << "Enter x: ";    // Note: no new line
  cin >> x;               // Note: operator ">>", not operator "<<"

  cout << "The reciprocal of " << x << " is " << 1.0/x << endl;
  cout << "The square root of " << x << " is " << sqrt(x) << endl;
  cout << "e^(" << x << ") = " << exp(x) << endl;
…
```

```
> mathExample4.exe
Enter x: 10.0
```

```
…
int main()
{
  double x;

  cout << "Enter x: ";    // Note: no new line
  cin >> x;              // Note: operator ">>", not operator "<<"

  cout << "The reciprocal of " << x << " is " << 1.0/x << endl;
  cout << "The square root of " << x << " is " << sqrt(x) << endl;
  cout << "e^(" << x << ") = " << exp(x) << endl;
…
```

```
> mathExample4.exe
Enter x: 10.0
The reciprocal of 10 is 0.1
The square root of 10 is 3.16228
e^(10) = 22026.5
>
```

mathExample4.cpp

```
// math example

#include <iostream>
#include <cmath>

using namespace std;

int main()
{
  double x;

  cout << "Enter x: ";        // Note: no new line
  cin >> x;                   // Note: operator ">>", not operator "<<"

  cout << "The reciprocal of " << x << " is " << 1.0/x << endl;
  cout << "The square root of " << x << " is " << sqrt(x) << endl;
  cout << "e^(" << x << ") = " << exp(x) << endl;

  return 0;      // exit program
}
```

Try inputs:

20

-20

1000

-1000

Multiple Variable Declarations

- Declare three variables called **`x`**, **`y`**, and **`z`**:

```
double x;
double y;
double z;
```

Remember: If a var is created but not assigned a value, it is "NULL" and cannot be used until assigned a value.

- Instead, we can declare these in one statement:

```
double x, y, z;
```

OR double x = 4, y = 5, z = 10;

- Very useful when creating several variables of the same type

xyz1.cpp

```
// multiple declarations example

#include <iostream>
using namespace std;

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

  cout << "Enter x, y and z: ";   // prompt the user
  cin >> x;
  cin >> y;
  cin >> z;

  cout << "x = " << x << endl;
  cout << "y = " << y << endl;
  cout << "z = " << z << endl;

  return 0;              // exit program
}
```

```
…

  cout << "Enter x, y and z: ";
  cin >> x;
  cin >> y;
  cin >> z;

  cout << "x = " << x << endl;
  cout << "y = " << y << endl;
  cout << "z = " << z << endl;
…
```

```
> xyz1.exe
Enter x, y and z:
```

```
…

  cout << "Enter x, y and z: ";
  cin >> x;
  cin >> y;
  cin >> z;

  cout << "x = " << x << endl;
  cout << "y = " << y << endl;
  cout << "z = " << z << endl;
…
```

```
> xyz1.exe
Enter x, y and z: 1 2 3
```

```
…

  cout << "Enter x, y and z: ";
  cin >> x;
  cin >> y;
  cin >> z;

  cout << "x = " << x << endl;
  cout << "y = " << y << endl;
  cout << "z = " << z << endl;
…
```

```
> xyz1.exe
Enter x, y and z: 1 2 3
x = 1
y = 2
z = 3

>
```

```
…

  cout << "Enter x, y and z: ";
  cin >> x;
  cin >> y;
  cin >> z;

  cout << "x = " << x << endl;
  cout << "y = " << y << endl;
  cout << "z = " << z << endl;
…
```

```
> xyz1.exe
Enter x, y and z:
1
2
3
x = 1
y = 2
z = 3

>
```

Input buffer stops at each var that needs a value, and pointer will move between them at each whitespace entered (newline, space, etc.)

xyz2.cpp

```
// multiple declarations example

#include <iostream>
using namespace std;

int main()
{
  double x, y, z;   // multiple declarations on same line

  cout << "Enter x, y and z: ";
  cin >> x;
  cin >> y;
  cin >> z;

  cout << "x = " << x << endl;
  cout << "y = " << y << endl;
  cout << "z = " << z << endl;

  return 0;              // exit program
}
```

Multiple Inputs Using `cin`

- `cin` can be used to obtain multiple inputs
- `cin` knows when to delimit
  - I.e., start looking for the next input upon reaching a “whitespace”

Use:

```
cin >> x >> y >> z;
```

Instead of:

```
cin >> x;
cin >> y;
cin >> z;
```

- Whitespaces are: tabs, spaces, new lines

xyz3.cpp

```
// multiple declarations example

#include <iostream>
using namespace std;

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

  cout << "Enter x, y and z: ";
  cin >> x >> y >> z;                  // read x, then y, then z

  cout << "x = " << x << endl;
  cout << "y = " << y << endl;
  cout << "z = " << z << endl;

  return 0;                 // exit program
}
```

```
…
   cout << "Enter x, y and z: ";
   cin >> x >> y >> z;

   cout << "x = " << x << endl;
   cout << "y = " << y << endl;
   cout << "z = " << z << endl;
…
```

```
> xyz1.exe
Enter x, y and z: 1 2 3
x = 1
y = 2
z = 3

>
```

```
…
   cout << "Enter x, y and z: ";
   cin >> x >> y >> z;

   cout << "x = " << x << endl;
   cout << "y = " << y << endl;
   cout << "z = " << z << endl;
…
```

```
> xyz1.exe
Enter x, y and z:
1
2
3
x = 1
y = 2
z = 3

>
```

Breaking up Multiple Inputs

- Sometimes it makes more sense to break up multiple inputs into single inputs if you want to prompt particular variables:

```
int x, y, z;

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

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

cout << " Enter z: ";
cin >> z;
```

Your Turn: `cin` and `cout`

Which of the following C++ statements have syntax errors and what are the errors?

Assume that all variables have been declared and initialized

A. `cout >> "Answer = " >> x + y >> endl;` Arrows wrong direction

B. `cin << x;` Arrows wrong direction

C.
```
cout << "Yes " << " or " << " no " << " or "
<< " maybe. " << endl;
```

D. `cin >> yes >> no >> maybe;`

E. `cout << "x + y = " (x + y) << endl;` Missing arrows

Your Turn: `cin` and `cout`

Which of the following C++ statements have syntax errors and what are the errors?

Assume that all variables have been declared and initialized

The blue lines contain syntax errors.

A.
```
cout >> "Answer = " >> x + y >> endl;  // should use <<
```

B.
```
cin << x;   // should use >>
```

C.
```
cout << "Yes " << " or " << " no " << " or " << " maybe. "
<< endl;  // OK
```

D.
```
cin >> yes >> no >> maybe;  // OK
```

E.
```
cout << "x + y = " (x + y) << endl;// needs << before (
```

Data Types

- Remember, a variable must be declared with a data type BEFORE it is used in the program

- Commonly used data types

  - Integer – whole numbers Int
  - Real numbers – decimal numbers Float
  - Characters – a single symbol Char
  - Strings, i.e. text – 0 or more characters String
  - Boolean, i.e. true or false Bool

Data Type: Integers

- There are four basic data types:
  short (16 bits / 2 bytes)
  int (32 bits / 4 bytes)
  long (64 bits / 8 bytes)
  long long (128 bits / 16 bytes)
- In this class, always use `int` as the data type
- An integer value is any number that has no decimal point (whole number)

  123 -45000 +1432431 0 are all valid integers
- 1,244 is not a valid integer in C++
  - Commas are not allowed to express an integer
- $12 is not valid either
  - $ is not a valid part of an integer

Data Type: Integers

- What are the largest and smallest integers that a computer can support?
  - Depends on the computer on which the program is compiled
  - Most computers today use 32 bits to represent an int
    - So, $\mathbf{2^{32}}$ values can be represented (base 2 math)
    - How many is that??? … a lot!
- Integers can be **signed** or **unsigned**
  - What is the max value of an unsigned 32 bit integer? (>4B)
  - What is the max value of a signed 32 bit integer? (>2B)

Integer Division

- The following arithmetic expressions using integer division
  - 10 / 2 evaluates to 5
  - 21 / 7 evaluates to 3
  - 15 / 2 evaluates to what?
    - 15 / 2 evaluates to 7
  - 3 / 17 evaluates to what?
    - 3 / 17 evaluates to 0

Cuts off ANY decimal
1.9 is just 1

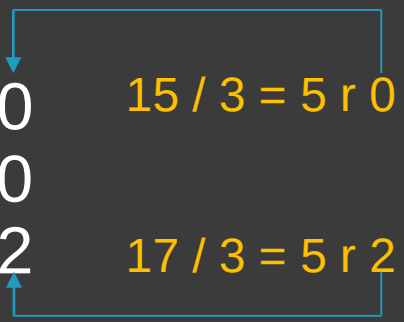
- Integer division:
  - Uses *truncation* (fractional part of the answer is removed)
  - Always evaluates to an integer

Integer Division Remainder

- Modulus (Mod) operator (%)
  - A binary integer operator
  - Computes the remainder of dividing two integers (integer math)
  - The remainder is an integer
  - Remember when you did long division in grade school and you used “r” to write the remainder

NEEDED IN A FUTURE LAB

- Examples
- 15 % 3 evaluates to 0
- 15 % 5 evaluates to 0
- 17 % 3 evaluates to 2



Arithmetic Operations

- The basic operations are +, - , *, /, %
  - Binary operators - simple arithmetic expressions of the form:

operand	*operator*	operand
5	%	3

Data Types: Decimal Numbers

- Floating-point numbers
  - Have a decimal point
  - Can be signed or unsigned
- There are three basic data types:
  float
  double
  long double
  - In this class, always use **double**
  - The differences between these are their supported range and precision

Data Types: Floating Point Numbers

- To represent a floating point number:
  - `float` uses 32 bits (4 bytes)
  - `double` uses 64 bits (8 bytes)
  - `long double` uses 128 bits (16 bytes)
- The tradeoff is storage vs. precision and range
- What exactly is the precision and range, and how are floating point numbers represented in binary format? **IEEE 754 Standard**

Data Types: Floating Point Numbers

- Let's always use "`double`" to represent floating point numbers
- A floating point number will always contain a decimal point

Data Types: Floating Point Numbers

- Floating-point numbers can be written in exponential notation:

  134.56 or *1.3456e2*

  -0.00345 or *-3.45e-3*

- Here, **`e`** is short for "*times ten to the power of* ", just like in scientific notation

Integer vs Floating point Division

- Integer division occurs when both operands are integer
  - 5 / 2 evaluates to 2
- Floating point division occurs when at least one operand is floating point
  - 5.0 / 2 evaluates to 2.5
  - 5 / 2.0 evaluates to 2.5
  - 5.0 / 2.0 evaluates to 2.5

Data Type: Characters

- A character can be any of the following:
  - All letters of the alphabet (upper and lower case, i.e. case sensitive)
  - The symbolic representation of digits 0 – 9
  - All various symbols such as: + * & ^ % $ | , !
- When declaring a variable to hold a character use **`char`** as the data type
- Write a character in single quotes
  - E.g., 'A' or '8' or ':' or ' ' (blank space)
  - Called a *character value* or *character literal*
  - Consists of exactly one character

Data Type: Characters

- A character is stored in the computer as a number
  - Thus, each character has an assigned number
- Characters are usually stored with 8 bits, i.e. 1 byte
  - So, there are 2^8 (or 256) different characters
  - Every number within the range of [0, 255] is mapped onto some character
- A character is simply a numerical representation known as **ASCII encoding**

ASCII Code

Code	Char
32	Space
33	!
34	"
35	#
36	$
37	%
38	&
39	'
40	(
41	)
…	…

Code	Char
48	0
49	1
50	2
51	3
52	4
53	5
54	6
55	7
56	8
57	9
…	…

Code	Char
65	A
66	B
67	C
68	D
69	E
70	F
71	G
72	H
73	I
74	J
…	…

Code	Char
97	a
98	b
99	c
100	d
101	e
102	f
103	g
104	h
105	i
106	j
…	…

ASCII Table

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0	00 NUL	01 SOH	02 STX	03 ETX	04 EOT	05 ENQ	06 ACK	07 BEL	08 BS	09 HT	10 LF	11 VT	12 FF	13 CR	14 SO	15 SI	8
1	16 DLE	17 DC1	18 DC2	19 DC3	20 DC4	21 NAK	22 SYN	23 ETB	24 CAN	25 EM	26 SUB	27 ESC	28 FS	29 GS	30 RS	31 US	9
2	32 SP	33 !	34 "	35 #	36 $	37 %	38 &	39 '	40 (	41)	42 *	43 +	44 ,	45 -	46 .	47 /	A
3	48 0	49 1	50 2	51 3	52 4	53 5	54 6	55 7	56 8	57 9	58 :	59 ;	60 <	61 =	62 >	63 ?	B
4	64 @	65 A	66 B	67 C	68 D	69 E	70 F	71 G	72 H	73 I	74 J	75 K	76 L	77 M	78 N	79 O	C
5	80 P	81 Q	82 R	83 S	84 T	85 U	86 V	87 W	88 X	89 Y	90 Z	91 [	92 \	93]	94 ^	95 _	D
6	96 `	97 a	98 b	99 c	100 d	101 e	102 f	103 g	104 h	105 i	106 j	107 k	108 l	109 m	110 n	111 o	E
7	112 p	113 q	114 r	115 s	116 t	117 u	118 v	119 w	120 x	121 y	122 z	123 {	124 |	125 }	126 ~	127 DEL	F

charExample1.cpp

```
// example using type char

#include <iostream>
using namespace std;

int main()
{
  char c1, c2, c3;

  cout << "Enter first initial: ";          // prompt the user
  cin >> c1;                                // read the character
  cout << "Enter second initial: ";         // prompt the user
  cin >> c2;                                // read the character

  c3 = 'X';

  cout << "Created by: ";
  cout << c1 << c2 << c3 << endl;

  return 0;      // exit program
}
```

```
…
   char c1, c2, c3;

   cout << "Enter first initial: ";
   cin >> c1;
   cout << "Enter second initial: ";
   cin >> c2;

   c3 = 'X';

   cout << "Created by: ";
   cout << c1 << c2 << c3 << endl;
…
```

```
> charExample1.exe
Enter first initial: R
```

```
…
   char c1, c2, c3;

   cout << "Enter first initial: ";
   cin >> c1;
   cout << "Enter second initial: ";
   cin >> c2;

   c3 = 'X';

   cout << "Created by: ";
   cout << c1 << c2 << c3 << endl;
…
```

```
> charExample1.exe
Enter first initial: R
Enter second initial: W
```

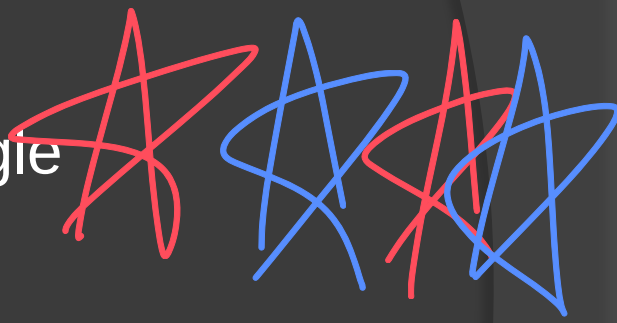
What is the output?

```
…
   char c1, c2, c3;

   cout << "Enter first initial: ";
   cin >> c1;
   cout << "Enter second initial: ";
   cin >> c2;

   c3 = 'X';

   cout << "Created by: ";
   cout << c1 << c2 << c3 << endl;
…
```

```
> charExample1.exe
Enter first initial: R
Enter second initial: W
Created by: RWX
```

Characters and String Literals

- NOTE: 'A' and "A" are different
  - 'A' is a *character literal*
  - "A" is a *string literal* containing a single character



- NOTE: '8' and "8" and 8 are different!!!
  - '8' is a character literal
  - "8" is a string literal containing a single character
  - 8 is the integer 8

String Literals

- A string literal is a sequence of zero or more characters
- Examples
  - “hello”
  - “Hello World”
    - The blank space is a character
  - “Hello\nGoodbye”
    - Note: \n is the new line character
  - “ “
    - Contains a single blank space character
  - “”
    - The empty string contains no characters
- string name(“John”);

Data Type: Boolean

- Represents two logical values **`true`** and **`false`**
  - They only consume 1 byte of storage
- bool flag = false;
- Interesting note
  - In C++, any number other than 0 is always interpreted as the Boolean value **`true`**
  - E.g., the number 16 is considered to be a **`true`** value

Summary: Data Types

- Commonly used data types
  - int – whole numbers
  - double – decimal numbers
  - char – a single symbol
  - string, i.e. text – 0 or more characters (more later)
  - bool, i.e. true or false