# PIC 10A: Week 3a

Section 1C, Winter 2016
Prof. Michael Lindstrom (TA: Eric Kim)
v1.0

# Announcements

- HW2 due this Wednesday (11 PM)
    - UPDATE: Professor added a page 2 to the pdf that has big hints!
    - [http://www.math.ucla.edu/~mikel/teaching/pic10a/work/](http://www.math.ucla.edu/~mikel/teaching/pic10a/work/)
        - *Note: Username/password can be found on CCLE*
-
-

# Today

- Variables
- More on Data Types
  - int, double, char, bool
- User Input
- HW2: Converting Binary/Decimal

# Variables

```
cout << "Year:" << 2016;
```

```
Output:
Year: 2016
```

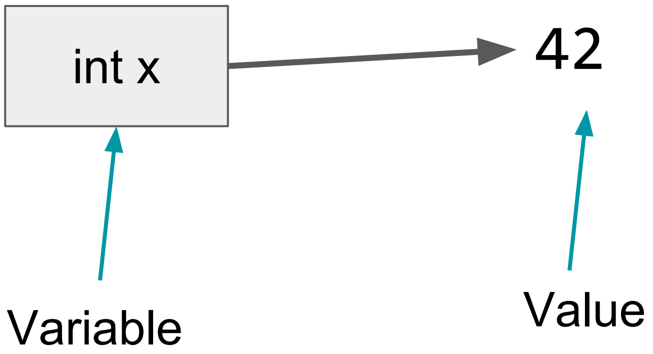
```
int year = 2016;
cout << "Year:" << year;
```

```
Output:
Year: 2016
```

Variables allow us to keep track of values by **name**.

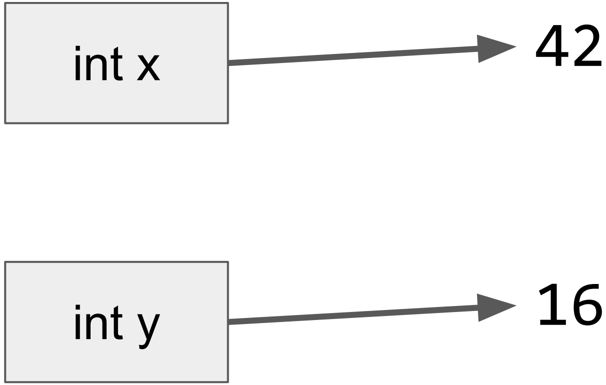
# Visualizing Variables

```
int x = 42;
```



# Visualizing Variables

```
int x = 42;
int y = 16;
```



int x

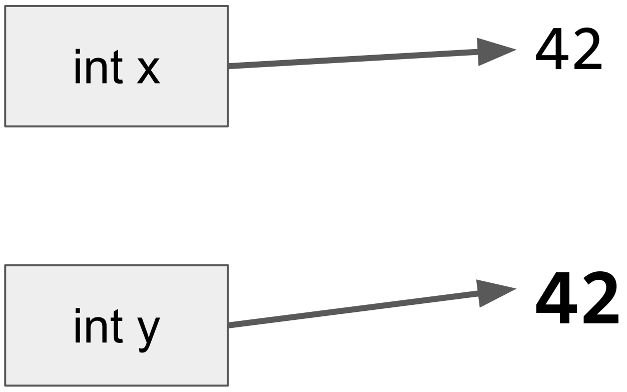
42

int y

16

# Visualizing Variables

```
int x = 42;
int y = 16;
y = x;
cout << "y:"<< y;
```

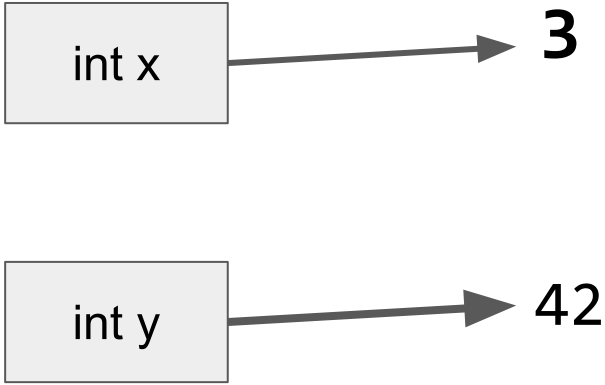


**Output:**

```
y:42
```

# Visualizing Variables

```
int x = 42;
int y = 16;
y = x;
cout << "y:"<< y;
cout << endl;
x = 3;
cout << "x:"<< x;
cout << endl;
cout << "y:"<< y;
```



int x

3

int y

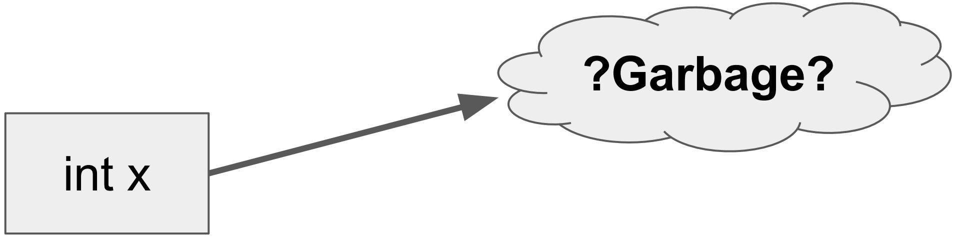
42

**Output:**

```
y:42
x:3
y:42
```

# Declaring Variables

```
int x;
```



Declares that a variable x of type int exists.

**Warning**: Since x was not set to any value (initialized), x will point to some "garbage" value. Don't use uninitialized variables!

In Visual Studio 2013, using *uninitialized variables* is a **compilation error**.

# Uninitialized Variables

```
int x;
cout << "x is: " << x;
```

This code will **not** compile, because we are trying to use an uninitialized variable.

# Initializing Variables

```
int x;
```

**Declare** variable x

```
x = 42;
```

**Initialize** variable x to have value 42

```
int x = 42;
```

Declare **and** initialize x

# Multiple Declarations

```
int x, y, z;
x = 3;
y = 5;
z = 7;
```

Declare several variables at once

```
double x = 3, y = 1;
```

Declare **and** initialize variables

```
int a, b = 42, c;
a = 1;
c = 8;
```

Can mix and match.

Note: All multiple-declared variables are the **same type**.

# Order of Evaluation

```
int x = 2, y = 5;
x = x + y + 1;
```

**Question**: What is the final value of x?

**Answer**: 8

```
x = x + y + 1;
=> x = 2 + 5 + 1;
=> x = 8;
```

When evaluating an assignment statement:
(1) Evaluate the right-hand-side (RHS)
(2) Assign the LHS to the RHS's value

# Mixing Data Types (int, double)

- Rule of Thumb: When operating on both int's and double's, the resulting value's type is *upgraded* to the **larger/more-expressive** type
  - Example: double can handle more values than int

```
int x = 3;
double y = 4.2;
cout << x + y;
```

Type was upgraded to double

**Question**: What is the output?

**Answer**: 7.2

# Data Type Exercises

```
int x = 3;
int y = 4.2;
cout << x + y;
```

4.2 is truncated to 4 when assigning to an int type

**Output:** 7

```
int x = 3;
double y = 4.2;
double z  =  x + y;
cout << z;
```

Type upgraded to double

**Output:** 7.2

# Data Type Exercises

```
int a = 3;
cout << a/2 << endl;
cout << a/2. << endl;
```

**Output:**

```
1
1.5
```

a/2 is dividing int by an int. Final type is an **int**. Truncate 1.5 to 1.

Result: a/2 -> 1

a/2. is dividing int by a **double**. Final type is a **double**.

Result: a/2 -> **1.5**

Note: 2. is shorthand for 2.0

# Casting (static_cast)

- Can explicitly tell compiler to treat a value as a certain type (ie int or double)

```
int x = 3;
double y = 4.2;
cout << x + y;
```

Type is *implicitly* upgraded to double

**Output for Both**: 7.2

```
int x = 3;
double y = 4.2;
cout << static_cast<double>(x + y);
```

***Explicitly*** treat value as a double

# static_cast

**Syntax:** `static_cast<NEWTYPE>(<EXPR>);`

**Example:**

```
int x = 1;
cout << x / 2 << endl;
cout << static_cast<double>(x) / 2 << endl;
```

**Output:**

```
0
0.5
```

# Exercise: static_cast

```
int x = 2;
cout << static_cast<double>(x / 4) << endl;
cout << static_cast<int>(x / 4.0) << endl;
cout << x / static_cast<double>(4) << endl;
```

**Question**: What is the output?

**Answer:**

```
0
0
0.5
```

# char

- Used to store single characters
- Use single quotes to define char's

```
char c1 = 'E';
char c2 = 'K';
cout << "My initials are: " << c1 << c2;
```

**Output:**

```
My initials are: EK
```

# char: Single vs Double Quotes

- Careful - don't use double-quotes for char's!

```
char c1 = "E";
char c2 = 'K';
cout << "My initials are: " << c1 << c2;
```

**Compiler error:** complains that you can't assign a char to something in double-quotes.

# bool

- Boolean. Data type used to store either *true* or *false*.
- Example:

```
bool mybool1 = true;
bool mybool2 = false;
cout << "mybool1: " << mybool1 << endl;
cout << "mybool2: " << mybool2;
```

**Output:**

```
mybool1: 1
mybool2: 0
```

Note: Very common for programming languages to treat "true" as 1, and "false" as 0.

We'll likely use bool more when we learn about if statements, for loops, and while loops.

# cin: Getting User Input

- Can ask for user input using `cin`: Console Input
  - Defined by <iostream> library (A C++ standard library)
- Example:

```
int myage;
cout << "What is your age?" << endl;
cin >> myage;
cout << "You are " << myage << " years old.";
```

Try it out in Visual Studio!

# Chaining cin

- Like cout, one can chain together multiple cin's

```
int x, y;
cin >> x >> y;
```

User can input separate values in *two* different ways:

**Option 1**: Separate values by *spaces*

```
42 9<ENTER>
```

**Option 2**: Separate values by *newlines*

```
42<ENTER>9<ENTER>
```

# Demo: Using cin in a program

# Binary/Decimal

| Decimal (Base 10) | Binary (Base 2) |
|---|---|
| 3 | 0011 |
| 2 | |
| | 1000 |
| 15 | |
| | 1001 |
| | 0111 |

Fill in the table, converting to/from decimal/binary as necessary.

*[From discussion 2b problems, question 4]*

# Binary/Decimal

| Decimal (Base 10) | Binary (Base 2) |
|---|---|
| 3 | 0011 |
| 2 | 0010 |
| 8 | 1000 |
| 15 | 1111 |
| 9 | 1001 |
| 7 | 0111 |

*[From discussion 2b problems, question 4]*

# HW2: Binary/Decimal Conversion Tips

- We are only working with decimal values from 0 to 31
- **Question**: How many binary digits do we need to represent all integers from 0 to 31?
    - 5 binary digits
    - 0 in decimal is 00000 in binary
    - 31 in decimal is 11111 in binary
        - 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 16 + 8 + 4 + 2 + 1 = 31
- So, can write code to only deal with 5 binary digits
    - Note: Writing a program to allow arbitrary integers requires additional programming mechanisms that we haven't learned yet (ie for-loops, if-stmts)

# HW2: Binary to Decimal

- **Goal**: Convert binary (10100) to decimal (20)

```
1*2^4  +  0*2^3  +  1*2^2  +  0*2^1  +  0*2^0
=> 16 + 0 + 4 + 0 + 0
=> 20
```

**How to automate this?**

# HW2: Binary to Decimal

- **Goal**: Convert binary (10100) to decimal (20)

```
1*2^4  +  0*2^3  +  1*2^2  +  0*2^1  +  0*2^0
=> 16 + 0 + 4 + 0 + 0
=> 20
```

Main Idea:

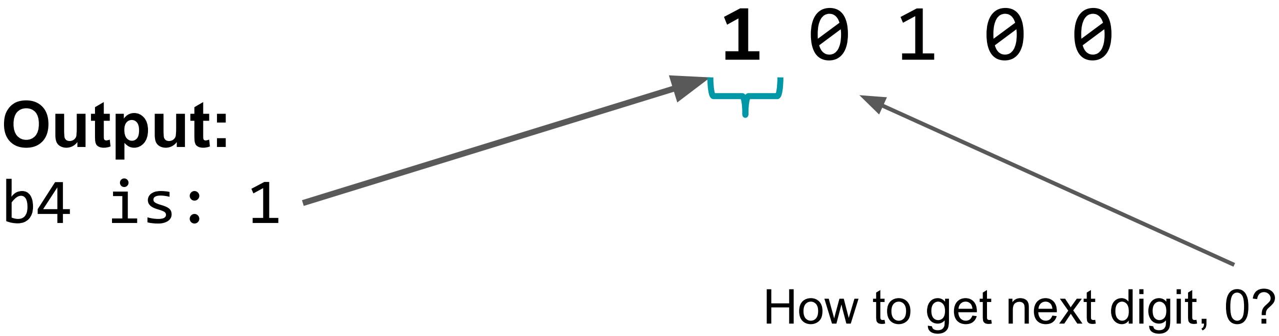
1. Use division by powers of 10 to "select" the left-most digit
2. Then, subtract the value of that left-most digit, and repeat.

# HW2: Binary to Decimal

Note: pow() is a function defined in the <cmath> library.

- **Goal**: Convert binary (10100) to decimal (20)

```
int xbin; // value in binary, ie 10100 (stored as decimal)
cin >> xbin;
int b4 = xbin / pow(10, 4);
cout << "b4 is: " << b4;
```



```
1 0 1 0 0
```

**Output:**

```
b4 is: 1
```

How to get next digit, 0?

# HW2: Binary to Decimal

- **Goal**: Convert binary (10100) to decimal (20)

```
int xbin; // value in binary, ie 10100 (stored as decimal)
cin >> xbin;
int b4 = xbin / pow(10, 4);
cout << "b4 is: " << b4;
int xbintmp = xbin - (b4*pow(10,4));
int b3 = xbintmp / pow(10, 3);
cout << "b3 is: " << b3;
```

```
10100 - 10000
=> 0 0 1 0 0
```

**Output:**

```
b4 is: 1
b3 is: 0
```

Repeat to get b2, b1, b0.
Then, use b5, b4, b3, b2, b1
to output the **decimal value**!

# HW2: Decimal to Binary

- Very similar idea as binary to decimal
- See page 2 of the HW2 pdf for a step-by-step hint
  - PDF was updated over the weekend
- Good luck!