Binary and Hexadecimal Numbers

ITP 165 – Fall 2015
Week 5, Lecture 2

Decimal Numbers

- In decimal (base 10), every digit can range from 0 to 9
- 150 in decimal could be thought of as…

Digit	1	5	0
Place	$10^2 = 100$	$10^1 = 10$	$10^0 = 1$
Decimal Value	1 * 100 = 100	5 * 10 = 50	0 * 1 = 0

- So 150 in decimal is equal to the value of 100 + 50 + 0 = 150 in decimal (well, it should be!!)

Binary

- Binary numbers are in ***base 2***
- This means that every digit can only be a 1 or a 0
- So for example, 101 in binary would be represented as:

Digit	1	0	1
Place	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
Decimal Value	1 * 4 = 4	0 * 2 = 0	1* 1 = 1

- So 101 in binary is 4 + 0 + 1 = 5 in decimal
- Internally, computers use binary to represent all of their numbers

Powers of 2

- In order to easily convert from decimal to binary, you need to know the powers of 2

Power	Value
2^0	1
2^1	2
2^2	4
2^3	8
2^4	16
2^5	32
2^6	64
2^7	128
2^8	256

Converting from Decimal to Binary

1. Determine the largest power of 2 that fits inside the decimal number
2. Subtract that power of 2 from the decimal number, and put a 1 in the place corresponding to that power of 2
3. Repeat steps 1 and 2 until the decimal number is 0
4. Any places in the binary number that don't have 1s in them should have 0s

Example: Convert 34 to binary

1. The largest power of 2 that fits in 34 is 32
2. 34 – 32 = 2. The place that corresponds to 32 is 2^5

Power	Value
2^0	1
2^1	2
2^2	4
2^3	8
2^4	16
2^5	32
2^6	64
2^7	128
2^8	256

Digit	**1**					
Place	2^5	2^4	2^3	2^2	2^1	2^0

3. 2 != 0, so continue

Example: Convert 34 to binary, Cont'd

1. The largest power of 2 that fits in 2 is 2
2. 2 – 2 = 0. The place corresponding is 2^1

Power	Value
2^0	1
2^1	2
2^2	4
2^3	8
2^4	16
2^5	32
2^6	64
2^7	128
2^8	256

Digit	**1**				**1**	
Place	2^5	2^4	2^3	2^2	2^1	2^0

3. 0 == 0, so stop

Example: Convert 34 to binary, Cont'd

4. Fill in zeroes for all remaining digits

Digit	**1**	0	0	0	**1**	0
Place	2^5	2^4	2^3	2^2	2^1	2^0

Power	Value
2^0	1
2^1	2
2^2	4
2^3	8
2^4	16
2^5	32
2^6	64
2^7	128
2^8	256

So 34 in decimal is 100010 in binary

Bits and Bytes

- A ***bit*** is a single digit in binary

- In theory, a "64-bit" computer can support 2^{64}, or 16 *exabytes* (16 billion gigabytes) of memory*

Bytes

- A ***byte*** is 8 bits

- If we only have positive numbers, a single byte can represent decimal values from 0 to 255 (or 2^8 total permutations)

Writing out a byte

- We typically show all 8 digits for a byte, even if there are leading zeroes

- For example, 34 in decimal is 100010 in binary, so if we were representing it as a byte, we would write it like so:

0	0	1	0	0	0	1	0

The unsigned type modifier

- `unsigned` is a ***type modifier*** that works on whole number types

- An `unsigned` number can only be >= 0

- A type modifier goes in the declaration, right before the type name, like so:

```
// This number can't be negative
unsigned int positiveNum = 0;
```

int and bytes

- An `int` is represented by 32 bits or 4 bytes

- This means that an `unsigned int` can range from 0 to $(2^{32} – 1)$

- ***Or in other words***…from 0 to 4,294,967,295

char type

- A `char` is a whole number type that's represented by a ***single byte***

- So an `unsigned char` can range from 0 to ($2^8 - 1$)
- That is to say, from 0 to 255

```
// This number can range from 0 to 255
unsigned char smallerNum = 0;
```

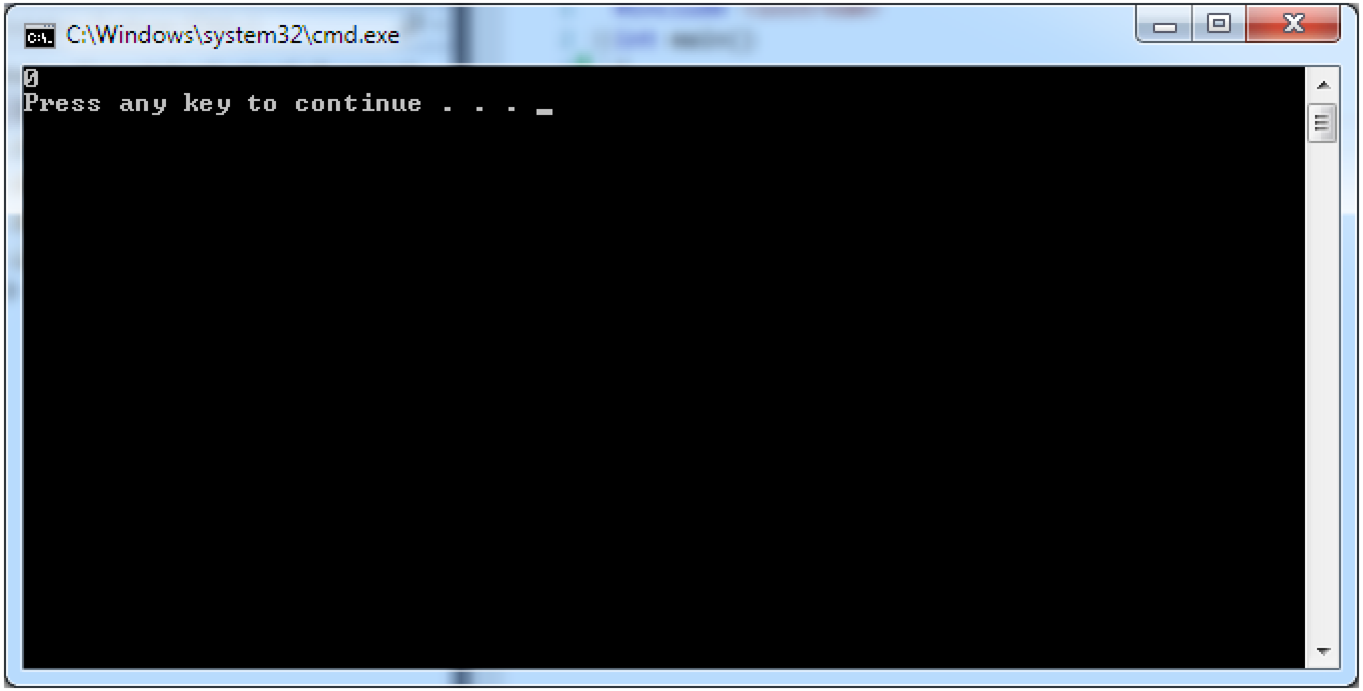
Overflow

- Since an `unsigned char` can only range from 0 to 255, what happens with the following code?

```
unsigned char test = 255;
test++;
// The cast to int is just so it displays;
// it doesn't change the value of test.
std::cout << int(test) << std::endl;
```

Overflow, Cont'd

- With the code on the previous slide:



Overflow, Cont'd

- Start with 255

1	1	1	1	1	1	1	1

Overflow, Cont'd

- Add 1… (for the ++)

	1	1	1	1	1	1	1	1
+	0	0	0	0	0	0	0	1

Overflow, Cont'd

	1	1	1	1	1	1	1	1
+	0	0	0	0	0	0	0	1
=	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

- The carry wiped out all the ones!!
- This is called ***overflow***

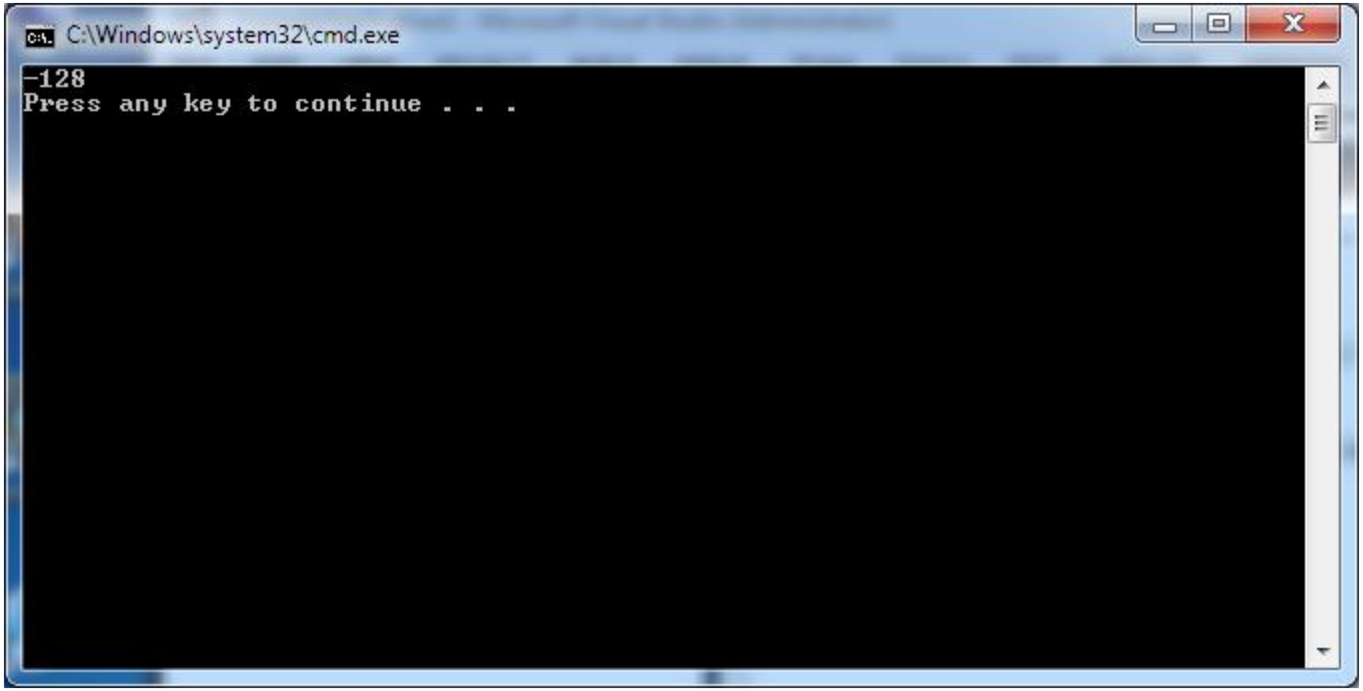
Overflow for signed numbers

- What happens with the following code?

```
// char can range from -128 to +127
char test = 127;
test++;
std::cout << int(test) << std::endl;
```

Overflow for signed numbers, Cont'd

- Same thing as before, except now it overflows to -128



Hexadecimal

- Hexadecimal (or hex) is ***base 16***, so the digits range from 0 to F

Hex Digit	Decimal Value
0 – 9	(Same)
A	10
B	11
C	12
D	13
E	14
F	15

Converting from Binary to Hex

- The easy way is to break the binary number into groups of 4

- So for example, 45 written in binary is 00101101
- Then broken into two groups of 4:

0	0	1	0	1	1	0	1

Converting from Binary to Hex, Cont'd

- Treat each group of 4 bits as a separate 4-bit binary number, and convert each group of 4 into decimal

0	0	1	0	1	1	0	1
2				8+4+1=13			

Converting from Binary to Hex, Cont'd

- Finally, convert each decimal number to the corresponding hex digit

0	0	1	0	1	1	0	1
2				8+4+1=13			
2				D			

- So 45 is 00101101 in binary, and 2D in hex

Hex in C++ Code

- We can write a number in hex by using `0x` in front of the number

- For example:

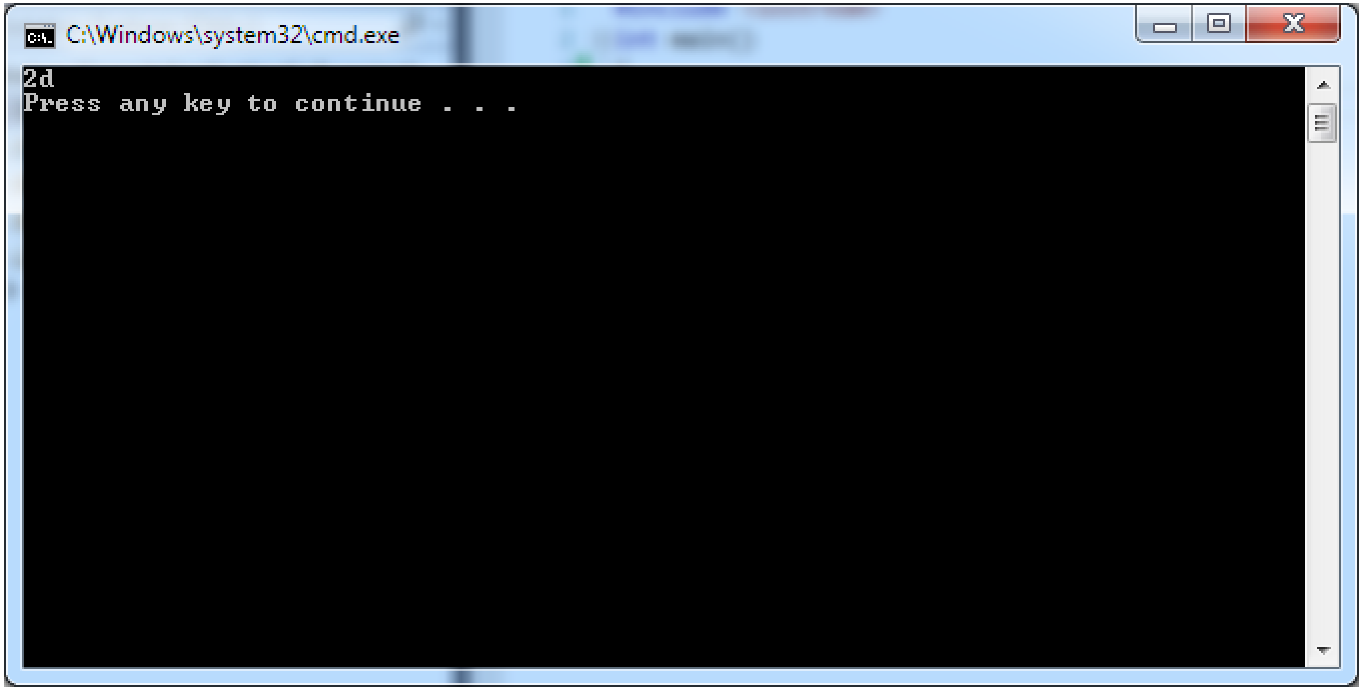
```
unsigned int x = 45;
unsigned int y = 0x2D;

// x and y are both the same, so this is true!
bool test = x == y;
```

Outputting an int in hex

- If we want to output an int in hex, you can in a cout, like so:

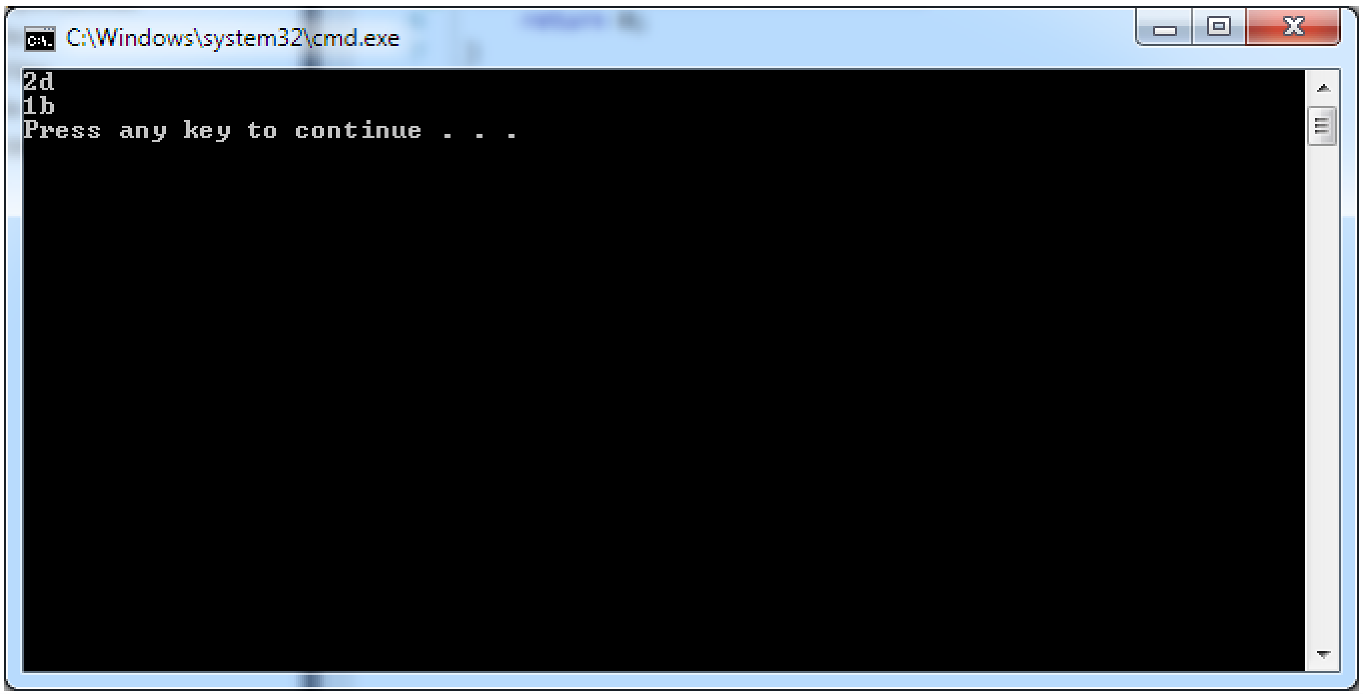
```
std::cout << std::hex << 45 << std::endl;
```



Changing back to decimal

- If you use `std::hex`, integers will be displayed in hex until you tell it to change back. For example:

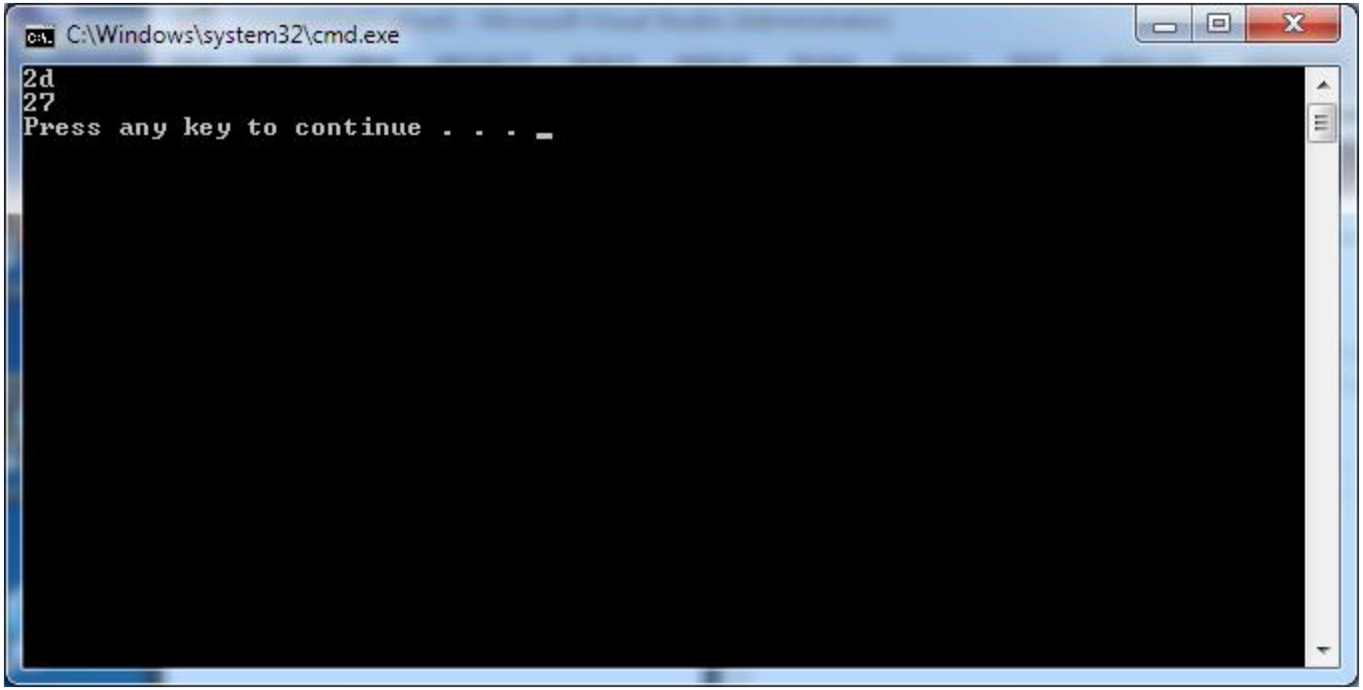
```
std::cout << std::hex << 45 << std::endl;
std::cout << 27 << std::endl;
```



Changing back to decimal, Cont'd

- You can change it back to decimal using `std::dec`:

```
std::cout << std::hex << 45 << std::endl;
std::cout << std::dec << 27 << std::endl;
```



Lab Practical #8