OPERATOR PRECEDENCE, DATA REPRESENTATION

Problem Solving with Computers-I

https://ucsb-cs16-wi17.github.io/

Announcements

- Lab02 – we have found an error in the last exercise (calculate approximate value for pi), please wait for further instructions via Piazza
- Midterm next week –Thursday (02/02)
- Study guide will be posted by tomorrow at this location: https://ucsb-cs16-wi17.github.io/exam/e01/
- Midterm will cover topics from
  - Lectures 1 to 7 (including code covered in class)
  - Labs 0 to 2
  - Homeworks 1 to 6

Note: Slides are not a replacement for the book

Review homework 4, problem 3

What is the output of the following program?

```
int x = 0;
while ( x = 2 && x < 10){
    cout << x << endl;
    x+=2;
}
```

A. Nothing is printed to output

B. Infinitely prints the number 2

C. Infinitely prints the number 1

D. Prints the following numbers to output: 2 4 6 8

Operator Precedence

Paranthesis () does not mean “Do what is inside the parenthesis first” It specifies how to explicitly bind operators to operands

```
w = x*(y+z)+y*z;

w =(x = 2) && (x < 10);
```

Operator precedence: Default binding of operators to operands in the absence of parenthesis

```
w = x * y + z + y * z;
x = a + b * c;
x = a || b && c;
x = a++ + 10;
x = 2 && x < 10;
```

Operator Precedence

```
int w, x(0);
w = (x = 2) && (x < 10);

w = (x = (2 && x) < 10));

w = (x = 2 && x < 10);
```

Precedence	Operator	Description	Associativity
1	::	Scope resolution	Left-to-right
2	a++ a--	Suffix/postfix increment and decrement	
	type() *type*{}	Functional cast	
	a()	Function call	
	a[]	Subscript	
	. ->	Member access	
3	++a --a	Prefix increment and decrement	Right-to-left
	+a -a	Unary plus and minus	
	! ~	Logical NOT and bitwise NOT	
	(*type*)	C-style cast	
	*a	Indirection (dereference)	
	&a	Address-of	
	sizeof	Size-of[note 1]	
	new new[]	Dynamic memory allocation	
	delete delete[]	Dynamic memory deallocation	
4	.* ->*	Pointer-to-member	Left-to-right
5	a*b a/b a%b	Multiplication, division, and remainder	
6	a+b a-b	Addition and subtraction	
7	<< >>	Bitwise left shift and right shift	
8	< <=	For relational operators < and ≤ respectively	
	> >=	For relational operators > and ≥ respectively	
9	== !=	For relational operators = and ≠ respectively	
10	a&b	Bitwise AND	
11	^	Bitwise XOR (exclusive or)	
12	|	Bitwise OR (inclusive or)	
13	&&	Logical AND	
14	||	Logical OR	
15	a?b:c	Ternary conditional[note 2]	Right-to-left
	throw	throw operator	
	=	Direct assignment (provided by default for C++ classes)	
	+= -=	Compound assignment by sum and difference	
	*= /= %=	Compound assignment by product, quotient, and remainder	
	<<= >>=	Compound assignment by bitwise left shift and right shift	
	&= ^= |=	Compound assignment by bitwise AND, XOR, and OR	
16	,	Comma	Left-to-right

Operator Associativity

Operator associativity: Deals with operators that are at the same precedence level or group

- Some groups associate from **left to right** e.g. Arithmetic

  x= a + b – c + d;

- Other groups associate from **right to left** e.g. Assignment

  x= y = z = 50;

Order of evaluation

- Deals with which side of an operator is evaluated first (Lt operand or Rt operand). Java/Python strictly defines Lt->Rt. C/C++ do not define order of evaluation

```
b=3;
b = b +(b=9);

a =5;
x= a + a++;

int i =4;
cout<< i++ * ++i;
```

Review homework 4, problem 3

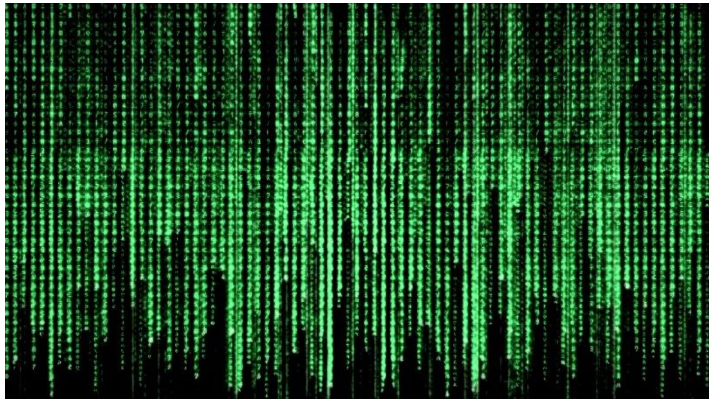
What is the output of the following program?

```
int x = 0;
while ( x = 2 && x < 10){
    cout << x << endl;
    x+=2;
}
```

A. Nothing is printed to output
B. Infinitely prints the number 2
C. Infinitely prints the number 1
D. Prints the following numbers to output: 2 4 6 8

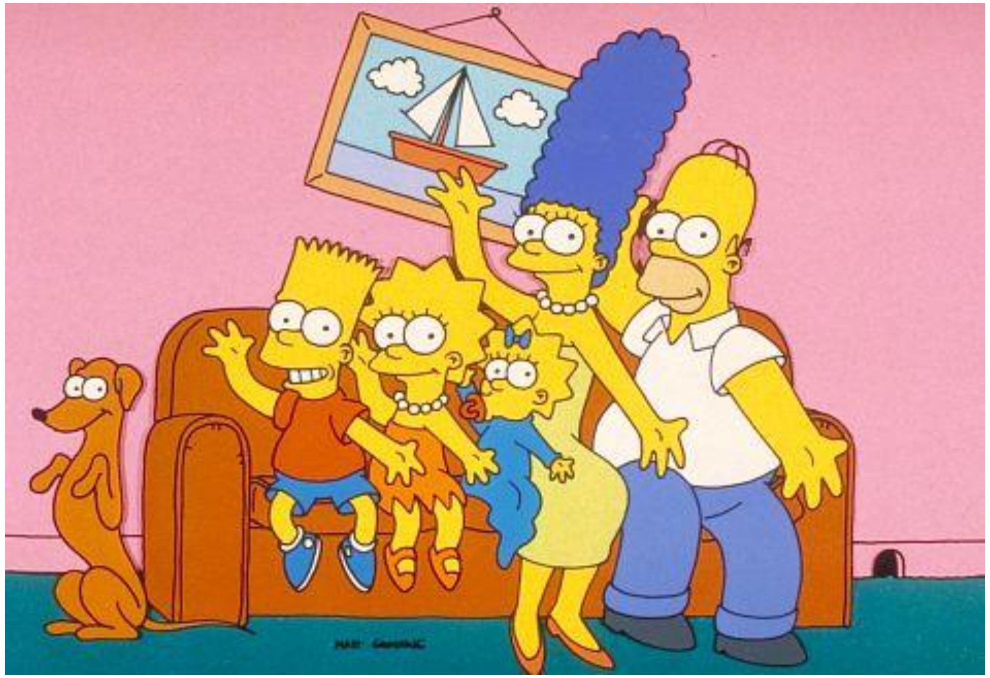
What does ‘data’ on a computer look like?

- Imagine diving deep into a computer
- Expect to see all your data as high and low voltages
- In CS we use the abstraction:
  - High voltage: 1 (true)
  - Low voltage: 0 (false)



Decimal (base ten)

- Why do we count in base ten?
- Which base would the Simpson's use?



Positional encoding for non-negative numbers

- Each position represents some power of the base

Base	Digits	Example

Why is each base important??

Binary representation (base 2)

- On a computer all data is stored in binary
- Only two symbols: 0 and 1
- Each position is called a *bit*
- *For example:*

0 1 0 1

101_5 = ? In decimal

A. 26

B. 51

C. 126

D. 130

Generalized positional encoding

- Polynomial expansion

External vs. Internal Representation

- External representation:
  - Convenient for programmer
- Internal representation:
  - Actual representation of data in the computer’s memory and registers: Always binary (1’s and 0’s)

Converting between binary and decimal

$1\ 0\ 1\ 1\ 0_2$ = ? In decimal

Decimal to binary: $34_{10} = ?_2$

Hex to binary

- Each hex digit corresponds directly to four binary digits
- Programmers love hex, why?

$35AE_{16}$ = ? In binary

00	0	0000
01	1	0001
02	2	0010
03	3	0011
04	4	0100
05	5	0101
06	6	0110
07	7	0111
08	8	1000
09	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Binary to hex: 1000111100

A. 8F0

B. 23C

C. None of the above

Hexadecimal to decimal

$$25B_{16} = ?\ \text{Decimal}$$

Hexadecimal to decimal

- Use polynomial expansion

- $25B_{16}$ = 2*256 + 5*16 + 11*1 = 512 + 80 + 11
  = 603
- Decimal to hex: $26_{10} = ?_{16}$

Decimal vs. Hexadecimal vs. Binary

Examples:

1010 1100 0011 (binary)
= 0xAC3

10111 (binary)
= 0001 0111 (binary)
= 0x17

0x3F9
= 11 1111 1001 (binary)

00	0	0000
01	1	0001
02	2	0010
03	3	0011
04	4	0100
05	5	0101
06	6	0110
07	7	0111
08	8	1000
09	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

BIG IDEA: Bits can represent anything!!

- Logical values?
  - 0 ⇒ False, 1 ⇒ True
- colors ?
- Characters?
  - 26 letters ⇒ 5 bits ($2^5 = 32$)
  - upper/lower case + punctuation
    ⇒ 7 bits (in 8) ("ASCII")
  - standard code to cover all the world's languages ⇒ 8,16,32 bits ("Unicode")
    www.unicode.com
- locations / addresses? commands?

Red

Green

Blue

- MEMORIZE: N bits ⇔ at most 2^N things

Data types

The data type of a variable determines the:

- exact representation of variable in memory
- number of bits available (finite and fixed)
  - range of values that can be correctly represented

What is the largest positive value that can be stored in a byte if we used the positional encoding scheme discussed in class today?

A. 127

B. 128

C. 255

D. 256

Generalize to N bits

Next time

- Under the hood of program compilation
- Separate compilation with makefiles