# CMSC201
# Computer Science I for Majors

## Lecture 20 – Project 3
## and Miscellaneous Topics

# Last Class We Covered

- Dictionaries
  - Creating
  - Accessing
  - Manipulating
  - Methods
- Hashing
- Dictionaries vs Lists

# Any Questions from Last Time?

# Today’s Objectives

- To understand more about how data is represented inside the computer
  - Binary numbers
- To see the benefits of short circuit evaluation
- To discuss details of Project 3
  - How many boards to have?

# Binary Numbers

# Binary Numbers

- Computers store all information (code, text, images, sound,) as a binary representation
  - "Binary" means only two parts: 0 and 1
- Specific formats for each file help the computer know what type of item/object it is
- But why use binary?

# Decimal vs Binary

- Why do we use decimal numbers?
  - Ones, tens, hundreds, thousands, etc.
- But computers don’t have fingers…
  - What do they have instead?
- They only have two states: “on” and “off”

# Decimal Example

- How do we represent a number like 50,932?

| ten thousands | thousands | hundreds | tens | ones |
|---|---|---|---|---|
| 5 | 0 | 9 | 3 | 2 |
| $10^4$ | $10^3$ | $10^2$ | $10^1$ | $10^0$ |

$$
\begin{array}{lcr}
2 \times 10^0 & = & 2 \\
3 \times 10^1 & = & 30 \\
9 \times 10^2 & = & 900 \\
0 \times 10^3 & = & 0000 \\
5 \times 10^4 & = & 50000 \\
 & & \text{------} \\
\text{Total:} & & 50932
\end{array}
$$

Decimal uses 10 digits, so…

# Another Decimal Example

| | | | | |
|---|---|---|---|---|
| 6 | 7 | 4 | 9 | 3 |
| $10^4$ | $10^3$ | $10^2$ | $10^1$ | $10^0$ |
| 10000 | 1000 | 100 | 10 | 1 |
| 60000 | 7000 | 400 | 90 | 3 |

60000+7000+400+90+3 = 67493

# Binary Example

- Let’s do the same with 10110 in binary

| sixteens | eights | fours | twos | ones |
|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 |
| $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |

$0 \times 2^0 = 0$
$1 \times 2^1 = 2$
$1 \times 2^2 = 4$
$0 \times 2^3 = 0$
$1 \times 2^4 = 16$

--

Total: 22

Binary uses 2 digits, so our base isn’t 10, but…

# Binary to Decimal Conversion

- Step 1: Draw Conversion Box
- Step 2: Enter Binary Number
- Step 3: Multiply
- Step 4: Add

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| 128 | 0 | 0 | 0 | 8 | 4 | 0 | 1 |

$$128 + 0 + 0 + 0 + 8 + 4 + 0 + 1 = 141$$

# Exercise: Converting From Binary

- What are the decimals equivalents of…

  `101`

  `1111`

  `100000`

  `101010`

  `0010 1010`

  `1000 0000`

Longer binary numbers are often broken into blocks of four digits for the sake of readability

# Exercise: Converting From Binary

- What are the decimals equivalents of…

```
101       = 4+0+1         = 5
1111      = 8+4+2+1       = 15
100000    = 32+0+0+0+0+0  = 32
101010    = 32+0+8+0+2+0  = 42
0010 1010 = 32+0+8+0+2+0  = 42
1000 0000 = 128+...+0+0   = 128
```

# Decimal to Binary Conversion

- Step 1: Draw Conversion Box
- Step 2: Compare decimal to highest binary value
- Step 3: If binary value is smaller, put a 1 there and subtract the value from the decimal number
- Step 4: Repeat until 0

Convert 163 to binary

| $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
|---|---|---|---|---|---|---|---|
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |

163-128 = 35 35-32 = 3 3-2=1 1-1=0

# Converting to Binary

- What are the binary equivalents of…

**9**

**27**

**68**

**216**

**255**

# Converting to Binary

- What are the binary equivalents of…

```
9     = 1001 (or 8+1)
27    = 0001 1011 (or 16+8+2+1)
68    = 0100 0100 (or 64+4)
216   = 1101 1000
        (or 128+64+16+8)
255   = 1111 1111
        (or 128+64+32+16+8+4+2+1)
```

# Binary Tips and Tricks

- Some “sanity checking” rules for conversions:

1. Binary can only be 1 or 0
   – If you get “2” of something, it’s wrong
2. Odd numbers must have a 1 in the ones column
   – Why? (And what’s the rule for even numbers?)
3. Each column’s value is the sum of all of the previous columns (to the right) plus one
   – In decimal, what column comes after 999?

# “Short Circuit” Evaluation

# Review: Complex Expressions

- We can put multiple operators together!

```
bool4 = a and (b or c)
```

- What does Python do first?
  - Computes **`(b or c)`**
  - Computes **`a and`** the result

This isn't strictly true!

# Short Circuit Evaluation

- Python tries to be efficient (*i.e.*, lazy), and so it won't do any more work than necessary
  - If the remainder of an expression won't change the outcome, Python doesn't look at it

- This is called "short circuiting"
  - It's a powerful tool, and can simplify the conditionals in your programs

# Short Circuit Evaluation – Rules

- For obvious reasons, short circuiting behaves differently for **`and`** and **`or`** statements

- "**`and`**" statements short circuit as soon as an expression evaluates to **`False`**
- "**`or`**" statements short circuit as soon as an expression evaluates to **`True`**

# Short Circuiting – `and`

- Notice that in the expression:

```
bool1 = a and (b or c)
```

- If **`a`** is **`False`**
- The rest of the expression doesn't matter

- Python will realize this, and if **`a`** is **`False`** won't bother with the rest of the expression

# Short Circuiting – `or`

- Notice that in the expression:

```
bool1 = a or (b or c)
```

- If **`a`** is **`True`**
- The rest of the expression doesn't matter

- Python will realize this, and if **`a`** is **`True`** won't bother with the rest of the expression

# Causing Errors

- This can lead to "new" errors in old code

```
>>> a = True
>>> # Variables b and c not defined
>>> a or (b and c)
True

>>> a = False
>>> a or (b and c)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'b' is not defined
```

Python stopped at the "**or**", so it never saw **b** or **c**

# Simplifying Conditionals

- Order matters! You can use short circuiting to control what statements are reached

- While checking the validity of input, if the user can also enter a “Q” to quit

```
if num != QUIT and int(num) > MIN_VAL:
    return num
```

This will only be reached if num is not “Q”, so the cast to int() won’t cause a problem

# Project 3

# Do Not Cheat on Project 3

- Yes, this project has been given before
  - Yes, in this class
  - Yes, we have all of the old projects to compare it to
- Yes, this project has solutions on the internet
  - Yes, we have copies of all of them
  - Yes, we will go looking for new ones after it’s due
- Yes, you could pay someone else to do it
  - Yes, we know of the sites where you can get this done
  - Yes, we will spot “elegant” code that you didn’t write

# Boards in Project 3

- Discussed in class
- ¯_(ツ)_/¯

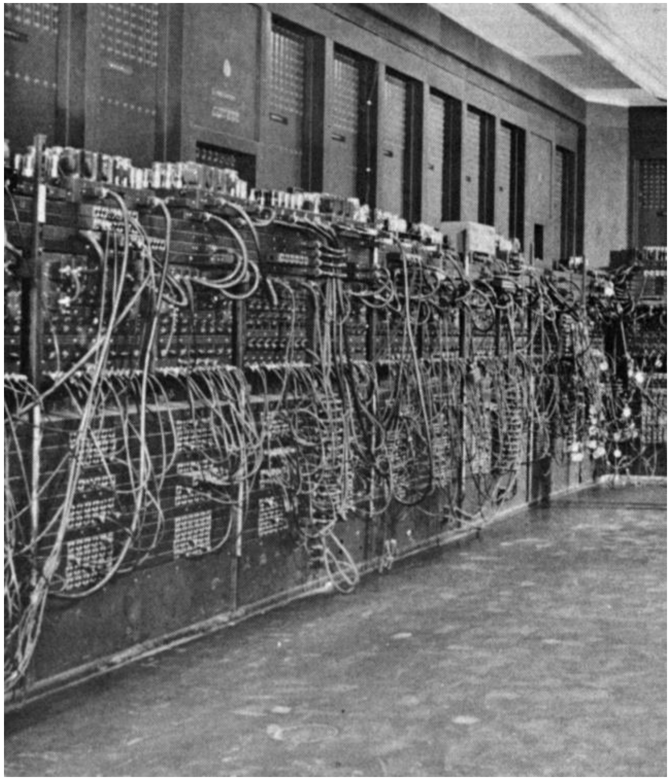
# Daily CS History

- John von Neumann
  - Creator of merge sort
    - We'll learn this soon!
  - Helped develop what is now known as "von Neumann architecture" (not all his work)
  - Created a rigorous framework for quantum mechanics
  - Developed implosion mechanism for nuclear bombs

# More Daily CS History

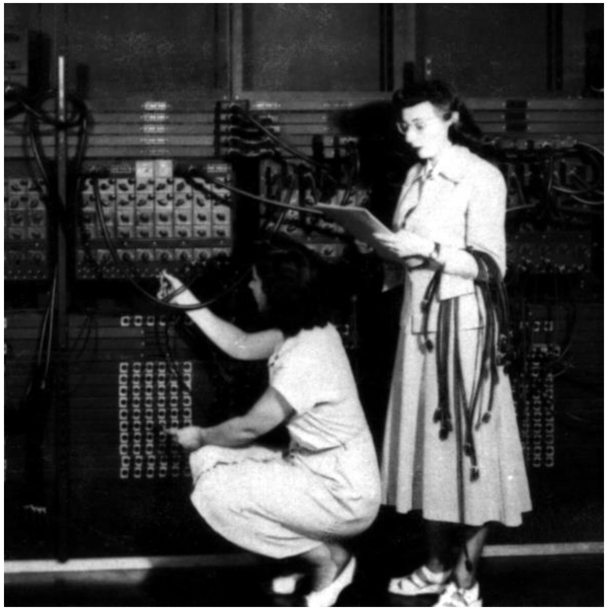
- ENIAC
  - Completed in 1946 at UPenn
    - Decommissioned in 1956
  - Computations were 2,400 times faster than humans
  - Cost $6.7 million to build
  - Meant to create artillery firing tables for the US Army
  - Also used for studying thermonuclear feasibility



# Even More Daily CS History

- ENIAC Programmers
  - Kay McNulty, Betty Jennings, Betty Snyder, Marlyn Meltzer, Fran Bilas, and Ruth Lichterman
  - These women turned abstract ideas into working, bug-free code
    - First program run on ENIAC had a million individual punchcards
  - Programming was seen back then as “easy” work, akin to typing up a handwritten letter



# Announcements

- Project 3 design is due on Friday, May 3rd
  - Project itself is due on Friday, May 10th
- Survey #3 out on Monday, May 6th

- Course evaluations are (not out yet)
- Final exam is when?
  - Friday, May 17th from 6 to 8 PM

# Image Sources

- ASCII table (adapted from):
  - https://commons.wikimedia.org/wiki/File:ASCII-Table-wide.svg
- Generic kitten:
  - http://www.publicdomainpictures.net/view-image.php?image=87454
- Generic puppy:
  - http://www.publicdomainpictures.net/view-image.php?image=192231
- John von Neumann:
  - https://en.wikipedia.org/wiki/File:JohnvonNeumann-LosAlamos.gif
- ENIAC (adapted from):
  - https://commons.wikimedia.org/wiki/File:Eniac.jpg
- ENIAC programmers (adapted from):
  - https://commons.wikimedia.org/wiki/File:Reprogramming_ENIAC.png
- Mad emoji (adapted from):
  - https://commons.wikimedia.org/wiki/File:Twemoji_1f620.svg