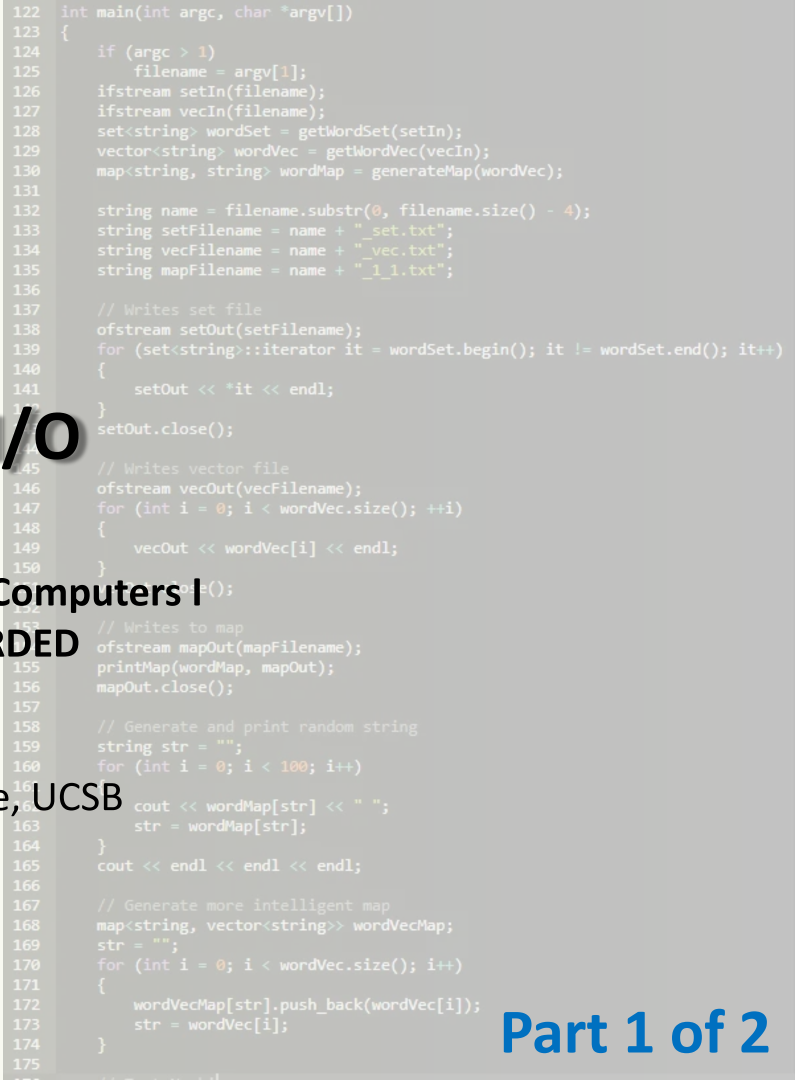
More on File I/O

CS 16: Solving Problems with Computers I
Lecture #13 PRE-RECORDED

Ziad Matni
Dept. of Computer Science, UCSB



Part 1 of 2

Lecture Outline

- More on File I/O
  - .get() and getline() for File I/O
  - Appending to output files
  - Formatting in output files
  - Use of Functions with File I/O

- Numerical Conversions
  - Decimal vs. Binary, Octal and Hexadecimal Numbers
  - How to convert between them

RECALL: Member Function .**get(char)**

- Member function of every input stream
  - i.e. it works for **cin** *and* for **ifstream**

- Reads ***one character*** from an input stream

- Stores the character read in a variable of **type char**, which is the single argument the function takes

- Does **not** use the extraction operator (>>)
- Does **not** skip whitespaces, like blanks, tabs, new lines
  - *Because these are characters too!*

Difference Between **get** and **getline** (a summary)

- Allow you to use input streams that include white-spaces
  - ***Unlike cin***, which separates inputs by white-spaces
  - Recall: white-space = space, tab, newline characters

See demo files:
changeCtoCPP.cpp
getline_example.cpp

.get	getline

`char c_fin, c_cin;`
`ifstream inf;`
 	
`inf.get(c_fin);`
`cin.get(c_cin);`
	`string fstring, cin_string;`
	`ifstream inf;`
	
	`getline(inf, fstring);`
	`getline(cin, cin_string);`

Appending Data to Output Files

- **Output** examples we've given so far ***create new files***
- If the output file that you've designated <u>already</u> <u>contained</u> data and you try to write to it again, then that data is **now lost!**
- Does C++ allow us to add data to the end of (i.e. *append*) an existing file?
  - Yes.

Appending Data to Output Files

See demo files: **AppendFile.cpp**

- To ***append*** (i.e. add) new output to the end an existing file use the constant `ios::app` defined in the **fstream** library:

```
outStream.open("important.txt", ios::app);
```

  - If the file does not exist, a new file will be created
  - **`ios::app`** is just another C++ "constant" that means "*seek to end before each write*"
    - Should only be used in this context

- There are other **fstream** methods that return the *location* in the I/O file where the next data will be, etc…
  - Helps us with customizing read and writing files
  - HAVE to be used carefully! We WILL NOT go over them in CS 16…

Formatting Output to Files

- Recall: Format output to the screen with:

```
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
```

- Similarly, you can format outputs to a file like this:

(assumes your file object is called "**out_stream**" in this example):

```
out_stream.setf(ios::fixed);
out_stream.setf(ios::showpoint);
out_stream.precision(2);
```

Can I Call a Function to do a File I/O?

- Yes!

- But there are strict rules about it:
  - Mainly this:
    data stream objects must be **passed by <u>reference</u>** into functions

See demo files:
FunctionFile.cpp

Stream Names as Arguments

- Streams can be arguments to a function
  - The function's formal parameter for the stream **must** be call-by-reference
- Example:

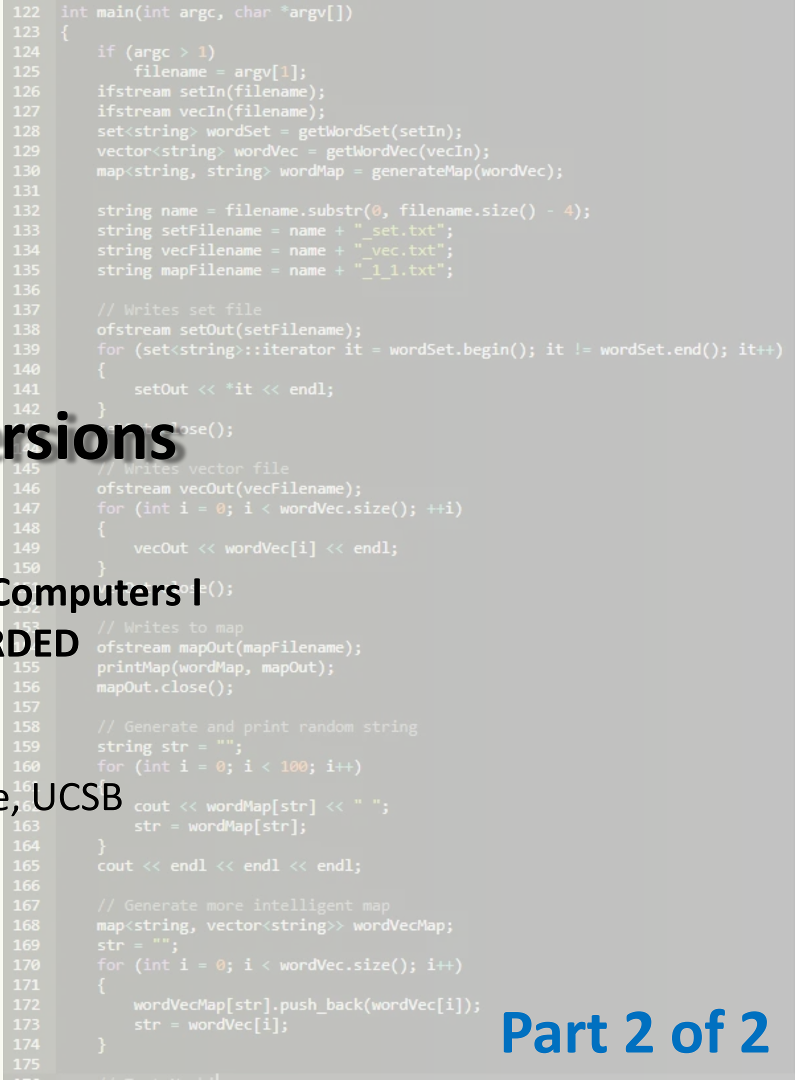
```
void makeNeat(ifstream& fileX,  ofstream& fileY);
```

END OF PART 1

Numerical Conversions

CS 16: Solving Problems with Computers I
Lecture #13 PRE-RECORDED

Ziad Matni
Dept. of Computer Science, UCSB



Part 2 of 2

Counting Numbers in Different Bases

- We "normally" count in 10s
  - Base 10: decimal numbers
  - Number symbols are 0 thru 9

- Computers count in 2s
  - Base 2: binary numbers
  - Number symbols are 0 and 1
  - Represented with **1 bit** ($2^1 = 2$)

- Other convenient bases in computer architecture:
  - Base 8: octal numbers
  - Number symbols are 0 thru 7
  - Represented with **3 bits** ($2^3 = 8$)

  - Base 16: hexadecimal numbers
  - Number symbols are 0 thru F
    - A = 10, B = 11, C = 12, D = 13, E = 14, F = 15
  - Represented with **4 bits** ($2^4 = 16$)
  - **Why are 4 bit representations convenient???**

Positional Notation in Decimal

a.k.a.: How I Learned Numbers in the 3rd Grade…

642 is: 6 hundreds, 4 tens, and 2 units

It's a number in base 10 (aka decimal)

*We can write it in **positional notation**:*

$$
\begin{array}{rlll}
6 \times 10^2 = & 6 \times 100 & = 600 & \\
+\,4 \times 10^1 = & 4 \times 10 & = +\,40 & \\
+\,2 \times 10^0 = & 2 \times 1 & = +\;\;2 & = \text{642 in base 10}
\end{array}
$$

Positional Notation

Anything → DEC

What if "642" is expressed in the base of 13?

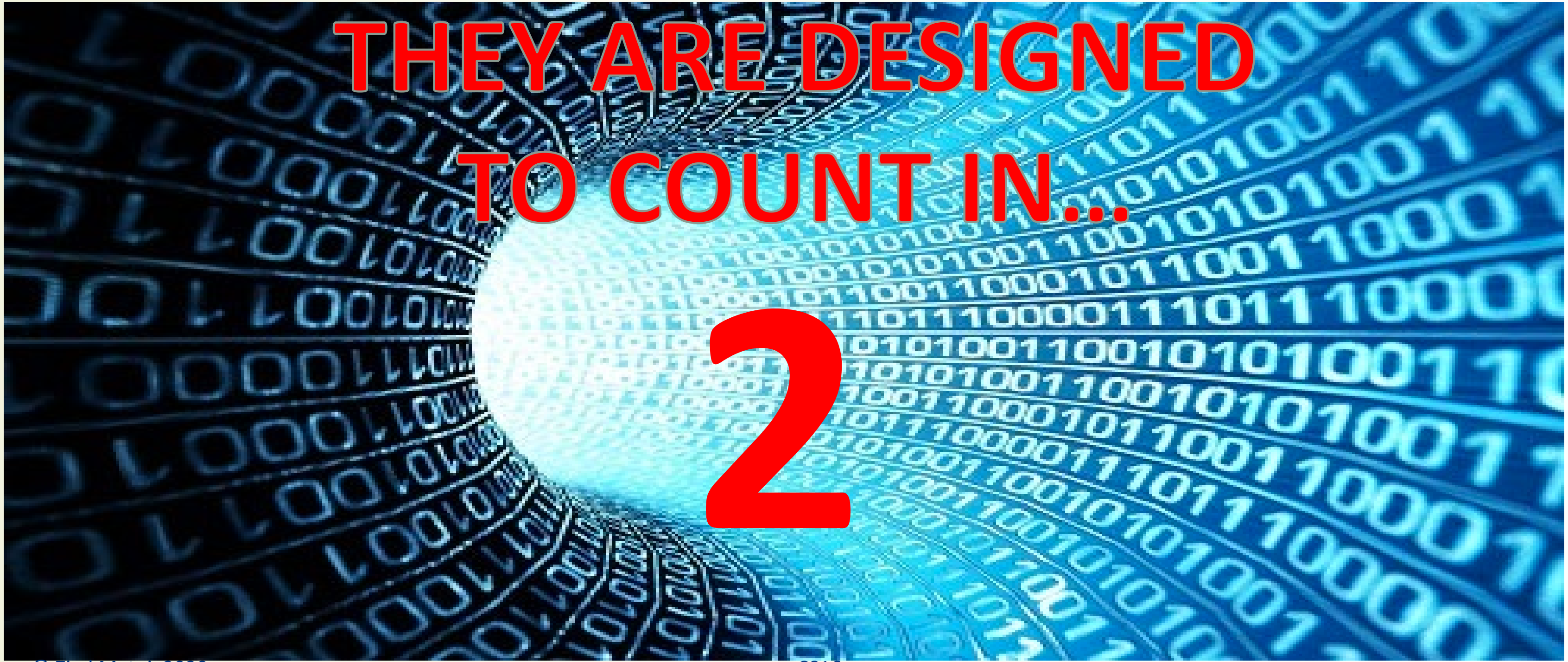
$$
\begin{aligned}
& 6 \times 13^2 &= \quad & 6 \times 169 &= 1014 \\
+\ & 4 \times 13^1 &= \quad & 4 \times 13 &= 52 \\
+\ & 2 \times 13^0 &= \quad & 2 \times 1 &= 2 \\
& & & &= 1068 \text{ in base } 10
\end{aligned}
$$

So, "642" in base 13 is equivalent to "1068" in base 10

BUT WHO COUNTS IN BASE 13???!?!?

Maybe, aliens with 13 fingers???

COMPUTERS ARE DIGITAL (Binary) MACHINES



Positional Notation in Binary

11011 in base 2 *positional notation* is:

$$
\begin{aligned}
& 1 \times 2^4 &= \; & 1 \times 16 = 16 \\
+ \; & 1 \times 2^3 &= \; & 1 \times 8 \;\; = 8 \\
+ \; & 0 \times 2^2 &= \; & 0 \times 4 \;\; = 0 \\
+ \; & 1 \times 2^1 &= \; & 1 \times 2 \;\; = 2 \\
+ \; & 1 \times 2^0 &= \; & 1 \times 1 \;\; = 1
\end{aligned}
$$

So, **1011** in base 2 is 16 + 8 + 0 + 2 + 1 = **27** in base 10

Converting Binary to Decimal

*Q: What is the decimal equivalent of the binary number **1101100**?*

A: Look for the position of the digits in the number.

This one has 7 digits, therefore positions 0 thru 6

1	**1**	**0**	**1**	**1**	**0**	**0**
64	32	16	8	4	2	1
2^6	2^5	2^4	2^3	2^2	2^1	2^0

$$
\begin{aligned}
& 1 \times 2^6 = 1 \times 64 = 64 \\
+ & 1 \times 2^5 = 1 \times 32 = 32 \\
+ & 0 \times 2^4 = 0 \times 16 = 0 \\
+ & 1 \times 2^3 = 1 \times 8 = 8 \\
+ & 1 \times 2^2 = 1 \times 4 = 4 \\
+ & 0 \times 2^1 = 0 \times 2 = 0 \\
+ & 0 \times 2^0 = 0 \times 1 = 0 \\
& = \underline{108} \text{ in base 10}
\end{aligned}
$$

Other Relevant Bases

- In Computer Science/Engineering, other binary-related numerical bases are used too.

- OCTAL: Base 8 (note that 8 is 2^3)
  - Uses the symbols: 0, 1, 2, 3, 4, 5, 6, 7

- HEXADECIMAL: Base 16 (note that 16 is 2^4)
  - Uses the symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

    (10) (11) (12) (13) (14) (15)

Converting Binary to Octal and Hexadecimal
(or any base that's a power of 2)

- Binary is 1 bit
- Octal is 3 bits ($2^3 = 8$) *octal is base 8*
- Hexadecimal is 4 bits ($2^4 = 16$) *hex is base 16*

- Use the "**group the bits**" technique
  - Always start from the *least significant digit*
  - Group every 3 bits together for bin → oct
  - Group every 4 bits together for bin → hex

Note on Notation…

- Hexadecimal numbers are often represented with a **0x** in front of them to make the reader aware that what follows is hexadecimal.
  - Example: 0x5D

- Binary numbers are often represented with a **0b** in front of them to make the reader aware that what follows is binary.
  - Example: 0b11010

Converting Binary to Octal and Hexadecimal

- Take the example: ***10100110***

…to octal:

1 0	1 0 0	1 1 0
2	**4**	**6**

246 in octal

…to hexadecimal:

1 0 1 0	0 1 1 0
10	**6**

0xA6 in hexadecimal

0
1
2
3
4
5
6
7
8
9
A
B
C
D
E
F

Decimal symbols

Octal symbols

Hex. symbols

Converting Decimal to Other Bases

Algorithm for converting number in base 10 to other bases

While (the **quotient** is not zero)

1. Divide the decimal number by the new base
2. Make the remainder the next digit to the left in the answer
3. Replace the original decimal number with the quotient
4. Repeat until your quotient is zero

EXAMPLE:

Convert the decimal (base 10) number **79** into hexadecimal (base 16)

79 / 16 = 4 R 15 *(15 in hex is the symbol "F")*

4 / 16 = 0 R 4

The answer is: **0x4F**

Converting Decimal into Binary

Convert 54 (base 10) into binary and hex:

- 54 / 2 = 27 R **0**
- 27 / 2 = 13 R **1**
- 13 / 2 = 6 R **1**
- 6 / 2 = 3 R **0**
- 3 / 2 = 1 R **1**
- 1 / 2 = $\boxed{0}$ R **1**

54 (decimal) = 0b110110 **(remember, 0b… means binary)**

= 0x36 (remember, 0x… means hex)

Sanity check:

0b110110

= 2 + 4 + 16 + 32

= 54

More Examples: Convert These Numbers!

- 0x5A7 into binary
  - Note this is 3 hex digits, i.e. 3x4 = 12 bit number
  - So, 0x5 is 0101, 0xA is 1010, 0x7 is 0111.
  - It becomes: 0b010110100111 (color coded for your convenience)

- 536$_{(octal)}$ into binary
  - This is a 3 octal digit number, i.e. 3x3 = 9 bit number
  - It becomes: 0b101011110
- Convert THAT into hexadecimal!
  - Group by 4 method
  - (1)(0101)(1110) = 0x15E

More Examples: Convert These Numbers!

- 0x5A7 into decimal
  - Easiest way is to use positional notation method
  - $5 \times 16^2 + 10 \times 16 + 7 =$ **1447**

- $536_{(octal)}$ into decimal
  - $5 \times 8^2 + 3 \times 8 + 6 =$ **350**

One More! Convert!

- $5536_{(decimal)}$ into octal
  - Best way is to use quotient-division method
  - 5536 / **8** = 692 R 0
  - 692 / 8 = 86 R 4
  - 86 / 8 = 10 R 6
  - 10 / 8 = 1 R 2
  - 1 / 8 = 0 R 1

- So, the answer is: **$12640_{(octal)}$** = $5536_{(decimal)}$
- Sanity-check:

  $1\text{x}\mathbf{8}^4 + 2\text{x}\mathbf{8}^3 + 6\text{x}\mathbf{8}^2 + 4\text{x}\mathbf{8} + 0\text{x}1 = 4096 + 1024 + 384 + 32 = 5536$ ***(works!)***

</LECTURE>