Lecture 16

Project A &

More on the workings of a computer

Project A: Minesweeper

```
   00000
   01234

 0 .....
 1 .....
 2 .....
 3 .....
 4 .....
```

Start of game with all the spaces covered

```
   00000
   01234

 0 0001.
 1 0012.
 2 111..
 3 .....
 4 .....
```

User selected 0,0. The game uncovered the space and all spaces adjacent to any uncovered zero.

```
   00000
   01234

 0 0001#
 1 00122
 2 111#1
 3 #1111
 4 11000
```

User selected 4,0. This was a mine, so the game is over and the full board is displayed (# are mines)

So…..

- We've supplied some code
  - The main
  - Display the board
  - Read from a file to load a board
- You get to fill in the rest.

Advice

- Read the specification carefully.
- Read the code *very* carefully
  - Think of it like HW2 where you had to answer questions about the code.
- Write the simpler functions first
  - Test each function as best you can.
  - Cout or the debugger can be very useful here.
  - Consider commenting out the displayMap() function so you can see your testing output.

Common errors

- I spent a fair amount of time writing this whole thing. I hit a number of really annoying bugs.
  - Most common was array bounds problems.
    - These are really hard to find.
  - Some "cut and paste" errors—mainly in my for loops.
    - for(i=0;j<100;i++) can create really annoying problems.

"tricks"

- Use –Wall a lot!
  - Saved me on a x==4 line when I wanted x=4
- If your program is crashing…
  - Use gdb (or ddd). Do the following:
    - gdb a.out
    - run
    - bt
  - The "bt" stands for backtrace and should tell you where the program crashed.
    - If it is someplace really wacky, look for array bounds problems.
      - Ask for help.

On to computer representations

Subtracting

```
  1001        1001        1111
- 0010      - 0011      - 0011
======      ======      ======
```

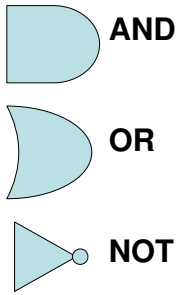
Consider adding

```
    1001    c3c2c1
                abcd
  + 0010        efgh
  ======        ====
    1011        wxyz
```

- Notice that
  - z= (d and !h) or (!d and h)
  - c_1=d and h
- Who cares?
  - It turns out we can build an adder out of “and”, “or” and “not” operations.
  - We can also build subtractors, multipliers, etc.
- **We can use these simple “ands”, “ors” and “nots” to build a computer. But we pretty much are stuck with base 2.**

Gates

- We can draw logic gates



Gates example

```
     0          c1
       1          a
  +    0     +    b
  ======     ====
       1          s
```

char

- A char is an 8-bit 2’s complement number.
  - So you can represent -128 to 127.
- But we usually use it to represent an character.
  - We map each of the 256 values to a different symbol.

Selected parts of the ASCII table

B10	B08	B16	B2		
000	000	000	00000000	NUL	(Null char.)
007	007	007	00000111	BEL	(Bell)
008	010	008	00001000	BS	(Backspace)
009	011	009	00001001	HT	(Horizontal Tab)
010	012	00A	00001010	LF	(Line Feed)
011	013	00B	00001011	VT	(Vertical Tab)
012	014	00C	00001100	FF	(Form Feed)
013	015	00D	00001101	CR	(Carriage Return)
014	016	00E	00001110	SO	(Shift Out)
037	045	025	00100101	%	(percent)
038	046	026	00100110	&	(ampersand)
039	047	027	00100111	'	(single quote)
040	050	028	00101000	(	
048	060	030	00110000	0	
049	061	031	00110001	1	
050	062	032	00110010	2	
051	063	033	00110011	3	
072	110	048	01001000	H	
073	111	049	01001001	I	

Some of the table
Is in your book (p958ish)
Easy to find on the web.