CS 161
Intro to CS I

Command-line Args/Dynamic 2-d array...
Odds and Ends...

• Keep working on Assignment #6
• Design first!!!!
• Exercise #9 due tonight!
In-class exercise #5

• Finish writing taking the command line args in any order, and all errors associate with making sure you get –r # -c # -m #

```
a.out -r 3 -c 3 -m 2
```

```
a.out -m 2 -c 3 -r 3
```

Errors:
```
a.out -m 2 -c 3 -r 3 -l 15
```
```
a.out -m 2
```
```
a.out -m 1 2 -c 3 -r 3
```
```
a.out -m -c -r 3 3 2
```
```cpp
#include <iostream>
#include <stdlib.h>
using namespace std;

int main(int argc, char *argv[])
{
    cout << argc << endl; // always at least one
    cout << argv[0] << endl; // name of your prog is first arg
    cout << argv[0][0] << endl;
    cout << argv[0][1] << endl;

    // process option value pairs in any order and find bad options
    for(int i=1; i<argc; i+=2)
    {
        if(argv[i][0]=='-' & argv[i][1]=='r' & argv[i][2]=='\0')
            cout << atoi(argv[i+1]) << endl;
        else
            cout << "usage: exe -r number" << endl;
    }
    exit(1);

    return 0;
}
```

```
#include <iostream>
#include <stdlib.h>
using namespace std;

int main(int argc, char *argv[]) {
    cout << argc << endl; // always at least one
    cout << argv[0] << endl; // name of your prog is first arg
    cout << argv[0][0] << endl;
    cout << argv[0][1] << endl;

    // process option value pairs in any order and find bad options
    for (int i = 1; i < argc; i += 2) {
        if (argv[i][0] == '-' && argv[i][1] == 'r' && argv[i][2] == '\0')
            cout << atoi(argv[i + 1]) << endl;
        else
            cout << "usage: exe -r number" << endl;
    }
    exit(1);

    return 0;
}
```
Static vs. Dynamic 2-D arrays...
Revisit Demo...

```cpp
for(int i=0; i<ROWS; i++)
    for(int j=0; j<COLS; j++)
        a[i][j]=100;

cout << "where does a live: " << &a << endl;
cout << "where does 1st row pointer live: " << a << endl;
cout << "where does 1st row live/1st element: " << &a[0] << endl;
cout << "where does 1st row live/1st element: " << &a[0][0] << endl;
cout << "contents of 1st element: " << a[0][0] << endl;
cout << "where does 2nd row live/2nd element: " << &a[0][1] << endl;
cout << "where does 1st row live/3rd element: " << &a[0][2] << endl;
cout << "where does 2nd row live/1st element: " << &a[1][0] << endl;
cout << "where does 2nd row live/1st element: " << a[1] << endl;
cout << a[1][0] << " " << a[0][4] << endl;
cout << *(a+4) << endl;
cout << *(*(a+1)+0) << endl;
cout << *(*(a+1)+COLS) << endl;
cout << *(*(a+(1*COLS)+0)) << endl; //why is this no good???
```

works for all rows

rows aren't contiguous anymore
Jagged Arrays

```c
int *array[2];
array[0] = new int[3];
array[1] = new int[2];
```
How does freeing memory work?

```c
int *r[5], **s;
for(int i=0; i < 5; i++)
    r[i]=new int;
for(int i=0; i < 5; i++)
    delete r[i];
for(int i=0; i < 5; i++)
    r[i]=new int[5];
for(int i=0; i < 5; i++)
    delete [] r[i];
s=new int*[5];
for(int i=0; i < 5; i++)
    s[i]=new int[5];
for(int i=0; i < 5; i++)
    delete [] s[i];
delete [] s;
```