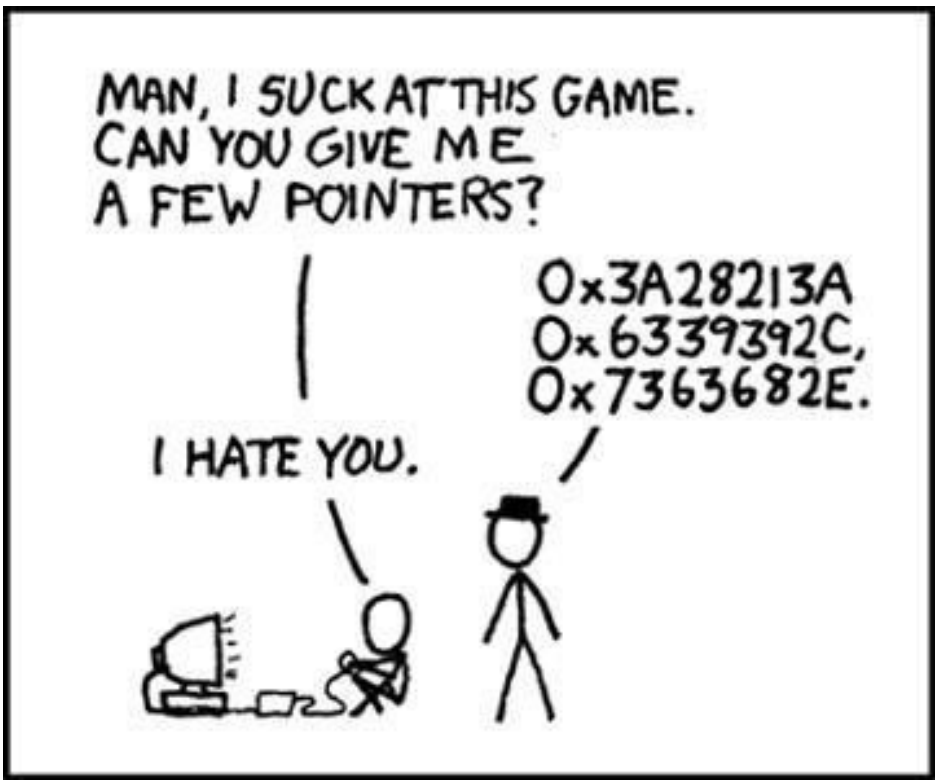
Week 7



Announcements

- Project 5
  - Monday, November 23rd
- Midterm creeping up
  - Tuesday, November 24th

Questions?

- anything?

Pointers

```
/*
Memory:
100 ---
    ..
104 ---
    ..
108 ---
     10 <- p
110 ---
    ..
*/

int n = 10;

// create a pointer to the address of n
int* p = &n;

// de-reference
cout << *p << endl; // prints 10
cout << p << endl;  // prints the memory location (108 in this case)

*p = 15;           // de-reference with *
cout << n << endl; // n has changed to 15

// compare pointers
int m = 30;
int *p2 = &m;

if (*p2 > *p)       // condition is true
  cout << "p2 is greater than p" << endl;

// lets say p2 is at memory location 204
if (p2 > p)         // condition is true
  cout << "p2's memory address is greater than p's" << endl;
```

Iterate Array with a Pointer

```
int a[5] = {50, 40, 30, 20, 10};
a == &a[0] // the name of the array is the same as a pointer to the first element

// change the function prototype to use pointers
void f(int a[], int n);
void f(int* a, int n);

// pointer subtraction
// return -1 if "Waldo" isn't in the array, position otherwise

string names[3] = { "Taasin", "Abby", "Waldo" };
```

```
int wheresWaldo(string s[], int n) {
  for (int i = 0; i < n; i++) {
    if (s[i] == "Waldo")
      return i;
  }
  return -1;
}

/*
Memory:
100 ---
    Taasin <- s
104 ---
    Abby
108 ---
     Waldo <- i
110 ---
    ..
*/

// return a pointer to the position now
string* wheresWaldo(string* s, int n) {

  // for ( string* i = s; i < &s[n]; i++) {     // another valid stop condition
  for ( string* i = s; i < s+n; i++) {
    if (*i == "Waldo")
      return i;
      // i - s is how you get the position back
      // refer to "Memory" above to see why s - i is wrong
  }
  return NULL;
  // return nullptr;
  // return 0; // valid, but don't do this
}

string* r = wheresWaldo(names, 3)
*r // don't de-reference null pointers! program will crash
if (r != nullptr)
  cout << "Waldo is at address: " << r << endl;
```

Structs

```
// want to store a Pokemon trainer
// has fields for main pokemon (a string), number of pokeballs

struct Trainer {
  string mainPokemon;
  int nPokeballs;
};

// create an array, just like normal data types
```

```
Trainer trainers[50];


/*
  Correction: In class I said we should pass the array by reference for efficiency,
              but C++ defaults to passing arrays by reference
  I meant to say that if our struct had a large number of fields and we only looked
  at a few, it would be better to pass the struct by reference.
*/

// write a function to print out a trainers main and number of pokeballs
// const b/c we don't modify
void printInfo(const Trainers list[], int numTrainers) {
for (int i = 0; i < numTrainers; i++) {
    cout << "Trainer " << i+1 << " has main: " << list[i].mainPokemon << " and has "
    << list[i].nPokeballs << " pokeballs" << endl;
  }
}

// use pointers instead
void printInfo(Trainer* list, int numTrainers) {
  for (Trainer* i = list; i < list + numTrainers; i++) {

    // two ways to access the fields of a pointer to a struct
    // don't forget that . has higher precedence than *
    cout << "Trainer " << (i-list) + 1 << " has main: " << (*i).mainPokemon << " and has "
    << (*i).nPokeballs << " pokeballs" << endl;

    // -> arrow operator
    cout << "Trainer " << (i-list) + 1 << " has main: " << i->mainPokemon << " and has "
    << i->nPokeballs << " pokeballs" << endl;
  }
}


// function: input a trainer, subtract a pokeball
// void minusPokeball(Trainer &t) {
void minusPokeball(Trainer* t) {
  // Trainer.nPokeballs  // common mistake! you need to access an instance
                         // of trainer, not the type
                         // this line is similar to saying string.length

  t->nPokeballs -= 1;
}

void minusPokeball(Trainer t[], int pos) {
  // Trainer.nPokeballs = Trainer.nPokeballs - 1;

  t[pos].nPokeballs -= 1;
}


/*
Document your code! Code from here on out will require design choices from
```

```
you, and to reduce headaches later, write some good comments
*/
```