Data Structures

- Value, Reference
  + Pointer variables

Announcements

- HW1: Some people only turned Q1.
  Possibly forgot to add other files?
  - bankson
  - jsrodriguez
  - mohammadhadi

  Come see me!

  Also: colemanct, Q1 issue
  (also come see me

- Missing: baberd
  cherupallas
  danganamme
  dmhicks
  elkowtm
  kahlerap
  suljicv

Look for file in your repo early next week.

Also, note: Must compile!

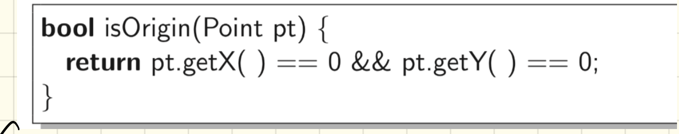
Cont:

- New office hours:
  Fri 1-2pm
- HW: due next Thursday
  [via git
  use comments
- Lab: due today
  (weekly from here on out)

More on variables

In Python, variables were just identifiers for some underlying object.

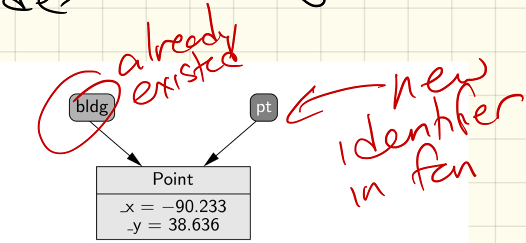
This had implications when passing variables to functions:



```
bool isOrigin(Point pt) {
  return pt.getX( ) == 0 && pt.getY( ) == 0;
}
```

↳ So if you do:

```
if (isOrigin(bldg))
  <code>
```



already existed

new identifier in fcn

Figure 14: An example of parameter passing in Python.

in lists – meant had shallow copies

C++: Much more versatile:

3 parameter types

1. Value
2. Reference
3. Pointer

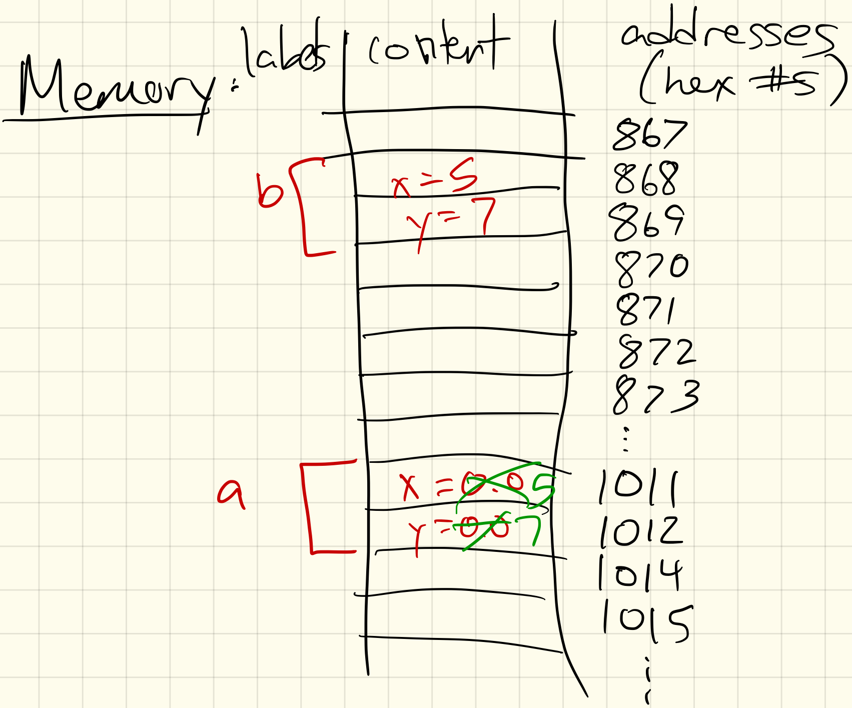
So far, you've been using value - easiest.

Reference + Pointer require looking at memory more carefully...

① Value Variables

When a variable is created, a precise amount of memory is allocated:

```
Point a;
Point b(5,7);
```



Now:

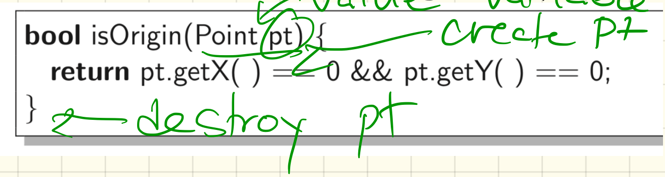
```
a=b;
```

What happens?

Functions + passing by value:

value variable

create pt



```
bool isOrigin(Point pt){
  return pt.getX( ) == 0 && pt.getY( ) == 0;
}
```

destroy pt

When someone calls

main {

isOrigin(mypoint);

}

The (local) variable pt is created as a new, separate variable

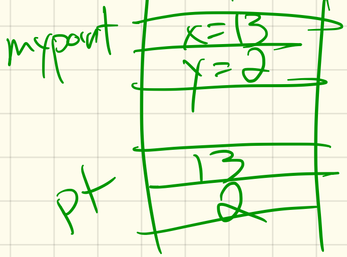
Essentially, compiler inserts

Point pt(mypoint);

as first line of the function.

So - what if we change pt?

mypoint stays the same



2 Reference variables

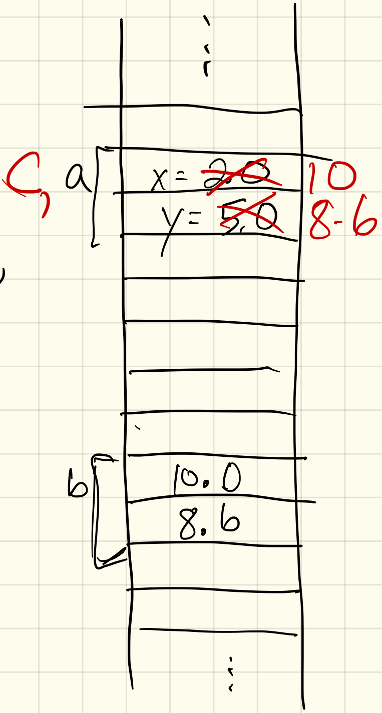
Syntax:

int& x(y);

Point& c(a);

What it does:

- c is created as an alias for a
- Similar to Python, but c is identical to a

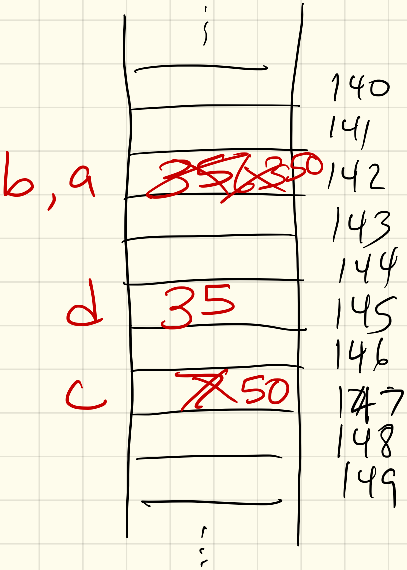


Ex: c = b;

Longer example

```
int a;
 a = 35;
  int & b(a);
  int c(7);
 int d(a);
   b = 63;
    a = 50;
    c = b;
```

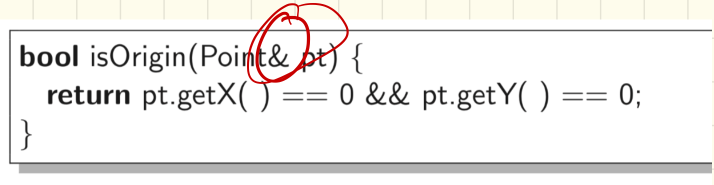
Value



Functions: pass by reference

Generally, you'll never see reference variables used directly in main or in code.

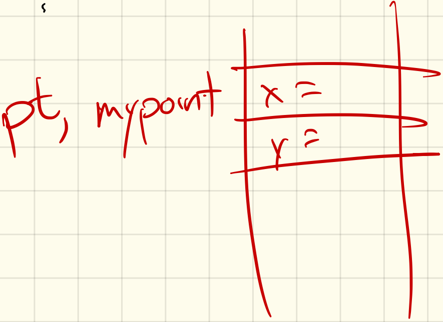
Primary purpose: function calls



```
bool isOrigin(Point& pt) {
  return pt.getX( ) == 0 && pt.getY( ) == 0;
}
```

Then, in main:

```
if (isOrigin(mypoint)){
    //code
     ⋮
}
```



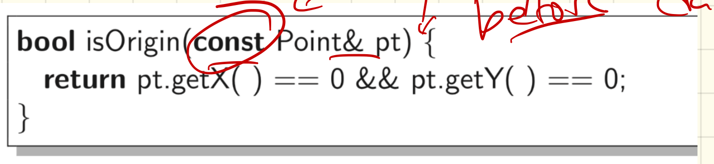
Why pass by reference?

3 main reasons:

- Space: making 2 copies of a huge list is often bad
- Time: must spend the time to copy
- Persistence: this lets changes stick around

If you want speed + space,
but don't want the function
to change the variable:

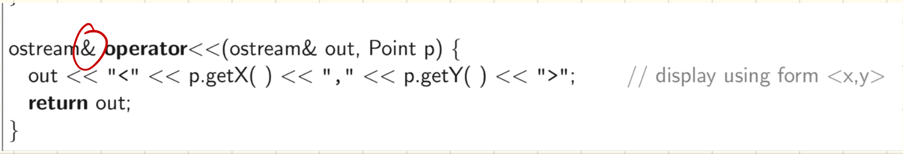
input parameter list,
before data type



```
bool isOrigin(const Point& pt) {
  return pt.getX( ) == 0 && pt.getY( ) == 0;
}
```

Compiler will enforce that
pt will have no changes.

Actually, recall:



```
ostream& operator<<(ostream& out, Point p) {
  out << "<" << p.getX( ) << "," << p.getY( ) << ">";        // display using form <x,y>
  return out;
}
```

③ Pointer variables

Syntax: `int * d;`

d is then a variable which stores a memory address.

Ex:

```
int b(8);
int *d;

d = &b;
(*d) = 5;
```

returns address b is at

follow d's pointer & change it

	⋮	⋮
		273
d	277	274
		275
		276
b	~~8~~ 5	277
		278
		279
		280
		281
	⋮	⋮

But: d is not an int.

d = b; ← ERROR

Pointers: getting to the data

- Called dereferencing.

Ex:

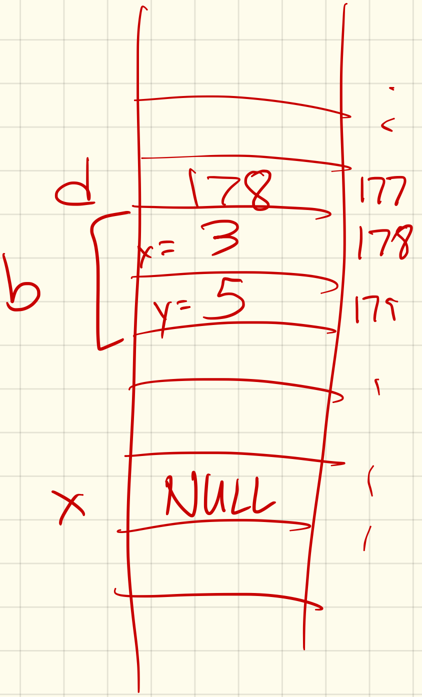
```
Point *d;
Point b(3,5);
d=&b;
Point * x;
```

Then 2 options:

```
(*d).getX();
```

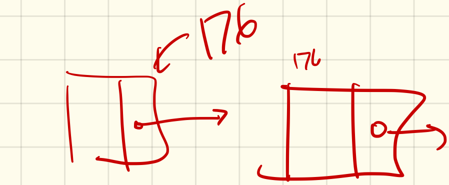
or

```
d->getX();
```



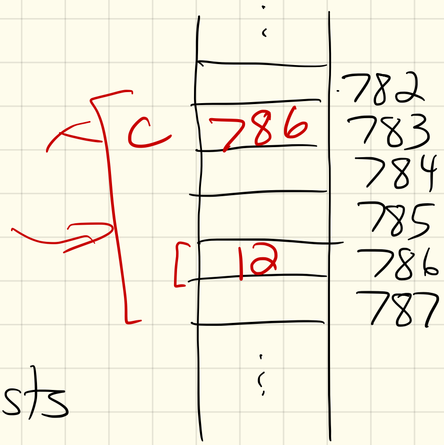
->

NULL == 0



The new command (in C, malloc)

```
int *c;
c= new int(12);
```



Why: The data persists even after the pointer is gone!

Main use:

in classes

(more in a slide or two)

Passing pointers

Can be useful, since allows NULL option.

Ex:
```
bool isOrigin(Point *pt = NULL){
        return pt→getX()==0 &&
                pt→getY()==0;
}
```

Similar to pass by reference, but can also pass a NULL this way.

Pointers in a class

Pointers are especially useful in classes.

Often, we don't know the details of private variables at time of object creation.

Example: using an array

At time of declaration, need:

- type
- var name
- size

An example: A simple vector class

vector in $\mathbb{R}^2$: <2,5>

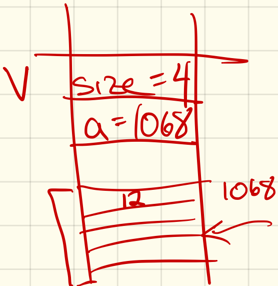
vector in $\mathbb{R}^4$: <0,1,0,5>

So size is not fixed!

How to make a class?

```
class MyFloatVec {
 private:
    int size;
     float * a;  //pointer to an array

 public:
     MyFloatVec (int s=10) {
         size = s;
         a = new float[size];
         a[0]=12;
     }
in main
 MyFloatVec v(4);
```



Accessing an array:

Pointers to arrays are special

↳ any array in fact <u>is</u> just a pointer to the 1st spot in the array

(no * or → needed)

<u>Ex</u>: Write a function to allow [ ] notation, so x[i] gives ith element in the vector:

```
pubic:
    //constructor
      ⋮

   float& operator[] (int i){
        if (i<size)
            return a[i];
        else
            <error>
   }
```

Another: Write a function
to scale vector by scalar:

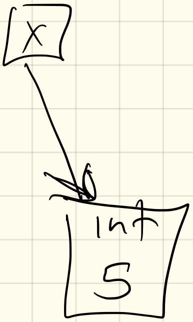
```
void scale(float value) {


}
```

Garbage Collection:

In python, data that is longer in use are automatically destroyed.

Ex:



```
x=5

x=10
```

Pros:

Cons:

C++:

- Value & reference variables are destroyed at the end of their scope

Standard variables are just a label attached to data

↳ data is deallocated, so those spaces are now free again.

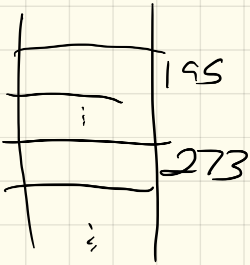
Problem: Pointers

The pointer is destroyed

↳ not underlying data

```
int main() {
   int * x = new int(5);


}
```



Rule:

Using .h files

In C++, .h files let you separate out a class or class declaration.

Formally, these header files are used to declare the interface of a class.

Ex:

- Seperate out Point.h
- Then have Point.cpp to fill in longer functions
- Finally, have a testing program (which includes Point.h & has the main)