CSCI 2100

C++:
Value, Reference
& Pointer Variables

Recap

- Demo of compiling & using hopper
- HW2 - due Friday
  - #1 due on paper by start of class
  - #2 on ZyLabs by midnight

Last time:

covered classes

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>

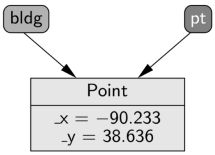


Figure 14: An example of parameter passing in Python.

Shallow: changing pt in function also changed value outside

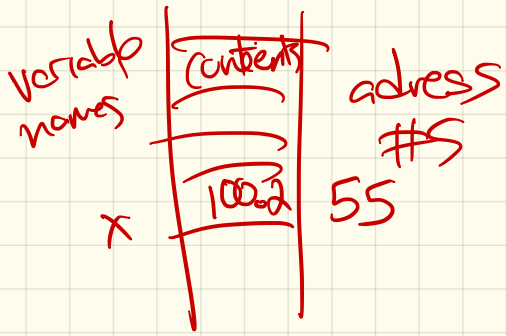
C++: Much more versatile.

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...

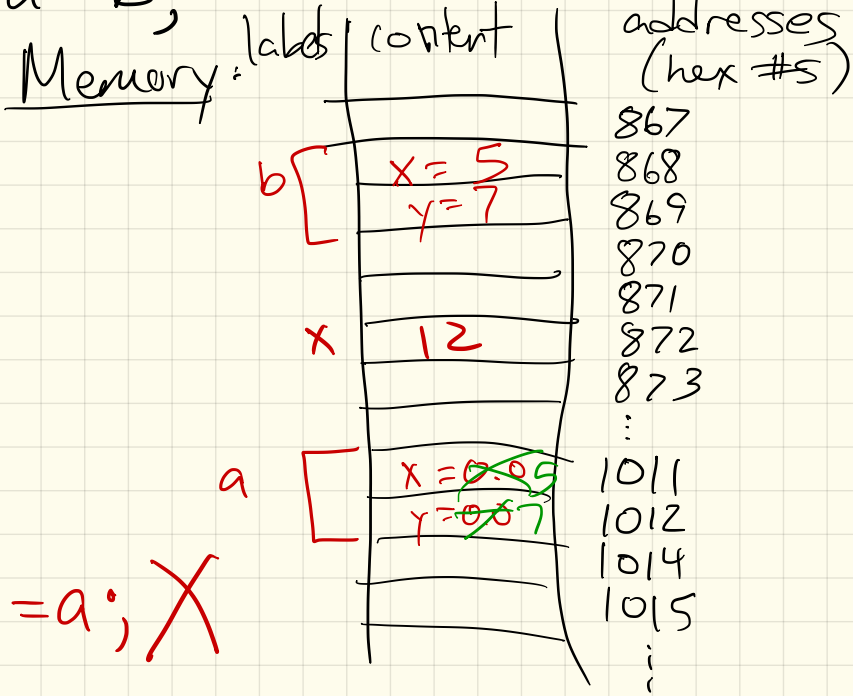


(1) Value Variables

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

```
int x = 12;
Point a;
Point b(5,7);
a = b;
```

Memory:



x = a; X

Functions & passing by value:

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

When someone calls

main {
  isOrigin(mypoint);
}

pt = 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?

No change outside of fcn

↳ "deep copy"

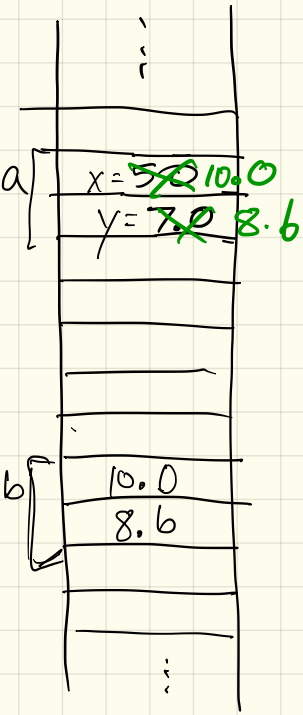
2) Reference variables

Syntax:

Point& c(a);

What it does:

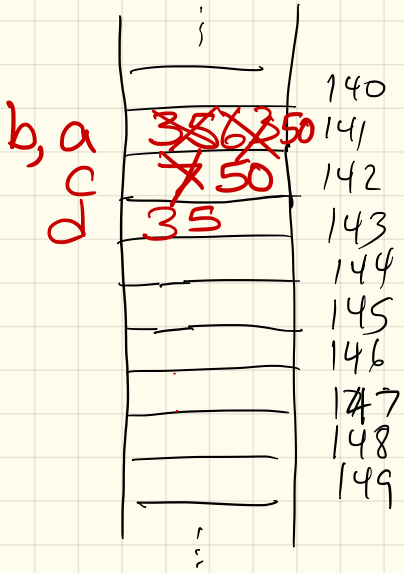
- c is created as an alias for a
- Similar to Python, but c is identical to a



Ex: c = b;

Longer example

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



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:

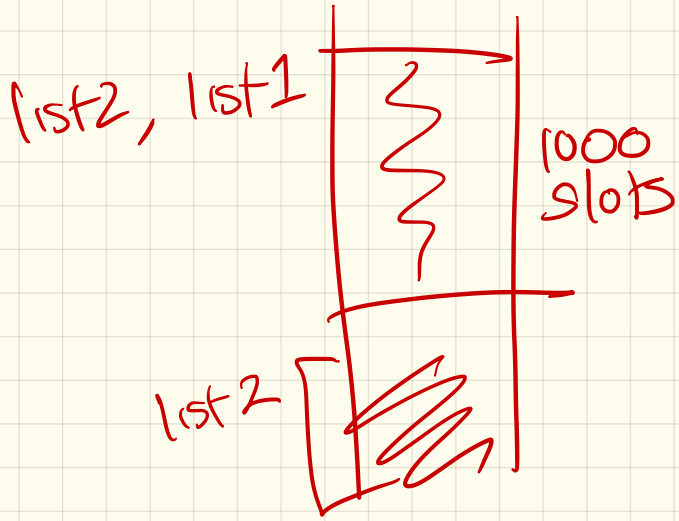
```
if (isOrigin(mypoint)){
    //code
}
```

If fun changes the variable, it lasts outside (in main)

Why pass by reference?

3 main reasons:

- Space: copying entire data structure is space prohibitive
- Time: Value variable copies the data.
  Huge list → large time cost.
- Persistence:
  If change in fcn should change data outside



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

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

??

pt, a

Compiles 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;
}
```

cout << pt1 << endl << pt2 << endl;

a = (b = (c = d));

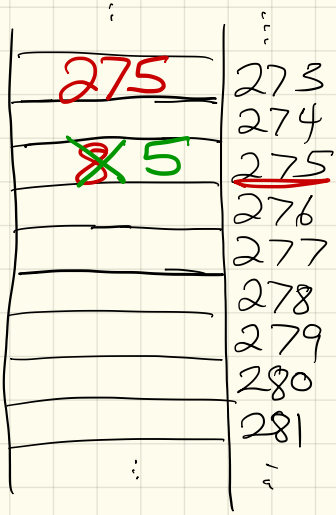
(3) Pointer variables (Ref-8)

Syntax: `int *d;`

d is then a variable which stores a memory address.

Ex:
```
int b(8);
int *d;
d = &b;
(*d) = 5;
```
mem address

d = 279; ✗ (int - NO)

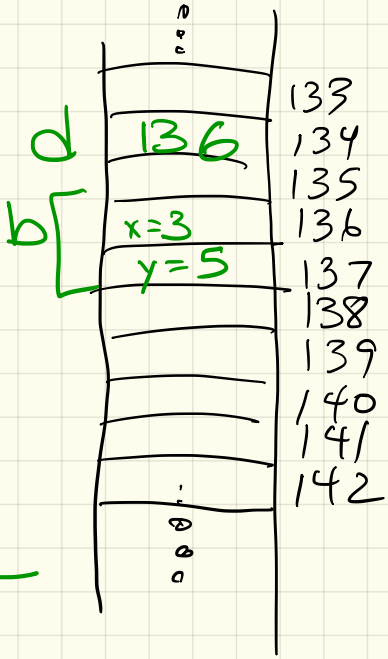


But: d is not an int.

d = b; ~~4~~ → ERROR

Pointers: getting to the data
- Called dereferencing.

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



Then 2 options:

```
(*d).getX();
```
or
```
d->getX();
```
(same)

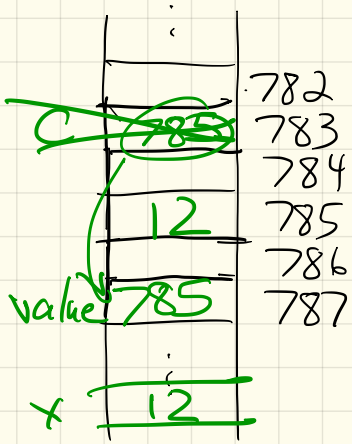
head->next->next

~~*(*head).next).next~~

The new command

in some function; myfunc

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



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

in funct 2:

```
int x = 12;
return x;
```

x → 12

Main use:

in main:

```
int* value = myfun();
```

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:

But- what if size might change, or is unknown?

An example: A simple vector class

vector in $\mathbb{R}^2$: $\langle 2, 5 \rangle$

vector in $\mathbb{R}^4$: $\langle 0, 1, 0, 5 \rangle$

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];
    }
```

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:

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.

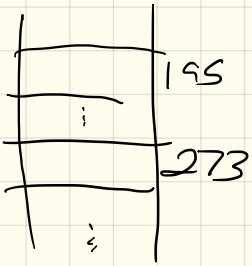
Problem: Pointers

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:

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