CS2100 - Variable Types

Note Title 9/5/2013

Announcements

- Math/CS club meets at 4 in Ritter lobby

Last time

- Scope

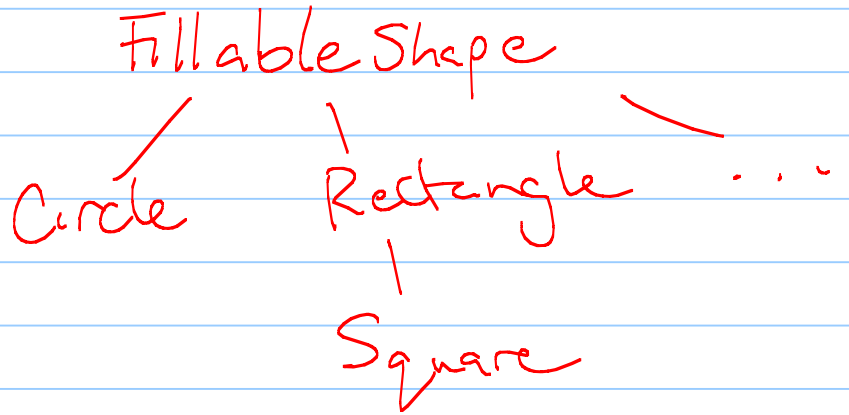
- Classes

Inheritance

What is inheritance?

- Creating a "child" object class that steals data/methods from parent class

Ex:



Example: Square class

child — parent

```
class Square : public Rectangle {
public:
  Square(double size=10, Point center=Point( )) :
    Rectangle(size, size, center)        // parent constructor
  { }

  void setHeight(double h) { setSize(h); }
  void setWidth(double w) { setSize(w); }

  void setSize(double size) {
    Rectangle::setWidth(size);       // make sure to invoke PARENT version
    Rectangle::setHeight(size);      // make sure to invoke PARENT version
  }

  double getSize( ) const { return getWidth( ); }
};  // end of Square
```

constructor fcn for square

call parent constructor

override Rectangle's functions by same name

call parent function

Other issues

A new type of data. So far, have seen: public and private.

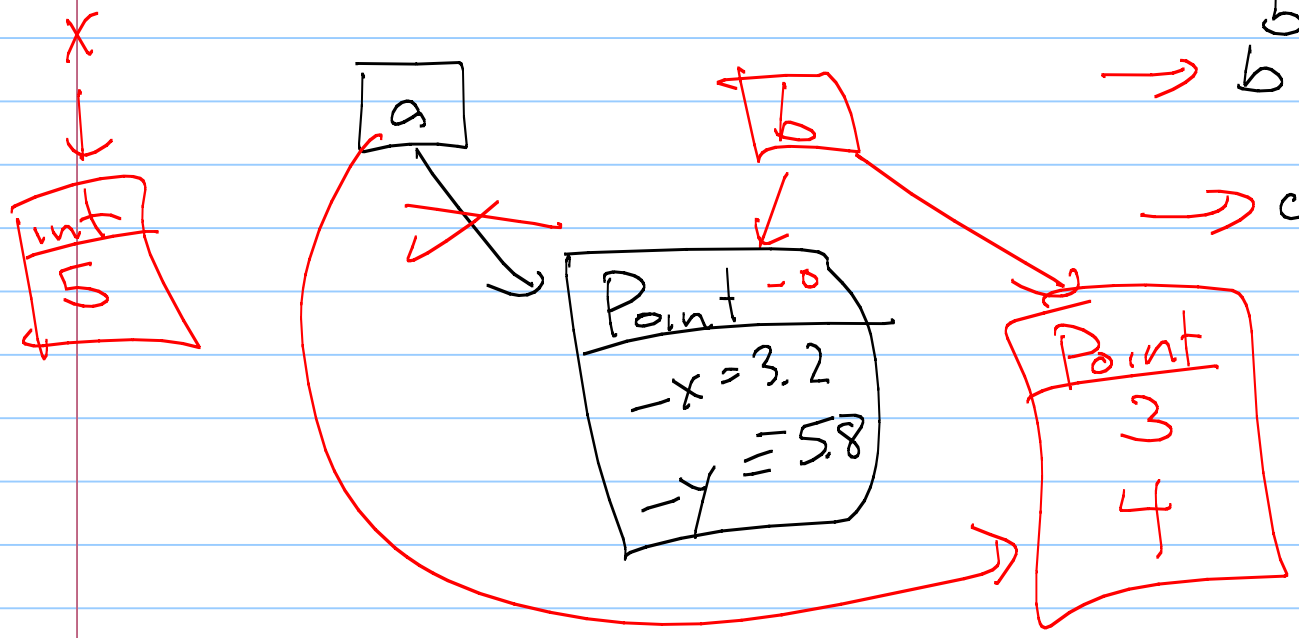
What about data that main can't have, but child classes should?

```
class Whatever {
    protected:
        helperFcn()    // children can use this
```

~~Objects~~ Variables

In Python, variables were pointers to actual data.

```
x = 5;
b = a;
b = Point(3, 4);
a = b;
```



C++ : More versatile

C++ allows for 3 different types of variables.

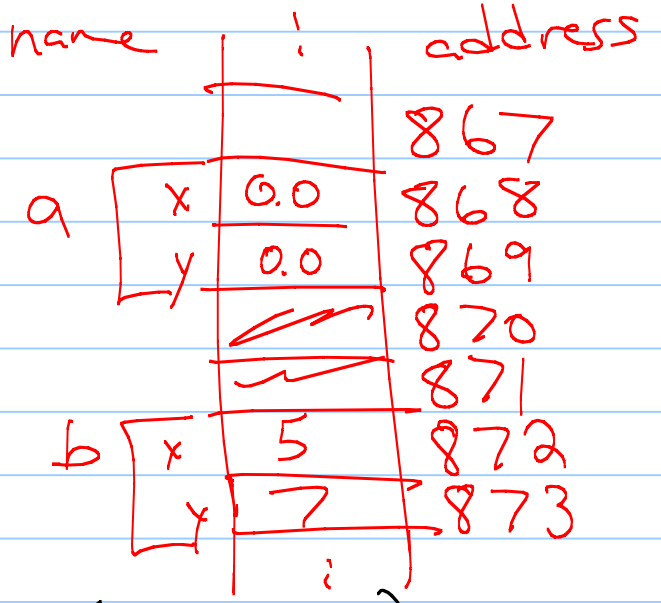
1. Value
2. Reference
3. Pointer

① Value Variables

When a variable is created a precise amount of memory is set aside.

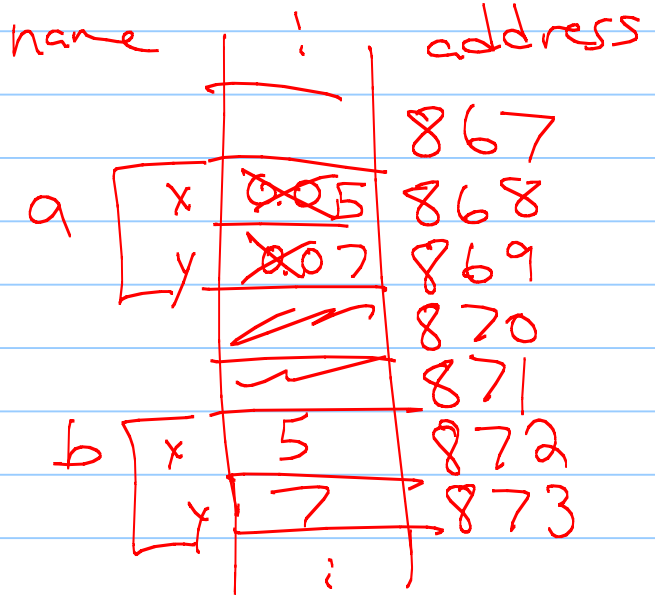
→ Point a;

Point b(5,7);



More efficient (for both speed & space).

Now set a = b :



They stay separate!

By default — deep copies.

Functions: passing by value

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

When someone calls isOrigin(myPoint), the value of pt is initialized as a new, separate variable.

Essentially, the line:

Point pt(myPoint);

is run at the beginning of the function!

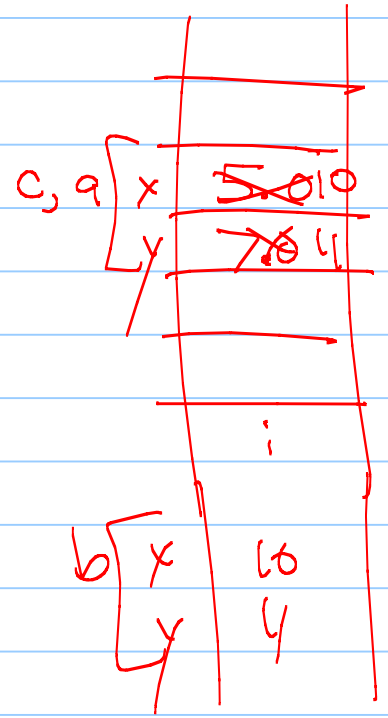
So do changes to the point last?

No - makes deep copy.

② Reference Variables

Syntax: `Point & c(a);`

- c is created as an alias for a
- More like Python, but c is always the same as a.

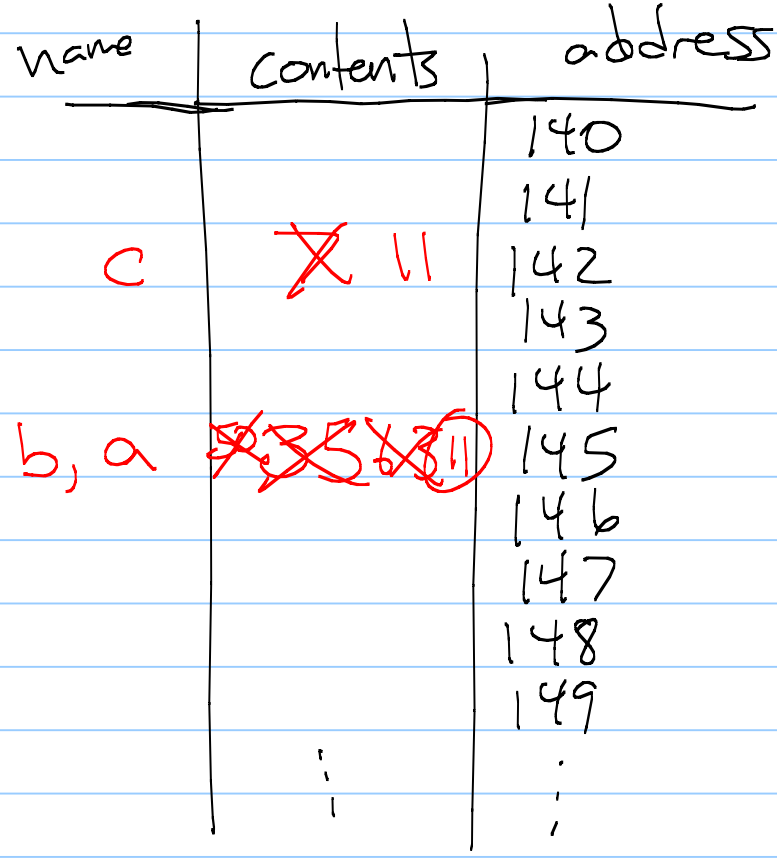


Ex: `C = b;`

Will not make c point to b, but will actually change value of a.

Ex:

```
int a;
a = 35;
int & b(a);
int c(7);
b = 63;
c = 11;
a = 50;
b = c;
```



Passing by reference

Reference variables aren't generally use in main.

Instead, primary purpose is in functions:

Ex:

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

pt, mypt → x, y

in main:

```
if (isOrigin(mypt))
```

Why pass by reference?

3 main reasons

- changes persist out side of fcn
- saves time — no need to make a copy of (potentially) a huge input!
- saves space!

If we want the speed of passing by reference, but we don't want changes to variable, use const:

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

Compiler will enforce that pt isn't changed inside the function.

Strongly - don't call anything that even could change the value

Recall: Point output

```
ostream& operator<<(ostream& out, Point p) {
  out << "<" << p.getX( ) << "," << p.getY( ) << ">";
  return out;
}
```

Here, & is required since streams cannot be copied.

Note: don't use const. Why?

(3) Pointer variables

Syntax: `int * d;`

d is created as a variable that stores a memory address.

Ex:

✓ `int b(8);`

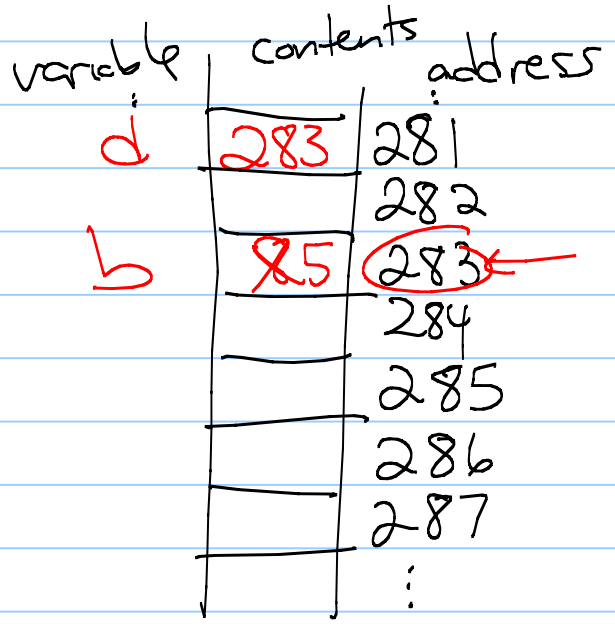
✓ `int* d;`

✓ `d = &b;` ← give me mem. address of variable

`(*d) = 5;`

But d is not an int.

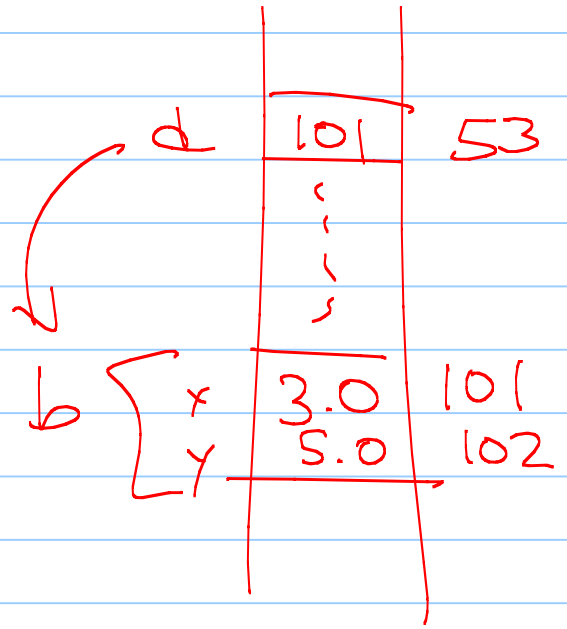
Can't write `d = b;`!



Pointers: getting to the data

Called dereferencing.

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



2 options:

```
(*d).getX();
```
use for int

or

```
d->getY();
```
> snazzier if using object

The new command

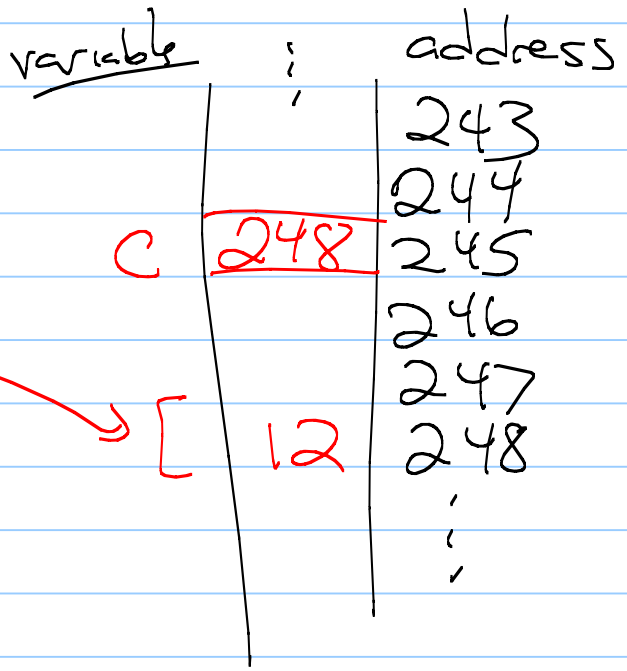
```
{
  int* c;
  c = new int(12);
}
```

248

// c is destroyed — but 12 is not

Main use: The data persists even after the pointer is gone!

So can create or modify inside multiple functions.



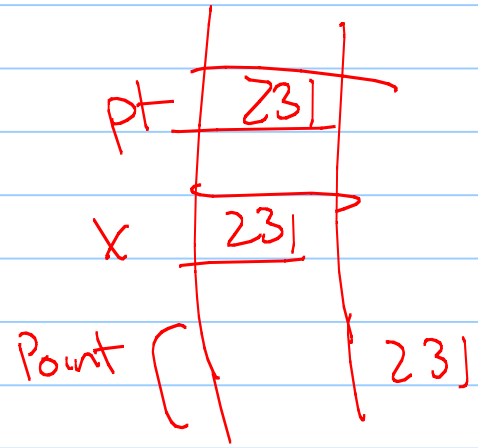
Passing pointers

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

=NULL

Similar to passing by reference, but allows passing a NULL pointer also.

isOrigin(x)



Pointers in a class

Pointers are especially useful in classes.

Often, we don't know all the details of private variables to put in the private declaration.

Example: arrays!

What do we need when creating an array?

type & size

↑ pts

Example class: vector of floats

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

A vector in $\mathbb{R}^4$: $\langle 0.0, 0, 1 \rangle$

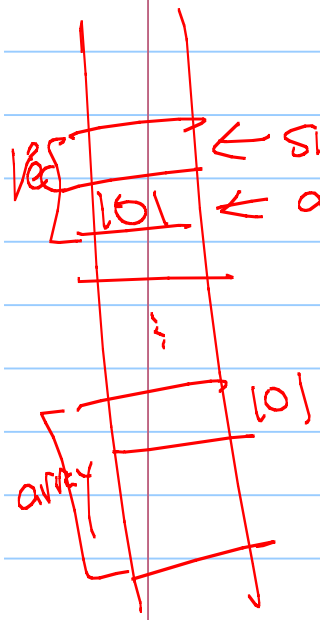
Dynamic size!

So how to make a class?

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

```
class MyFloatVec {
  private:
    int size;      // dim of space
    float* a;      // ptr to array holding #s

  public:
    MyFloatVec( int s=10 ) : size(s) {
      a = new float[size];
    }
    :
```



Accessing the array:

With an array, can just pretend the variable isn't a pointer.
(so no * or →)

Ex: override the [] so that x[i] will give back the i^{th} element in the vector:

(x: vector)

```
float operator[](int index) {
    return a[index];
}
```

Function to scale by int (in class):