CS180 - Variable Types in C++

Note Title 9/12/2011

Announcements

- HW due tomorrow by midnight
- Lab 2 by Sunday
  (don't forget to submit even if not perfect!)
- HW2 will be up today
  due next Friday (?)
- Need to reschedule next Thursday's office hours - at 9-10 am.

Last time:

Classes

```
class Point {
private:
  double _x;                                // explicit declaration of data members
  double _y;

public:
  //constructor
  Point( ) : _x(0), _y(0) { }               // constructor

  double getX( ) const {                    // accessor
    return _x;
  }

  void setX(double val) {                   // mutator
    _x = val;
  }

  double getY( ) const {                    // accessor
    return _y;
  }

  void setY(double val) {                   // mutator
    _y = val;
  }
};
```

C++ : More versatile

C++ allows for 3 different types of variables.

✓ ① Value

② Reference

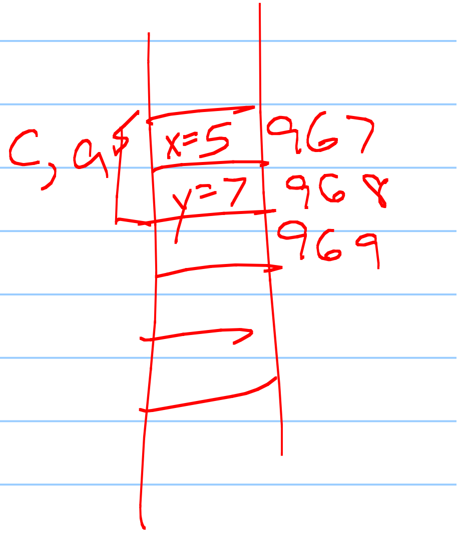
③ Pointer

② Reference Variables

int & x(y);

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;
```

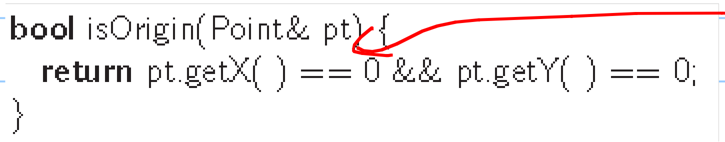
name	contents	address
		140
b, a	11 ~~50~~ ~~35~~ ~~63~~	141
		142
		143
		144
		145
c	~~7~~ 11	146
		147
		148
		149
	⋮	⋮

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;
}
```

Point & pt (input var)

Why pass by reference?

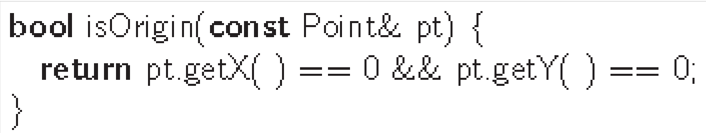
3 ~~2~~ main reasons

① saves space } ☆ ☺

② saves time

③ allows changes to persist

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

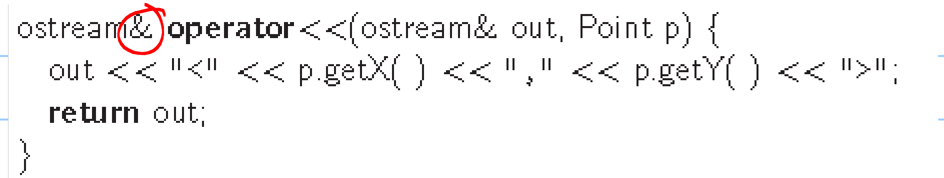


```
bool isOrigin(const Point& pt) {
   return pt.getX( ) == 0 && pt.getY( ) == 0;
}
```

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

Recall: Point output

<2, 3>



```
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?

Goal is to change output stream!

③ Pointer variables

Syntax: int * d;

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

Ex:

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

d = &b;
~~d = &x;~~
```

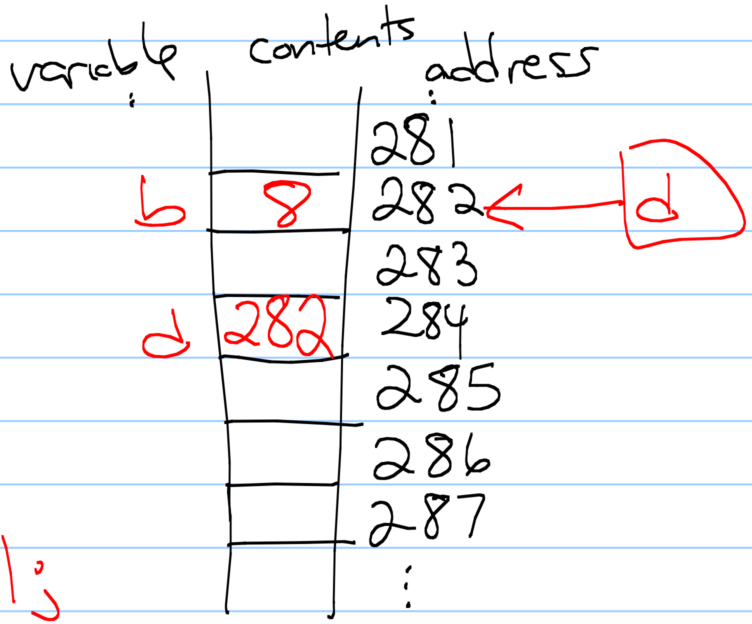
memory address of b

error

But d is not an int.

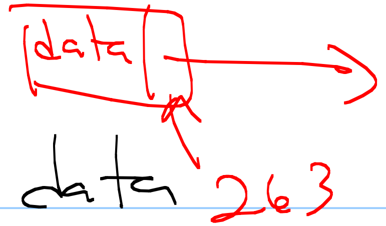
Can't write d=b!

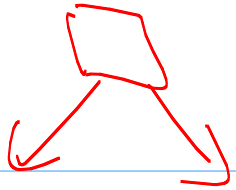
```
cout << *d << endl;
```



variable	contents	address
⋮		⋮
		281
b	8	282
		283
d	282	284
		285
		286
		287
		⋮

Pointers: getting to the data





Called dereferencing.

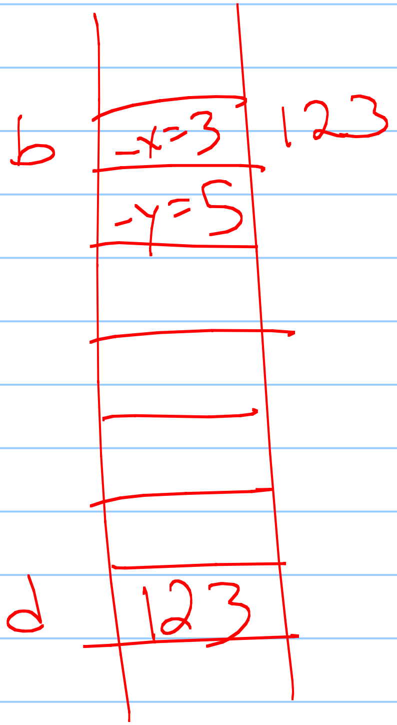
Ex:
```
Point * d;
Point b(3,5);
d = &b;
```

2 options:

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

or

```
d -> getY();
```



The new command

```
helper() {
int* c;
c = new int(12);
int * d=c;
} //c is destroyed
```

allocates a block of memory

variable		address
	⋮	
		243
c	247	244
		245
		246
	12	247
		248
d	247	
		⋮

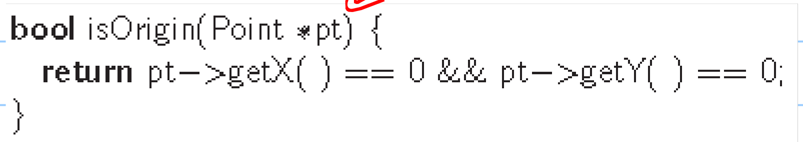
Main use: the data persists even after the pointer is gone!

So can create or modify inside multiple functions.

```
int ** x; //pointer to a pointer
```

Passing pointers

Pointer * pt == NULL



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

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

NULL = 0

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?

Example class: vector of floats

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

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

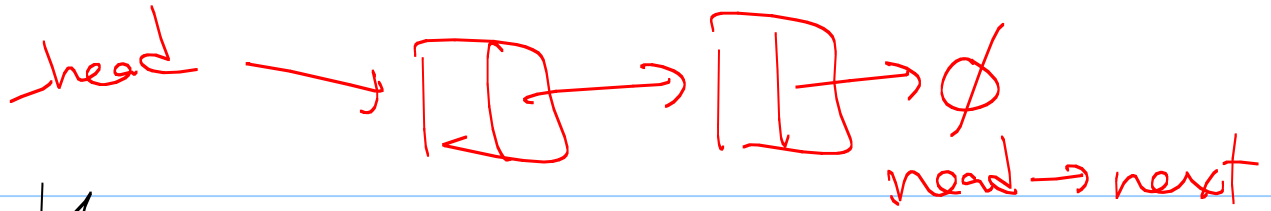
Dynamic size!

So how to make a class?

```
private:
        int _size;
        float* _V;
```

```
class    My~~Int~~FloatVec   {
   private:
      int  _size;   // size of this array
      float *_V;  // pointer to my array

   public:
      My~~Int~~FloatVec ( int  s=10) :_size(s) {
          _V = new float[_size];
   };
```



Accessing the array:

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

inside constructor to 0-out ~~the~~ vector:

```
for (int i=0; i<_size; i++)
    _V[i]=0;
```

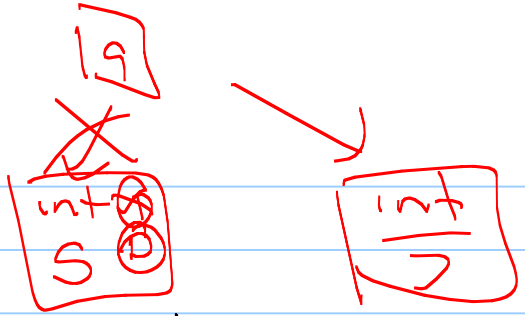
Function to scale by int (in class):

```
void operator*(int x) {
        for (int i=0; i< _size; i++)
            _V[i]*= x;
}
```

i = i+1

_V[i] = _V[i]*x

Garbage Collection



In Python, variables that are no longer in use are automatically destroyed.

Pros: easy!

Cons: Slow

C++

In C++, things are sometimes handled for you.

Basically, any standard variable is automatically destroyed at the end of its scope.

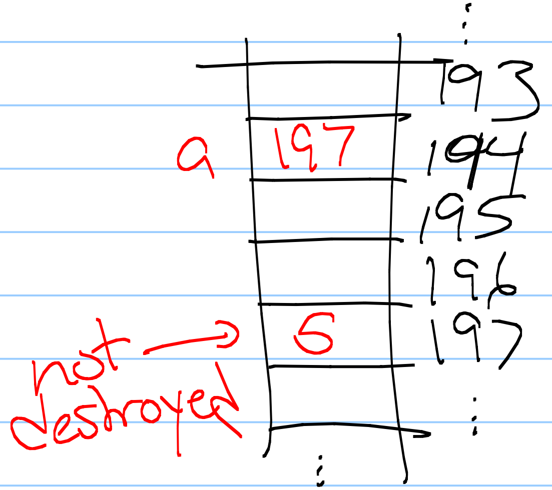
This holds for any type of variable!

Problem: Pointers

While the pointer variable is deleted, the spot you created with a "new" is not.

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

    delete a;
} // a is destroyed
```



Rule: If you have a new, must have a delete!