CS 180 – C++: Variable Types

Note Title 8/30/2010

Announcements

- Turn in HW now
- Program 1 posted
  - checkpoint on Tuesday
- Lab is Friday, not Thursday

Types of Variable

1. Value
2. Reference
3. Pointer

② Reference Variables

Syntax:

```
Point& c(a);  // reference variable
```

- c is created as an alias for a
- More like Python model, but can't be changed later

Ex: c=b;
Will not rebind c to point to b, but will change the value of c (and a).

Passing by reference:

Reference variables aren't usually needed in main program.

Instead, they are primarily used for passing to functions.

Ex:

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

Advantage? - changes persist ←

★ - better speed, less space

If we want the speed of passing by reference but don't want our object mutated, use const.

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

Compiler will ensure that pt isn't modified.

Speeding up the Point class:

original: `double distance(Point other) const {`

faster: `double distance(const Point& other) const {`

Another:

```
Point operator+(const Point& other) const {
  return Point(_x + other._x, _y + other._y);
}
```

Note: Return type is still value. Why?

Point created inside the function is destroyed at end of function.

Recall: Point output

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

Here, & is required because streams cannot be copied.

Note that we don't use const, since we are changing the stream by adding data.

(cout << ___) << ___ << endl;

(3) Pointer variables

Syntax : `Point *d;  // d is a pointer variable`

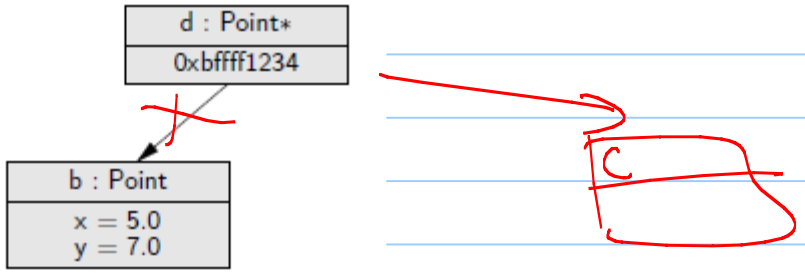
d is created as a variable that stores a memory address.

So:

```
Point *d;     // d is a pointer variable

d = &b;
```

d = &c;



But d is not a Point! can't say d = b

can't say d.getX();

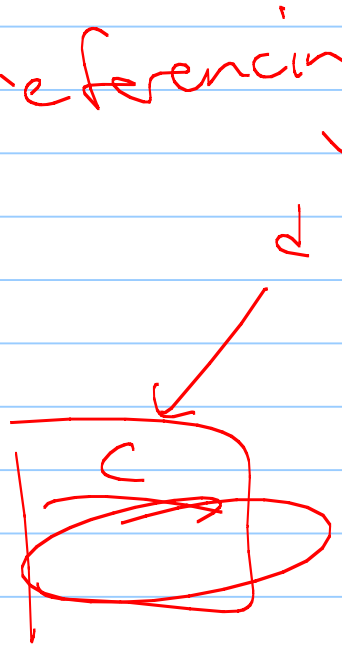
Using pointer variables

2 options:

dereferencing

```
(*d).getY();
```

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

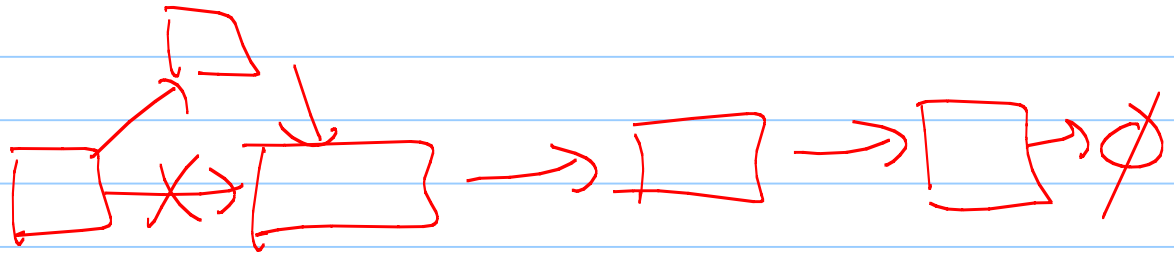


Passing by Pointer

Point *pt = NULL

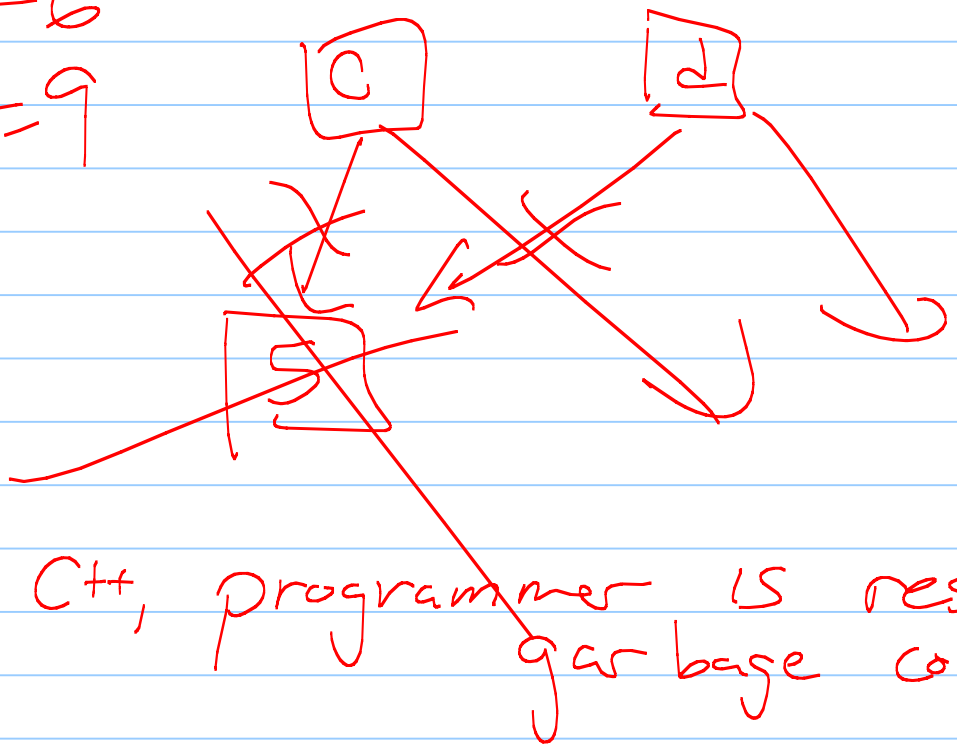
```
bool isOrigin(Point *pt) {
  return pt->getX( ) == 0 && pt->getY( ) == 0;
}
```

This is similar to passing by reference, but allows you to also pass a null pointer.



(Python)

```
c = 5
d = c
c = 6
d = 9
```



In C++, programmer is responsible for garbage collection.

More on Classes:

Destructors:

```
~Point() {
delete _x;
delete _y;
}
```

If your class opens files or allocates memory, then can't just use delete.

Must create a destructor.

~ClassName() - no arguements, no return type

~Point()

(garbage collection - automatic in Python, not in C++)

example with CreditCard

Copy Constructors:

Previously:

```
Point a;
Point b(a);
```

Consider the following:

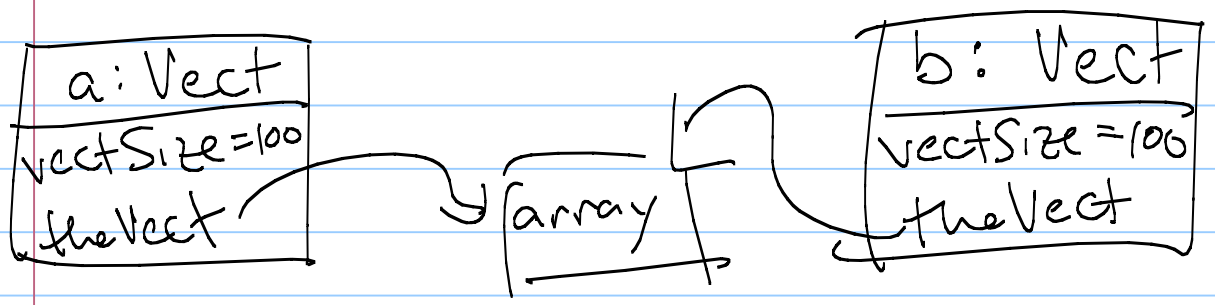
```
Vect a(100);
Vect b(a);
```

What does this do?

Copies each element of a to b.

int → vectSize = a.vectSize;

array → theVect = a.theVect;



Shallow copy

To fix, write our own copy constructor:

```
Vect b(a);
```

```
// copy constructor
Vect(const Vect &a) {
    vectSize = a.vectSize; // copy size
    for (int i=0; i<vectSize; i++) {
        theVect[i] = a.theVect[i];
    }
}
```

pointer to an array

```
theVect = new int[vectSize];
```

new is used for arrays

Another problem:

```
Vect a(100);
Vect c;
c=a;
```

What does this do? shallow copy
by default, copies each parameter:

```
c.vectSize = a.vectSize;
c.theVect = a.theVect;
```

Write operator = to make deep copy of data.

Enum: user defined types

```
enum Color {RED, BLUE, GREEN};
```

RED ↑ 0, BLUE ↑ 1, GREEN ↑ 2

```
Color sky = BLUE;
Color grass = GREEN;
```

Convention: write in all capital letters