

CS180 - Classes in C++

Note Title

9/8/2011

Announcements

- HW1 due Sat.
- Look for HW2 on website soon.
- Lab tomorrow (posted on lab page)
 - don't forget to email prelab
before class!!

A note on variable scopes :

how long it exists

```
int main () {  
    int a;  
    if (a > 0) {  
        int b = 12; } // b is destroyed  
    else {  
        int b = 16; } // b is destroyed  
    cout << "a is " << a << endl;  
    cout << "b is " << b << endl; } ← Compiler error  
} // a is destroyed
```

for loops :

```
for (int i=0; i< val; i++) {
```

} // i is destroyed

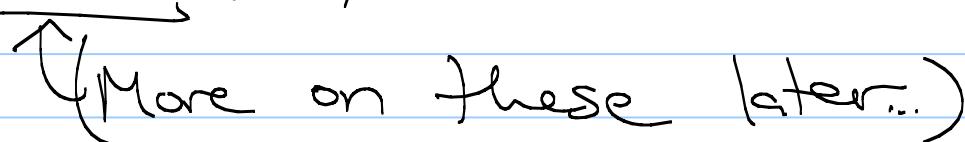
cout << i ← error

Arrays as inputs to functions

Example: Write a function to specify if sum of values in an array is even.

```
bool evenSum(int arr[], int n) {  
    int sum = 0;  
    for (int i=0; i<n; i++)  
        sum = sum + arr[i];  
    return ((sum % 2) == 0);  
}
```

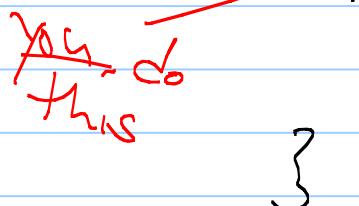
Note:

- `int *x[]` actually makes a (the array)
a pointer!

(More on these later..)

Doesn't copy whole array but can
pretend that it does - just use
it like an array.

To call:

```
int main() {  
    // create & put values in myArray  
    if (evenSum(myArray, length))  
        cout << "The sum is even" << endl;  
}
```



Classes

What is a class?

data field: a collection of data

store arbitrary collections

along with ^{of data} allowed operations

Ex: records for people

Creating an instance of a class

Example:

string s;

string greeting("Hello");

calls Constructor
for class

input to initialize

Never:

string s();

Why? declares a function named s
with no inputs which does
nothing

Never: string("Hello") greeting;

Why? compiler hates it

Example :

Constructor

```
class Point {  
private: name of class accessible only inside class  
    double _x;  
    double _y;  
public: declaration list // explicit declaration of data members  
    Point( ) : _x(0), _y(0) { int x; } // constructor  
    double getX( ) const { // accessor  
        return _x;  
    } no semicolon  
    void setX(double val) { // mutator  
        _x = val;  
    } no self._x  
    double getY( ) const { // accessor  
        return _y;  
    }  
    void setY(double val) { // mutator  
        _y = val;  
    }  
};
```

Classes:

① Data - public or private - is explicitly declared, not just used in constructor.

This is done inside the class, but
not inside a function.

Why?

Scope would only be that function.

② Constructor Function

- name: same as class

- no return type (only one!)

- can initialize variables via a list

Point() : x(0), -y(0) { } Point() {
 x=0;
 -y=0;}

Point(double initialX=0.0, double initialY=0.0) : x(initialX), -y(initialY) { }

Other differences

③ No self! Can just use -x or -y + it immediately scopes to the class attributes.

(There is a "this", but its usage is a bit more complex.)

④ Access control - public versus private.

Point mypoint;

mypoint.-x = 2; ↳ compiler error

⑤ Accessor versus mutator

```
double getX( ) const { return _x; }  
void setX(double val) { _x = val; }
```

can enforce this in C++.

Robust point class : add functionality

In main:

double dist =
pt1.distance(pt2);

Point pt3 = pt1 + pt2; }
or
pt3 = pt1.operator+(pt2);

pt1 * pt2; }
pt1 * 3.0 }
}

```
double distance(Point other) const {
    double dx = _x - other._x;
    double dy = _y - other._y;
    return sqrt(dx * dx + dy * dy); // sqrt imported from cmath library
}

void normalize() {
    double mag = distance( Point() ); // measure distance to the origin
    if (mag > 0)
        scale(1/mag);
}

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

Point operator*(double factor) const {
    return Point(_x * factor, _y * factor);
}

double operator*(Point other) const {
    return _x * other._x + _y * other._y;
} // end of Point class (semicolon is required)
```

Important things

- 1) $-x + \text{other_} -x$ ← allowed only inside
the class
- 2) using operator +
- 3) two versions of * ← can't use
since return types were different
(Instance)

Additional functions

(Not in the class)
3. End of Point)

```
// Free-standing operator definitions, outside the formal Point class definition
Point operator*(double factor, Point p) {
    return p * factor; // invoke existing form with Point as left operand
}

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

Why?

overloading print

cout << pt;

