Announcements
Watch lecture video on c++ templates this weekend.

```cpp
class sphere {
public:
sphere();
sphere(double r);
double getVolume();
void setRadius(double r);
void display();
private:
double theRadius;};

class ball:public sphere {
public:
bball();
bball(double r string n);
string getName();
void setName(string n);
void display();
private:
string name;};
sphere s;
bball b;
s=b;
b=s;
sphere * s;
bball * b;
b=d;
d=b;
```
something to consider:

class sphere {
public:
    sphere();
    sphere(double r);
    ...;

    double getVolume();
    void setRadius(double r);
    ...;

    void display();
private:
    double theRadius;
};

class ball:public sphere {
public:
    ball();
    ball(double r string n);
    ...;

    string getName();
    void setName(string n);
    ...;

    void display();
private:
    string name;
};

void sphere::display() {
    cout << "sphere" << endl;
}

void ball::display() {
    cout << "ball" << endl;
}

ex1
sphere s;
bball b;
s.display();
b.display();

ex2
sphere * sptr;
sptr = &s;
sptr->display();

ex3
sphere * sptr;
sptr = &b;
sptr->display();
class sphere {
  public:
    sphere();
    sphere(double r);
    ...
    double getVolume();
    void setRadius(double r);
    ...
    void display();
  private:
    double theRadius;
};

class ball: public sphere {
  public:
    ball();
    ball(double r string n);
    string getName();
    ...
    void display();
  private:
    string name;
};

"virtual" functions:

void sphere::display() {
  cout << "sphere" << endl;
}

void ball::display() {
  cout << "ball" << endl;
}

if (a==0)
  sptr = &s;
else sptr = &b;
sptr->display();
virtual functions – the rules:

A virtual method is one a ______________ can override.

A class’s virtual methods ________ be implemented. If not, then the class is an “abstract base class” and no objects of that type can be declared.

A derived class is not required to override an existing implementation of an ______________ virtual method.

Constructors ______________ be virtual

Destructors can and ____________ virtual

Virtual method return type ____________ be overwritten.
Constructors for derived class:

```cpp
ball::ball(): sphere()
{
    name = "not known";
}

ball::ball(double r, string n): sphere(r)
{
    name = n;
}

ball b;
ball b(0.5,"grape");
```
## “virtual” destructors:

```cpp
class Base{
    public:
        Base(){cout<<"Ctor: B"<<endl;}
        ~Base(){cout<<"Dtor: B"<<endl;}
};

class Derived: public Base{
    public:
        Derived(){cout<<"Ctor: D"<<endl;}
        ~Derived(){cout<<"Dtor: D"<<endl;}
};

void main(){
    Base * V = new Derived();
    delete V;
}
```
Abstract Base Classes:

class flower {
public:
    flower();
    virtual void drawBlossom() = 0;
    virtual void drawStem() = 0;
    virtual void drawFoliage() = 0;
    ... 
};

class daisy : public flower {
public:
    virtual void drawBlossom();
    virtual void drawStem();
    virtual void drawFoliage();
    ... 
private:
    int blossom; // number of petals
    int stem; // length of stem
    int foliage // leaves per inch
};

void daisy::drawBlossom() { // whatever
}
void daisy::drawStem() { // whatever
}
void daisy::drawFoliage() { // whatever
}

flower f;
daisy d;
flower * fptr;
Concluding remarks on inheritance:

Polymorphism: objects of different types can employ methods of the same name and parameterization.

```cpp
animal ** farm;
farm = new animal*[5];
farm[0] = new dog;
farm[1] = new pig;
farm[2] = new horse;
farm[3] = new cow;
farm[4] = new duck;

for (int i=0; i<5; i++)
    farm[i]->speak();
```

Inheritance provides DYNAMIC polymorphism—type dependent functions can be selected at run-time. Wikipedia: Polymorphism in OOP

Next topic: “templates” are C++ implementation of static polymorphism, where type dependent functions are chosen at compile-time.