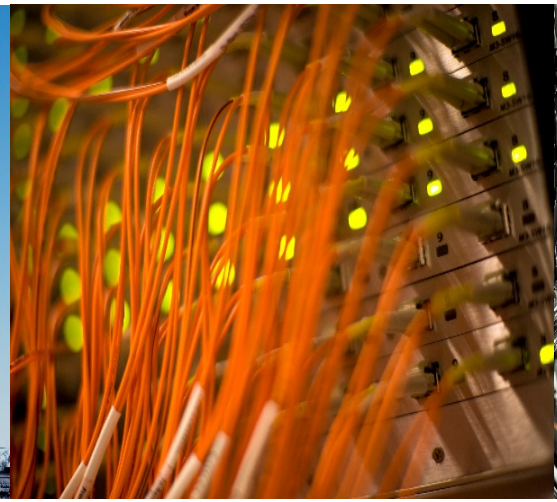
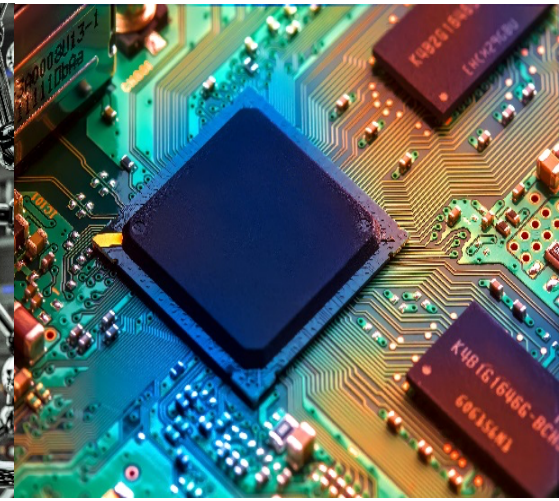
ECE 220 Computer Systems & Programming

Lecture 33 – C++ Polymorphism

August 3, 2020



ILLINOIS
Electrical & Computer Engineering
GRAINGER COLLEGE OF ENGINEERING

- Schedule your final exam with CBTF
- Final: 7pm on Friday, August 7th
- Conflict: 10:30am on Saturday, August 8th

What we've learned so far in C++

1. Abstraction
2. Encapsulation
3. Inheritance
4. Polymorphism

- Class vs. Struct
- Dynamic Memory Allocation
- Basic I/O
- Pass by Value vs. Pass by Reference vs. Pass by Address (pointer)
- Operator Overloading
- Base Class & Derived Class

Polymorphism

- a call to a member function will cause a **different function to be executed** depending on the type of the object that invokes the function

Example:

```
//base class
class Shape{
   protected:
   double width, height;
   public:
   Shape() {width = 1; height = 1;}
   Shape(double a, double b) { width = a; height = b; }
   double area() { cout << "Base class area unknown" << endl;
                   return 0; }
};
```

```
int main(){
    Rectangle rec(3,5);
    Triangle tri(4,5);

    rect.area();
    tri.area();

    return 0;
}
```

```
//derived classes
class Rectangle : public Shape{
   public:
   Rectangle(double a, double b) : Shape(a,b){}
   double area() {
      std::cout<<"Rectangle Object's area = "<<width*height<<endl;
      return width*height;
   }
};

class Triangle : public Shape{
   public:
   Triangle(double a, double b) : Shape(a,b){}
   double area() {
      std::cout<<"Triangle Object's area = "<<width*height/2<<endl;
      return width*height/2;
   }
};
```

Declared Type vs. Actual Type

```
int main(){
        Shape *ptr;
        Rectangle rec(3,5);
        Triangle tri(4,5);

        //use ptr to point to rec object
        ptr = &rec;
        ptr->area();

        //use ptr to point to tri object
        ptr = &tri;
        ptr->area();

        return 0;
}
```

What would this program print?

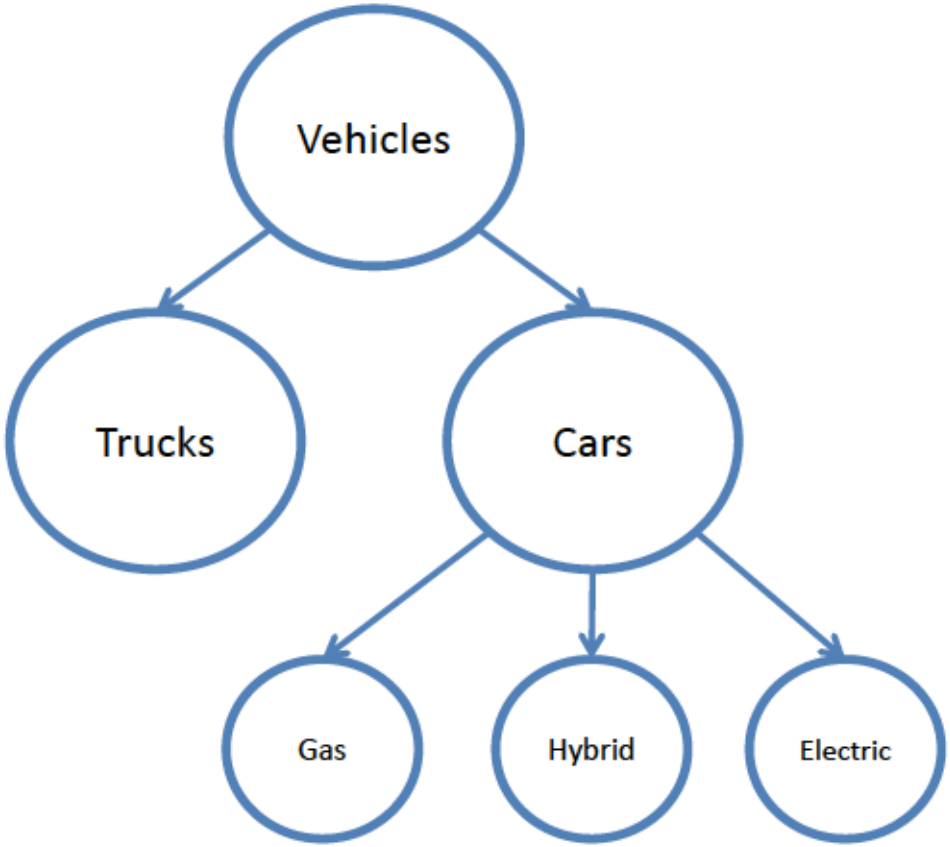
Abstract Base Class & Pure Virtual Functions

```
class Shape{
   protected:
   int width, height;
   public:
   Shape(int a, int b) { width = a; height = b; }
   virtual int area()=0; //pure virtual function – it has no body
};
```

```
int main(){
    Shape shape1(2,4); // this will cause compiler error!
    Shape *p_shape1; // this is allowed
}
```

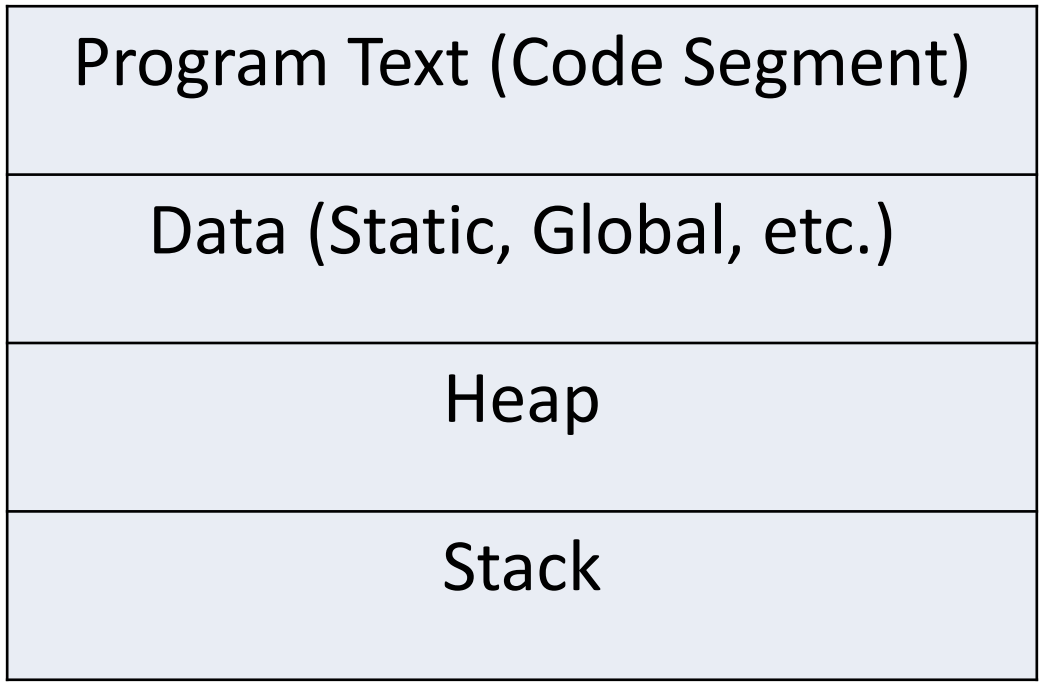
- derived class must define a body for this virtual function, or it will also be considered an abstract base class

Abstract Base Class

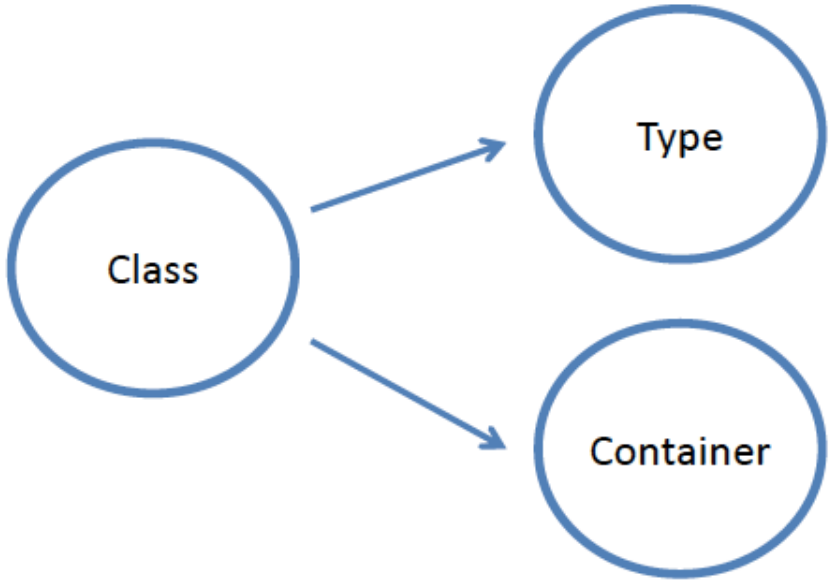


- The class Vehicle is not really an actual object
- We'll never have an actual Vehicle that isn't a more specific version of a Vehicle
- The class Vehicle is only used as a base class for other classes

Where are things being stored?



Templates



- Separate type from the structure
- Type focuses on data values, basic operators
- Container focuses on data structure (stack, list, array, tree, etc…)

Function Template

```
//functions can have the same names (overload)
int sum(int a, int b){
        return a+b;
}

double sum(double a, double b){
        return a+b;
}


//define function with generic type instead!
template <class T>
T sum (T a, T b){
        return a+b;
}

int main(){
        cout << sum(5,7) << endl;
        cout << sum(1.5, 2.7) << endl;
}
```

Class Template

```
template <class T>
class mypair {
   T a, b;
   public:
   mypair (T first, T second)
      {a=first; b=second;}
   T getmax ();
};

template <class T>
T mypair<T>::getmax () {
   T retval;
   retval = a>b? a : b;
   return retval;
}
```

```
int main () {
    mypair <int> myobject (100, 75);
    cout << myobject.getmax();
    return 0;
}
```

Friend Function and Class

- Allows outside function/class to access a class' private and protected members
- "Friendship" is one-way