Classes (Object Orientation) Data

Chapter 4 Revisited

Motivation

- Imagine storing a database of customer information in memory
  - Many data fields of different types (ints, floats, strings, etc.)
  - Potentially millions of Customers
  - Imagine passing all the data to a function

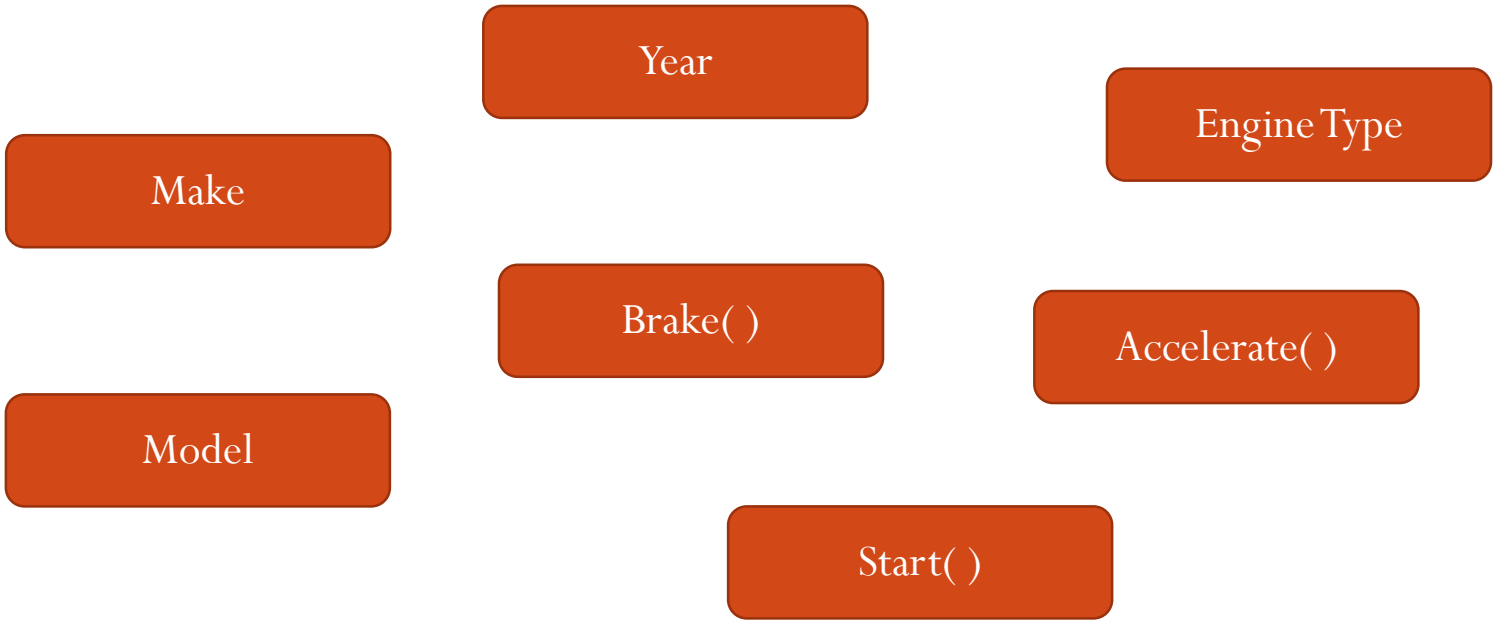
- How would you achieve this with the knowledge that you currently know?
  - An array for each data field

Classes

- Custom Data Type

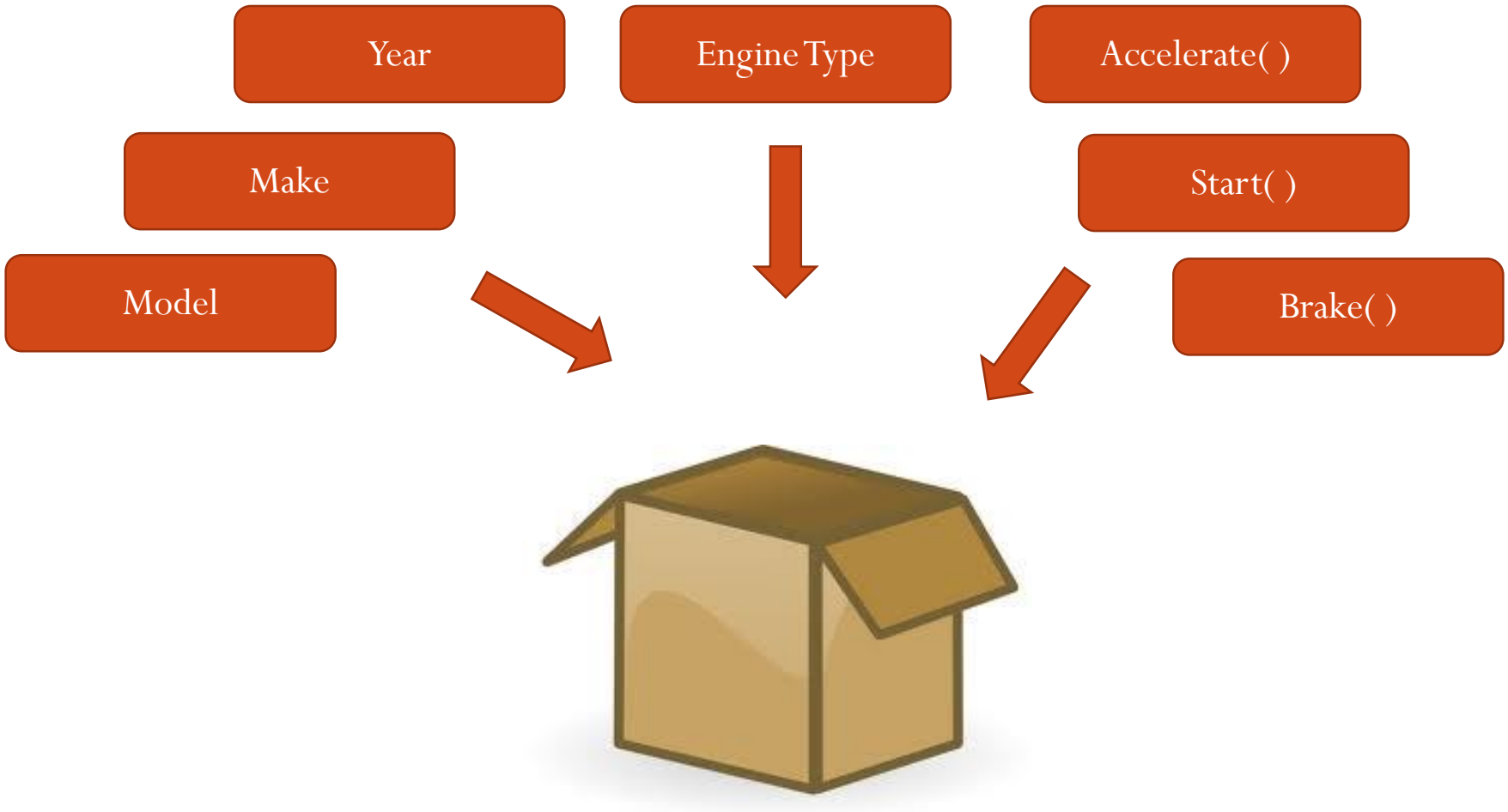
- Organizes data and code into concepts
  - Allows for modification of internal variables without passing data
  - Groups data with functions
  - Limits access privileges providing more secure coding methods
  - Greater code reuse through inheritance and polymorphism (ECE 321 Topic)
  - Allows greater flexibility and modularity of code through the use of abstract interfaces (ECE 321 Topic)

Car Concept Without Classes

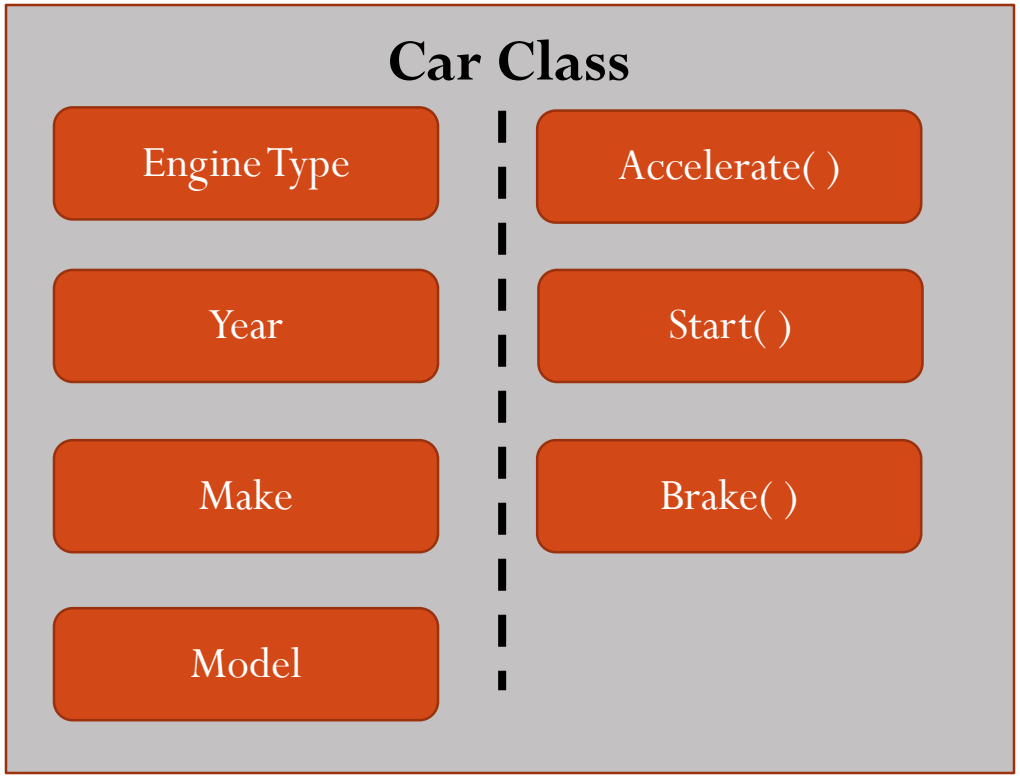


Data and functions are scattered throughout the code and programmer is required to keep track of which data is which.

With Classes, data and functions are organized in a container



Classes



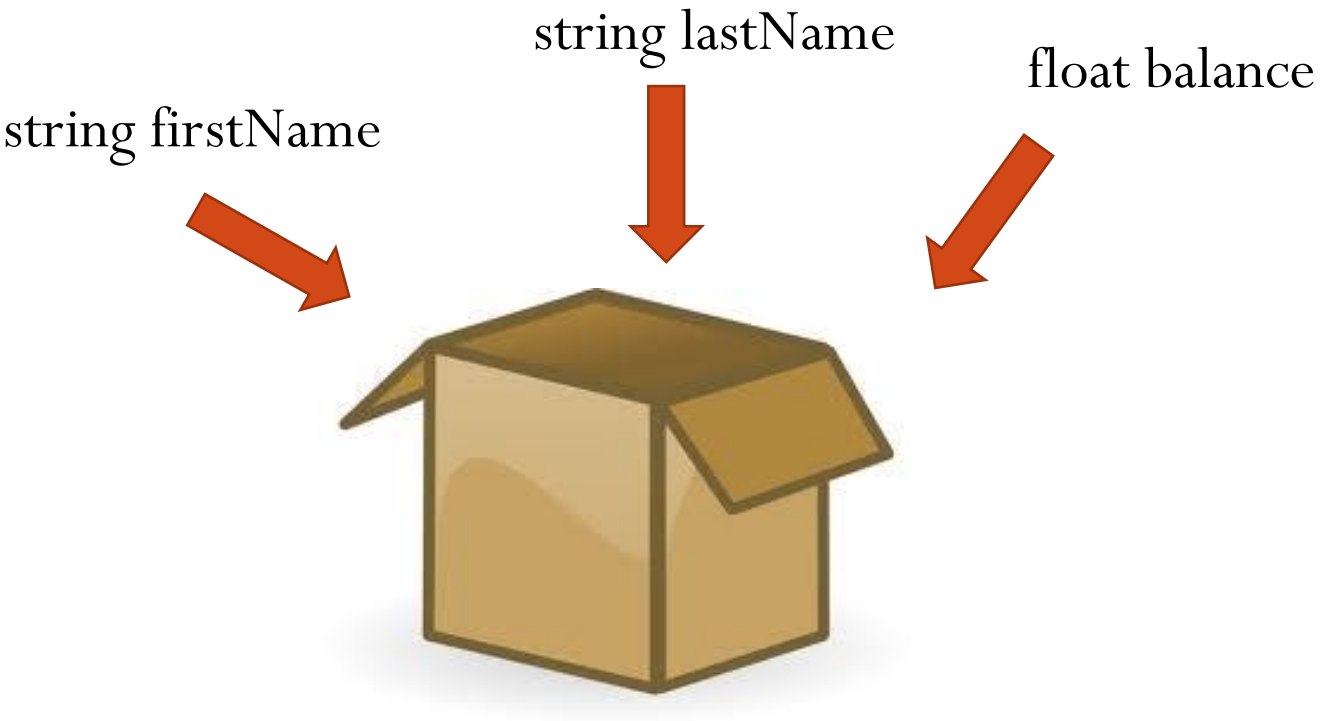
Data Members

Function Members (Methods)

Defining a Class

```
class BankCustomer {
public:
   string firstName;
   string lastName;
   float balance;
};
```

```
class name_of_class {
public:
   datatype data1;
   datatype data2;
   …
};
```



Declaring an Instance of a Class

- Class is a definition of the data structure

- Object is an instance of a class
  - Since a class is merely a definition, it cannot hold any data
  - Only when an object of a class is created can data be stored
  - There can be multiple objects of a given class

```
int main()
{
  BankCustomer cust1;
  BankCustomer cust2;
  BankCustomer larry;
  …
```

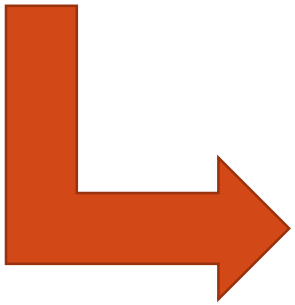
```
int main()
{
  name_of_class object_name1;
  name_of_class object_name2;
  name_of_class object_name3;
  …
```

Dot Operator

- Gives access to an object's members

Placing Values into Object

```
int main()
{
    BankAccount cust;
    cust.firstName = "Walter";
    cust.lastName = "White";
    cust.balance = 3500000;
    …
```



string firstName	Walter
string lastName	White
float balance	3500000

cust1

Retrieving Values from Object

Fetching Steps:

1. Select Structure
2. Select item in structure

string firstName	Walter
string lastName	White
float balance	3500000

cust

```
string someName;
cout << cust.firstName << endl;
cout << cust.lastName << endl;
cout << cust.balance << endl;
someName = cust.firstName + " " + cust.lastName;
```

Multiple Objects of same Class

```
int main()
{
    BankCustomer cust1, cust2;
    cust1.firstName = "Walter";
    cust1.lastName = "White";
    cust1.balance = 3500000;

    cust2.firstName = "Jesse";
    cust2.lastName = "Pinkman";
    cust2.balance = 2500;

…
```

string firstName	Walter
string lastName	White
float balance	3500000

cust1

string firstName	Jesse
string lastName	Pinkman
float balance	2500

cust2

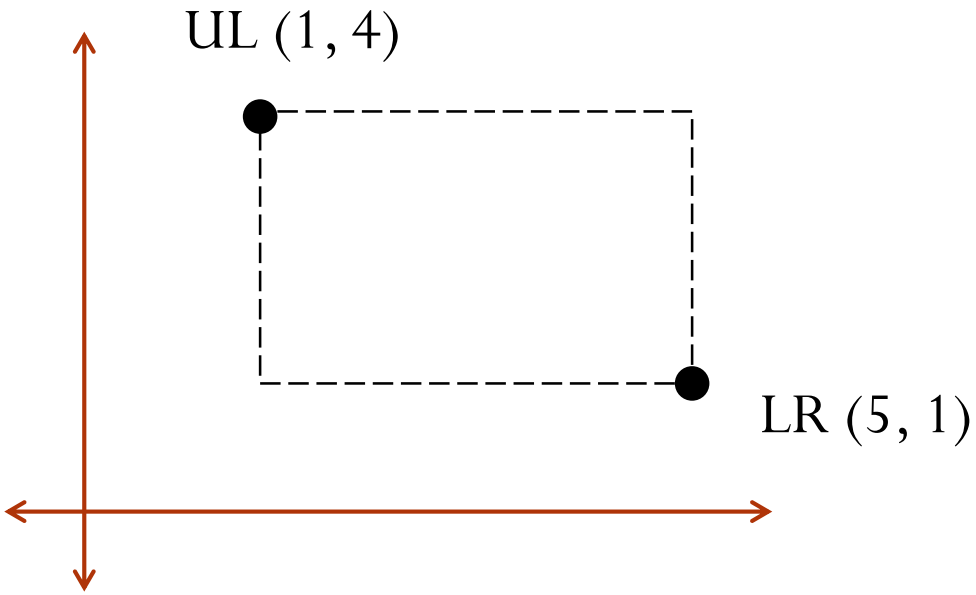
Example 9.1

- Create a Class the represents a point on a Cartesian Grid
  - Declare an instance and fill that data with point (1,4)
  - Declare another instance and fill the data with point (2,3)
  - Calculate the distance between two points and print the result to the console

$$d = \sqrt{(\Delta x)^2 + (\Delta y)^2} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$$

Nested Classes

- A rectangle on a grid can be stored as two points, upper left and lower right
- Lets use the point structure in the previous example to generate a new rectangle class

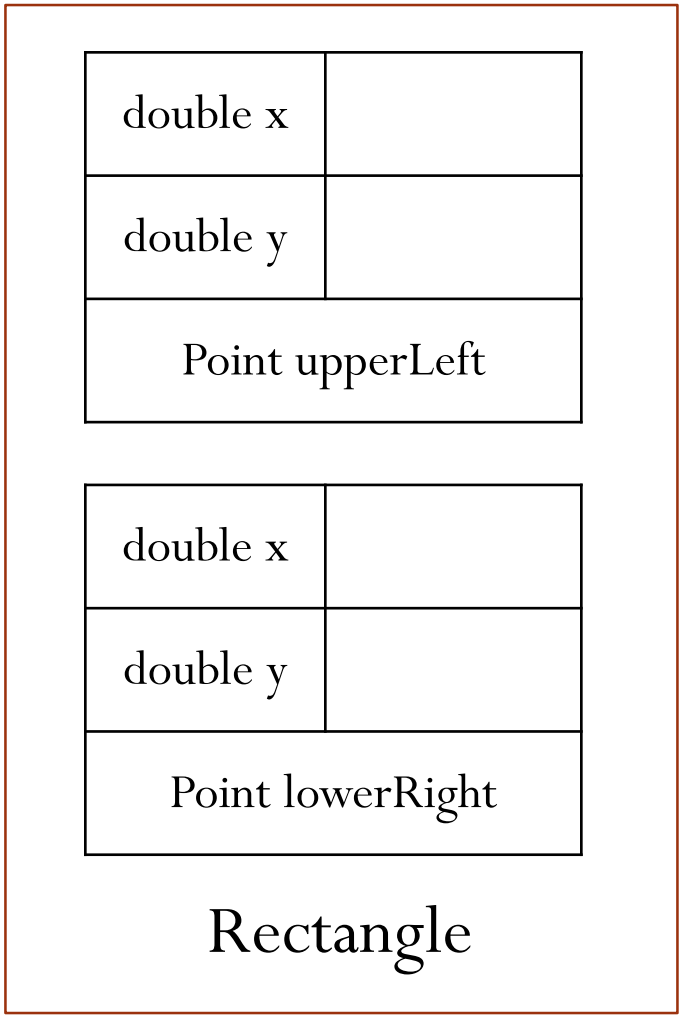


Nested Structures

```
class Point {
  public:
  double x;
  double y;
};

class Rectangle {
  public:
  Point upperLeft;
  Point lowerRight;
};
```

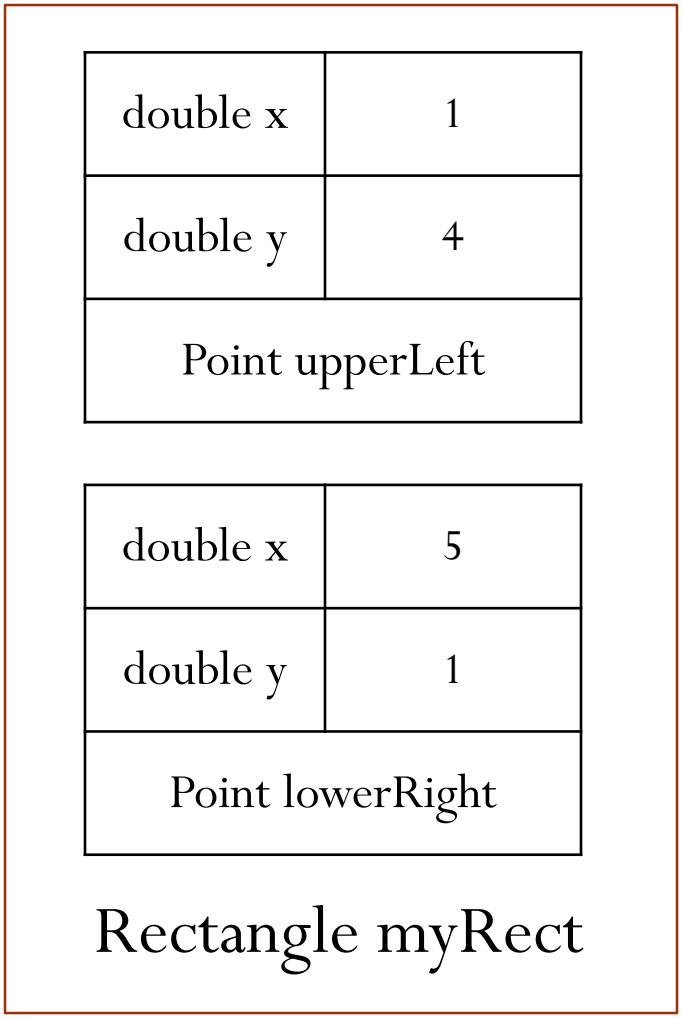
- Order of definitions matters
- Compiler must know of Point prior to creating Rectangle therefore Point must be defined before Rectangle



Declaring and Using Nested

```
int main()
{
   Rectangle myRect;
   myRect.upperLeft.x = 1;
   myRect.upperLeft.y = 4;
   myRect.lowerRight.x = 5;
   myRect.lowerRight.y = 1;
    …
```

- Every level of structure must be navigated based on the structure hierarchy

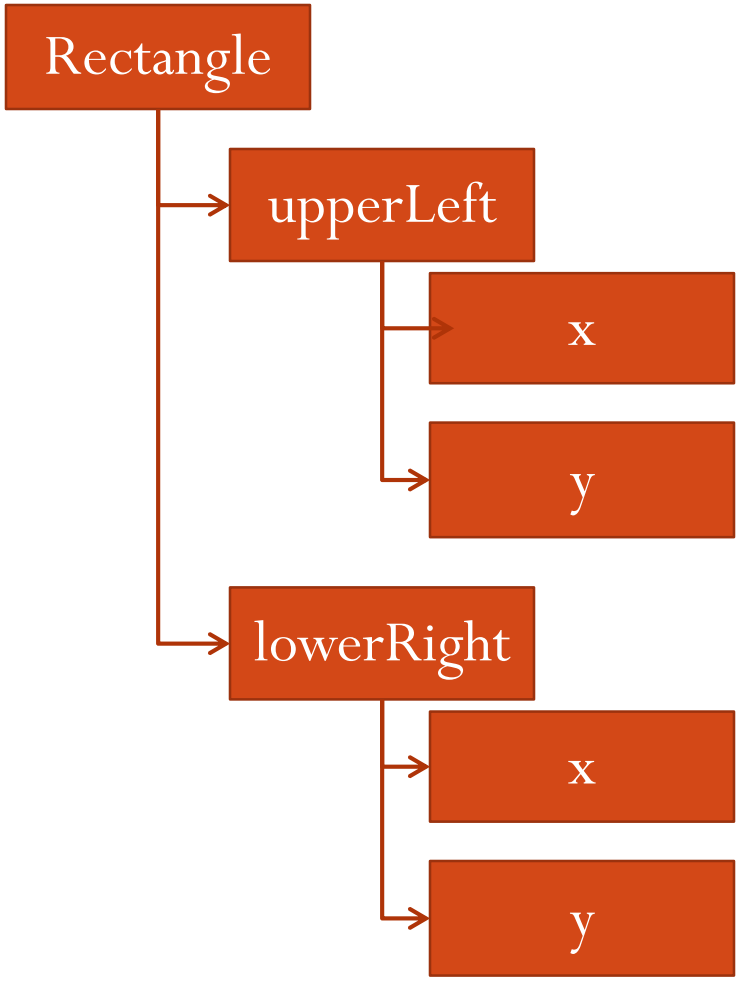


Class Hierarchy

```
class Point {
  public:
  double x;
  double y;
};

class Rectangle {
  public:
  Point upperLeft;
  Point lowerRight;
};

int main()
{
  Rectangle myRect;
  myRect.upperLeft.x = 1;
  myRect.upperLeft.y = 4;
  myRect.lowerRight.x = 5;
  myRect.lowerRight.x = 1;
   …
```



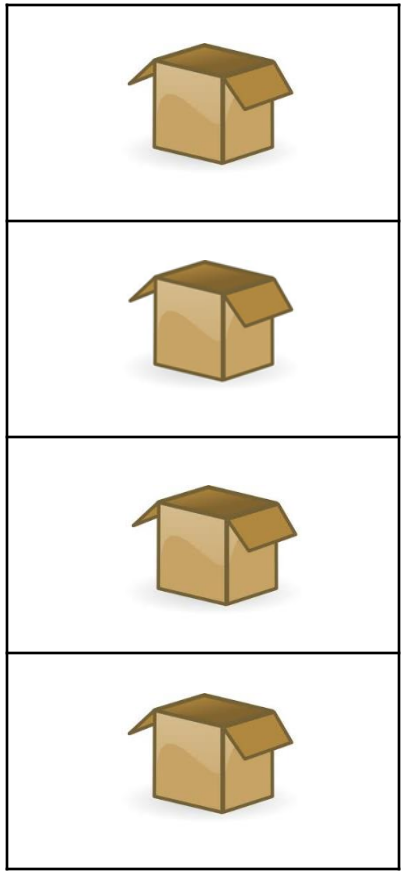
Arrays Of Classes

- Returning to Customer Example

```
int main()
{
    Customer custs[4];

    custs[0].name = “luke”;
    custs[0].age = 72;
    …


    custs[1].name = “alfred”;
    …
```



Arrays Of Classes

```
int main()
{
    Customer custs[4];

    custs[0].name = “luke”;
    custs[0].age = 72;
    …


    custs[1].name = “alfred”;
    …
```

- Creates several structures inside the array
- To access:
  1. Select structure from array
  2. Select field from structure

Passing Class to Function

```
void printCustomer(Customer cust)
{
    cout << cust.name << endl;
    cout << cust.age << endl;
    cout << cust.phoneNum << endl;
    return;
}


int main()
{
    Customer firstCust;
    firstCust = createCustomer("luke", 72, “867-5309");
    printCustomer(firstCust);
    return 0;
}
```

Returning Class From Function

```
Customer createCustomer(string name, int age, string phoneNum)
{
    Customer cust;
    cust.name = name;
    cust.age = age;
    cust.phoneNum = phoneNum;
    return cust;
}

int main()
{
  Customer firstCust;
  firstCust = createCustomer("luke", 72, “867-5309");
  cout << firstCust.name << endl;
  return 0;
}
```

Classes and Functions

- Benefits:
  - It is possible to pass a lot of data implicitly by putting it in a structure
  - Several pieces of data can be returned from a single function if grouped into a single structure
  - In general: classes allow for better organization in code
- Negatives:
  - Classes can be HUGE and it can be very memory and computationally expensive to pass all that data
    - It is very common to pass large classes by reference to avoid such expense