CS106B Chris Piech
Winter 2016 March 12th, 2016

Structs and Classes

To build great programs, often you will want to define your own variable types. In CS106B we have introduced two ways to do so: with structs and classes. When you define Stucts or Classes you are providing a blueprint that tells the computer how to make variables of that type.

Structs

A struct makes a variable type that packages a set of "instance" variables together. For example

```
struct Student {
  string name;
  int age;
  Vector<string> classes;
};
```

creates a new variable type called Student. Each time you create a new variable of type student, that variable will have its own copy of each instance-variable. You can access them using the . operator (or the -> operator if you are using pointers. More on that later).

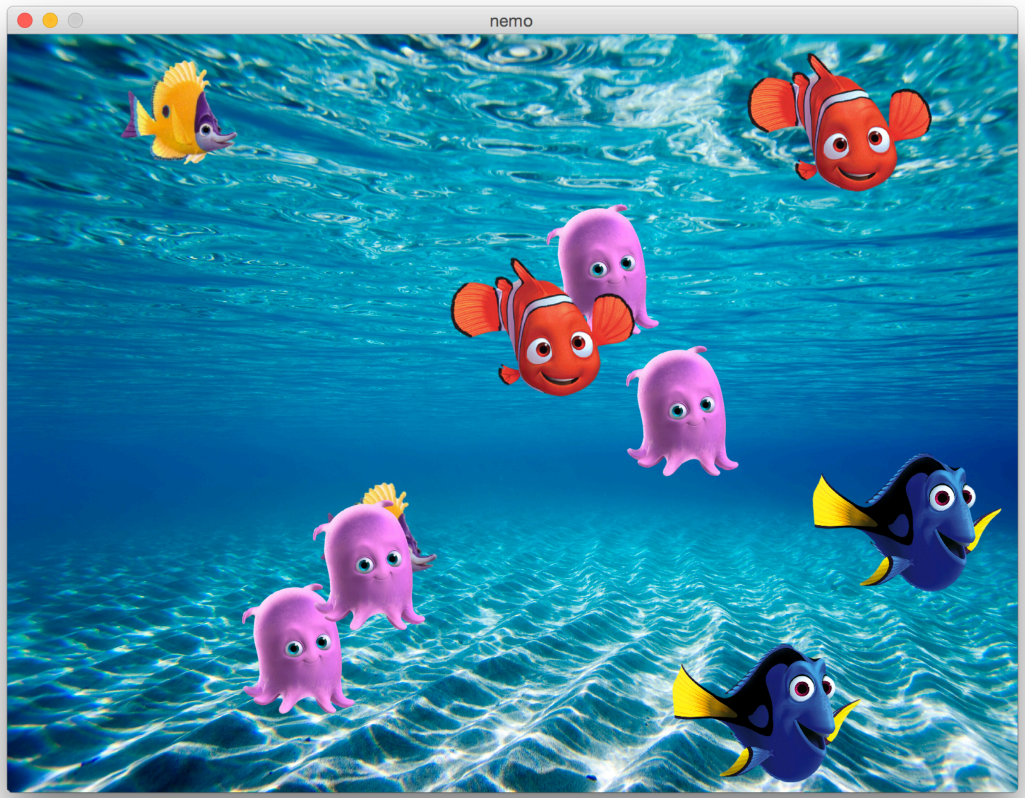
```
Student chris;
chris.name = "Chris Piech"
chris.classes.add("CS106B");
cout << chris.name << endl;
```

Classes

Structs leave a few things to be desired. Often you don't want to expose all your instance variables directly to the user of the variable type (think of a Vector which has a complex representation under the hood that the user doesn't need to know about and shouldn't mess with). And in addition its common to want to add methods that you can call on your new variable type. A class is an extension of a struct that allows for extra functionality.

In order to create a new variable type as a class you need to specify four things (1) the instance variables, just like a struct (2) what happens when a new object of your type is created/destroyed (3) what are the methods that a user can call on an instance of that class. Finally (4) you have to implement the methods. You write the code that answers these four things in two files: a dot h and a dot cpp.

In the following example we are going to create a variable to describe a fish in a fishtank:



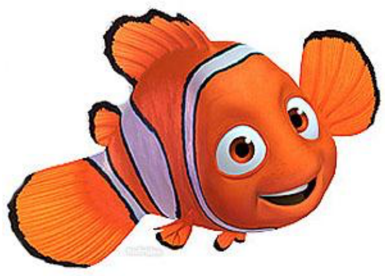
The program would be so much easier to write if we had a variable called Fish. It could take in the filename of an image and then it could encapsulate all the information we would want to associate with each fish such as where it is and where it's going.

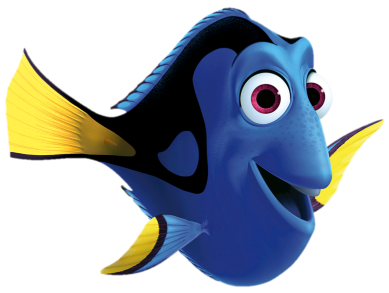
The Dot h

In a file with name <type>.h you define: (1) the instance variables, just like a struct, (2) what parameters must be provided when creating a new object (3) what are the methods that a user can call on an instance of that class.

```
class Fish {
public:
    // how to make a new fish (task 3)
    Fish(string fileName, GWindow * window);
    ~Fish();

    // animate one frame of swimming (task 2)
    void swim();

    // sometimes you have to rethink it (task 2).
    void choseNewDirection();

private:
    // The list of variables here describe all the variables that each
    // fish instance has. Just like a struct.

    // where is this fish now?
    double x;
    double y;

    // where is this fish going?
    double dx;
    double dy;

    // all fish point to the same GWindow.
    GWindow * tank;

    // GImages are always stored on the heap
    GImage * image;
};
```

That's a blueprint of a fish. You could use it to make these three different ones:





The Dot cpp

In a file with name <type>.cpp you define each of the methods including the constructor (that's the name of the method called when you a new object of that type is created) and the destructor (the name of the method called when an object is destroyed)

```
Fish::Fish(string filename, GWindow * window) {
  // Image is an instance variable. Each fish has one.
  image = new GImage(filename);
  int maxX = window->getWidth() - image->getWidth();
  int maxY = window->getHeight() - image->getHeight();
  x = randomInteger(0, maxX);
  y = randomInteger(0, maxY);
  window->add(image, x, y);
  tank = window;
  choseNewDirection();
}

Fish::~Fish() {
   // it is your responsibility to delete everything you allocate.
   delete image;
}

void Fish::swim() {
  double oldX = x;
  double oldY = y;
  x += dx;
  y += dy;
  bool collision = false;
  if(x < 0 || x > tank->getWidth() - image->getWidth()) {
    collision = true;
  }
  if(y < 0 || y > tank->getHeight() - image->getHeight()) {
    collision = true;
  }
  if(collision) {
    x = oldX;
    y = oldY;
    choseNewDirection();
  }

  image->setLocation(x, y);
}
```

(continued on next page...)

```
void Fish::choseNewDirection() {
    dx = randomInteger(2, 4);
    dy = randomInteger(2, 4);
    if(randomBool()) dx *= -1;
    if(randomBool()) dy *= -1;
}
```

That's all

Classes can sound harder than they are because of all of the syntax. But remember that they are just glorified structs. The package together instance variables and allow you to define what methods can be called on a variable of the newly defined time.

In C++ you can also "overload operators". That lets you do things like define what it means to use the + operator on a variable of your type. On the final I won't ask you to do so.

For full measure, I want to emphasize that once a class is declared, the user now has a brand new variable type. Here is a potential main function

```
int main() {
  GWindow * window = new GWindow(WINDOW_WIDTH, WINDOW_HEIGHT);
  addBackground(window);

  Vector<Fish*> allFish;
  for(int i = 0; i < N_FISH; i++) {
    // make a new Fish. Call the constructor.
    Fish * newFish = new Fish(getRandomFishFile(), window);
    allFish.add(newFish);
  }

  while(true) {
    Timer timer;
    timer.start();
    for(Fish *f : allFish) {
      // call the swim method!
      f->swim();
    }
    pause(PAUSE_MS - timer.stop());
  }

  return 0;
}
```