Object-Oriented Programming

Procedural programming

```
#include <iostream>
#include <cstdlib>
#include <cmath>

using namespace std;

double f(double x) {
   // return 10.0 * x * x - 3.0 * x;
   // return x*x*x - 2*x + 2;
    return sin(x*x) - 0.5;
}

double df(double x)
{
    //return 20.0 * x - 3.0;
    //return 3.0*x*x - 2.0;
    return 2.0 * cos(x*x);
}

double newton(int n, double a, double tol){
    double fx = f(a);
    double dfx;
    double x = a;
    int count = 0;
    while((fabs(fx) > tol) && (count++ < n)){
        fx = f(x);
        dfx = df(x);
        //cout << count << " " << x << " " << fx << endl;
        x = x - fx/dfx;
    }
    cout << count << " " << x << " " << fx << endl;
    return x;
}

double bisection(int n, double a, double b, double tol) {
    double fa = f(a);
    double fb = f(b);
    double c;
    double fc;
```

Procedural programming can sometimes be used as a synonym for imperative programming (specifying the steps the program must take to reach the desired state), but can also refer to a programming paradigm, derived from structured programming, based upon the concept of the *procedure call*.

http://blog.ircmaxell.com/2012/07/oop-vs-procedural-code.html

Object-oriented programming

```php
class Mediator {
    protected $events = array();
    public function attach($eventName, $callback) {
        if (!isset($this->events[$eventName])) {
            $this->events[$eventName] = array();
        }
        $this->events[$eventName][] = $callback;
    }
    public function trigger($eventName, $data = null) {
        foreach ($this->events[$eventName] as $callback) {
            $callback($eventName, $data);
        }
    }
}
$mediator = new Mediator;
$mediator->attach('load', function() { echo "Loading"; });
$mediator->attach('stop', function() { echo "Stopping"; });
$mediator->attach('stop', function() { echo "Stopped"; });
$mediator->trigger('load'); // prints "Loading"
$mediator->trigger('stop'); // prints "StoppingStopped"
```

Object-oriented programming (**OOP**) is a programming paradigm using "objects" – data structures consisting of data fields and methods together with their interactions – to design applications and computer programs. Programming techniques may include features such as data abstraction, encapsulation, messaging, modularity, polymorphism, and inheritance.

http://blog.ircmaxell.com/2012/07/oop-vs-procedural-code.html

```
8  #include <cstdlib>
9  #include <string>
10
11 using namespace std;
12
13 struct animal {
14       String genus;
15       String species;
16       double weight;
17       double length;
18       double age;
19       String bodycover;
20       String voice;
21 };
22
23 struct plant {
24       String genus;
25       String species;
26       double weight;
27       double length;
28       double age;
29       String pollinator;
30 };
31
32 int main(int argc, char** argv) {
33
34       animal lion;
35       plant acacia;
36       lion.weight = 420.0;
37       lion.voice = "ROARRR";
38       acacia.genus = "Acacia";
39       acacia.species = "tortilis";
40       cout << acacia.genus << " " << acacia.species << endl;
41
42       return 0;
43 }
```

Data structure

struct name **{**
    variable definitions;
};

Objects are augmented data structures that not only contain variables but also functions that operate on these variables

Object
- variables
- functions
  - manipulation
  - input/output
  - creation
  - destruction

- Objects are instances of a general description the class

- We define the class with variables and functions, there is a rich set of allowing access to data within the class or whether other objects can access parts or all of the innards of an object.

- Keywords: class, instance, private, protected, public, member function.

```
class Animal {
     string genus;
     string species;
     string voice;
public:
     Animal() {};
     Animal(string g, string s);
     ~Animal(){};
     void setVoice(string a) { voice=a;};
     string showVoice() ;
};

Animal::Animal(string g, string s)
{
     genus = g;
     species = s;
}

string Animal::showVoice(){
      return "The " + genus + " " + species + " says " + voice;
}
```

```
class Living{
protected:
    string genus;
    string species;
public:
    Living() {
    genus = "Unknown";
    species = "Species";
    };
    Living(string g, string s);
    ~Living(){};
    string name() { return genus + " " + species ; };
};

Living::Living(string g, string s) {
    genus = g;
    species = s;
}

class Animal : public Living {
    string voice;
public:
    Animal(): Living() {};
    Animal(string g, string s): Living(g,s) {};
    ~Animal() {
    };
    void setVoice(string a) {
        voice = a;
    };
    string showVoice();
};

string Animal::showVoice() {
    return "The " + genus + " " + species + " says " + voice;
}

int main(int argc, char** argv) {
    Animal lion;
    lion.setVoice("ROARRR");
    Animal tiger("Tigris", "pardalis");
    tiger.setVoice("MIAU");
    cout << tiger.showVoice() << endl;
    cout << lion.showVoice() << endl;
    return 0;
}
```