Error Checking

Does the file exist? Rename data2.txt to data two.txt and run program again

```
// ifstream::is_open
#include <iostream>     // std::cout
#include <fstream>      // std::ifstream
int main () {
   std::ifstream ifs ("test.txt");

   if (ifs.is_open()) {
    // print file:
     char c = ifs.get();
     while (ifs.good()) {
       std::cout << c;
       c = ifs.get();
     }
   }
   else {
    // show message:
     std::cout << "Error opening file";
  }
 return 0;
}
```

Public member functions:

good(), bad() :state of data stream
get(), getline(), read()
open(), is_open(), close()

Enumerated Types

```
enum type_name {
  value1,
  value2,
  value3,
  .
  .
} object_names;
```

```
enum colors_t {black, blue, green, cyan, red, purple, yellow, white};
colors_t mycolor;          //where black= 0, blue =1, etc

mycolor = blue;
if (mycolor == green) mycolor = red;

enum months_t { january=1, february, march, april,
                may, june, july, august,
                september, october, november, december} yr;  //starts with 1
```

Templates for Functions

Problem:

```
 // Function to double the number: PrintTwice.cpp
#include <iostream>
using namespace std;

void PrintTwice(int data)
 {
   cout << "Twice is: " << 2*data <<endl;
   }

int main ()
{
  int a = 12;
  float b = 12.2;
    PrintTwice(a);
    PrintTwice(b);  <— calling with float
  return 0;
}
```

Use Template

```
//PrintTwiceTemplate.cpp
#include <iostream>
using namespace std;

template<class TYPE>                    //function template
void PrintTwice(TYPE data)
 {
   cout << "Twice is: " << 2*data <<endl;
   }

int main ()
{
  int a = 12;
  float b = 12.2;
  PrintTwice(a);                  //TYPE replaced by int as a is int
  PrintTwice(b);
  return 0;
}
```

Avoids having to duplicate the function for integers and floats

C++ Template example

```
template <typename Type>
Type max(Type a, Type b) {
    return (a > b) ? a : b;
}

include <iostream>

int main()
{
  // This will call max <int>       (by argument deduction)
  std::cout << max(3, 7) << std::endl;

  // This will call max<float>      (by argument deduction)
  std::cout << max(3.0, 7.0) << std::endl;

  // This type is ambiguous, so explicitly instantiate max<float>
  std::cout << max<float>(3, 7.0) << std::endl;
  return 0;
}
```

Class Problem

Write a template function for an averaging function.

It should work for integers, floats, and doubles

Given:

```
int main()
{
    int  IntArray[5] = {100, 200, 400, 500, 1000};
    float FloatArray[3] = { 1.55f, 5.44f, 12.36f};

    cout << getAverage(IntArray, 5);
    cout << getAverage(FloatArray, 3);
}
```

```
template <class T>
double getAverage( T tArray[], int  nelements)
 {
   T tSum = T();  // tSum=0
   for (int i = 0; i< nelements; i++)
     { tSum += tArray[i];
     }
   return double(tSum) / nelements;
}
```

440

6.45

Classes: Object-oriented Programmng

Like structures, holding different types of data, but also including functions to access the data. They are standalone objects.

Objects are instantiation of a class.

Classes are distinguished by **class** keyword

```
class Rectangle {
    int width, height;
  public:
    void set_values (int,int);
    int area (void);
} rect;
```

variables called fields of the class; private

Prototypes of the class methods, accessible from outside class

rect is an object of class Rectangle

```
rect.set_values(5,5);                    //using methods of the class
int myarea = rect.area();
```

Can have pointers to class

http://www.cplusplus.com/doc/tutorial/classes/

Classes

```
// classes example
#include <iostream>
using namespace std;

class Rectangle {
    int width, height;
  public:
    void set_values (int,int);
    int area() {return width*height;}
};
```

Rectangle defines two functions: one by prototype and a simple one

```
void Rectangle::set_values (int x, int y) {
  width = x;
  height = y;
}
```

Methods defined outside of class.
Note: :: scope operator, which allows access to private variables

```
int main () {
  Rectangle rect1, rect2;       // declare rect1 and rect2 as Rectangle class
  rect1.set_values (3,4);       // . operator allows use to reference field variables
  cout << "area1: " << rect1.area();
  rect2.set_values(4,5);        // different area
  cout << "area2: " << rect2.area();
  return 0;
}
```

Constructor

```
// example: class constructor
#include <iostream>
using namespace std;
 class Rectangle {
    int width, height;
  public:                              // constructor needs to be public
    Rectangle (int,int);               // replace the set_values function
    int area () {return (width*height);}
};

Rectangle::Rectangle (int a, int b) { //no type for constructor; no return
  width = a;
  height = b;
}

int main () {
  Rectangle rect (3,4);
  Rectangle rectb (5,6);
  cout << "rect area: " << rect.area() << endl;
  cout << "rectb area: " << rectb.area() << endl;
  return 0;
}
```

If called area, without first calling set_value, problems arise. So use constructor

Constructor only used when object instantiated, it is not a function

Write class object to calculate the cost of parking

A public parking garage needs a program to calculate parking costs: the input is the **time_length** that a car has been parked, and the cost is $1.50 for the first hour and $1.20 for the remaining hours.

The program should have a Calculator class that uses input integer hours in the main() to calculate the hours parked and then a function to calculate the parking fee.

Parking Calculator Class

```
#include <iostream>
using namespace std;

class calculate
{
	double hours;
	double total;
public:
	calculate(int);
	void findTotal();
};
calculate::calculate(int data)
{
	total = 1.5;
	hours = data - 1;
}
void calculate::findTotal(void)
{
	while (hours > 0)
	{
		total += 1.2;
		hours--;
	}
	cout << "You owe $" << total << endl;
}
```

Main Parking

```
int main()
 {

      int time_length;
      cout << "How many hours were you parked: \n";
      cin >> time_length;
      calculate total(time_length);  //create an instance of calculate after time_length
                           //is known; what happens if I defined calculate ahead of
                           //read statements
      total.findTotal();
      return 0;
}
```

Constructor

Overloaded constructors

```
// overloading class constructors
#include <iostream>
using namespace std;

class Rectangle {
    int width, height;
  public:
    Rectangle ();
    Rectangle (int,int);
    int area (void) {return (width*height);}
};

Rectangle::Rectangle () {
  width = 5;
  height = 5;
}

Rectangle::Rectangle (int a, int b) {
  width = a;
  height = b;
}

int main () {
  Rectangle rect (3,4);
  Rectangle rectb;
  cout << "rect area: " << rect.area() << endl;
  cout << "rectb area: " << rectb.area() <<
endl;
  return 0;
}
```

```
rect area: 12
rectb area: 25
```

http://www.tutorialspoint.com/cplusplus/cpp_constructor_destructor.htm

Constructor: Initialization

```
Rectangle::Rectangle (int x, int y) {width=x; height = y;}

Rectangle::Rectangle (int x, int y) : width(x) {height = y;}

Rectangle:: Rectangle (int x, int y) : width(x), height(y) {}
```

Destructor

```
#include <iostream>
#include <string>
using namespace std;

class Example4 {
    string* ptr;
  public:
    // constructors:
    Example4() : ptr(new string) {}
    Example4 (const string& str) : ptr(new string(str)) {}
    // destructor:
    ~Example4 () {delete ptr;}
    // access content:
    const string& content() const {return *ptr;}
};

int main () {
  Example4 foo;
  Example4 bar ("Example");

  cout << "bar's content: " << bar.content() << '\n';
  return 0;
}
```

```
bar's content: Example
```

Inheritance

```
// multiple inheritance
#include <iostream>
using namespace std;

class Polygon {
  protected:
    int width, height;
  public:
    Polygon (int a, int b) : width(a), height(b)
{}
};

class Output {
  public:
    static void print (int i);
};

void Output::print (int i) {
  cout << i << '\n';
}

class Rectangle: public Polygon, public Output {
  public:
    Rectangle (int a, int b) : Polygon(a,b) {}
    int area ()
      { return width*height; }
};

class Triangle: public Polygon, public Output {
  public:
    Triangle (int a, int b) : Polygon(a,b) {}
    int area ()
      { return width*height/2; }
};

int main () {
  Rectangle rect (4,5);
  Triangle trgl (4,5);
  rect.print (rect.area());
```

```
20
10
```

See https://www.cs.bu.edu/teaching/cpp/inheritance/intro/