

CS 1428
Fall 2018
Gentry Atkinson
Lab 12

Introduction:

It is very important for coders to be able to share code. However, it is difficult for a business to make money off of the code that they write if they share the source code freely because that makes it very easy to duplicate code. So how can a coder separate code into packages so that the interface is visible but the content of the code is not? The answer to this question is libraries. When code is transformed into a library it is separated into a header file, which shows the interface of the code so that others can use it, and the source file which can be compiled into an object file before it is distributed so that it is unreadable. Libraries also serve a useful purpose by creating summaries of the object you write for your code and separating out the source code which is often very long and unpleasant to read.

Classes serve a similar purpose in that they let you separate logical groups of data and actions which can be performed on that data. Although classes do not have to be written into their own libraries, that is the norm in C++. Storing data inside of objects lets a coder control how others interact with that data.

The purpose of today's lab is to introduce you to creating and using libraries and classes.

Directions:

1- Launch Code::Blocks and start a new project. Make it a **Console Application** and make its language C++. Name it `Your_last_name_lab12`.

2- The `main.cpp` file will be created for free. Select `File->New->Class`. Name it `Material` and deselect the box that says "Virtual Destructor". Select **Yes** to add it to your project. You should now see a `Material.h` and `Material.cpp` file added to your project directories on the left. You may have to expand the **src** and **include** directories to see them.

3- Copy the standard file header into all 3 files:

```
//Your Name  
//CS1428 Fall 2018  
//Lab 12
```

4- Iostream is already included in main.cpp (or should be). Add “Material.h” (**NOT** <Material.h>) to main.cpp.

5- You’ve been hired by a small engineering firm. They’re working to test the breaking and melting points of several different materials. They need you to write a system to track the data produced by the tests they are doing on the materials. Each material will have its melting point tested 3 times and its breaking point tested 3 times. They have prepared the following code to provide you with guidance. Copy this code into Materials.h:

```
class Material
{
    public:
        Material();
        ~Material();

        void addMelting(float);
        void addBreaking(float);
        float getAvgMelt();
        float getAvgBreak();
        void reset();

    private:
        //Holds the results of 3 melting tests
        float melting[3];

        //Holds the results of 3 breaking tests
        float breaking[3];

        //Pointer to the melt and break array
        //Should be the index of the next position to write
        int meltPointer, breakPointer;
};
```

6- This header file is your blueprint for everything that you’ll need to write into the source file **Materials.cpp**. Every function listed here needs to be defined in Materials.cpp. Everything under the **public** tag can be called by other code. Anything under the **private** tag can only be accessed by the one object that creates it. Copy the following code in Materials.cpp to get started:

```

Material::Material()
{
    //This is the constructor
    //Give every variable a default value here
}

```

```

Material::~~Material()
{
    //This is the destructor
    //It is not necessary for this object
}

```

```

void Material::addMelting(float input){
    //Add input to the melting array
    //Then increment the meltPointer value
    //Should not do anything if array is full
}

```

```

void Material::addBreaking(float input){
    //Add input to the breaking array
    //Then increment the breakPointer value
    //Should not do anything if array is full
}

```

```

float Material::getAvgMelt(){
    /******
    Return the mean of all the values in
    the melting array. The last guy wrote this.
    Should return 0 if the array is empty.
    Can you fix it?
    int sum;
    for (i = 0; i < index; i++){
        sum += melting[i];
        return (sum/index);
    }
    *****/
}

```

```

float Material::getAvgBreak(){
    //Same as melting but for breaking
}

```

```

    }
    void Material::reset(){
        //Reset both arrays
    }

```

7- Once you fully implemented the Material class create 3 **Materials** in your **main** function with this code:

```

Material gentrynium;
Material wonderfonium;
Material turingite;

```

Notice that your class can now be declared as a data type just like with the **structs** we made several weeks ago. Instantiating an object creates space in memory for all of the variables that you've added to the class.

8- To add the first value to one of your objects add this to your main function:

```

gentrynium.addMelting(144.12);

```

Now add the following values into your objects:

```

Gentrynium Melting Tests: 133.12, 152.15
Gentrynium Breaking Tests: 58.2, 44.28, 47.98
Wonderfonium Melting Tests: 159.85, 143.37, 120.99
Wonderfonium Breaking Tests: 67.24, 52.19, 42.88
Turingite Melting Tests: 305.61, 308.99, 301.22
Turingite Breaking Tests: 64.08, 65.9, 61.00

```

9- The testing team has realized that the results for the Turingite melting point tests were gathered in Fahrenheit rather than Celsius and are invalid. Call the **reset()** function to remove all of the values from this object.

10- Your project manager has asked you to expand your Material class. They now want to know if any of the tested materials will break under a load. Add a public method to the Materials class which will return **true** if *any value* in the **breaking** array is less than the value given as a parameter. Add this line to the Materials.h file:

```

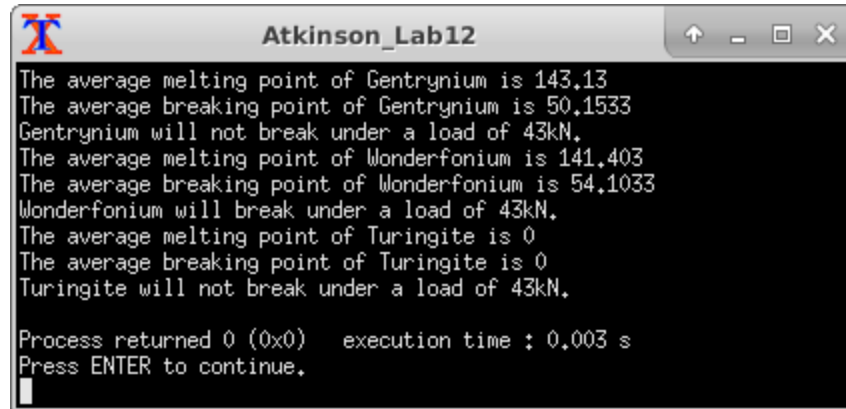
bool willBreak(float);

```

Now implement the new method in the Materials.cpp file. If a **breaking** array is empty this function should return **false**.

11- Use cout statements to print the melting and breaking points for all 3 materials, neatly formatted. Also say if each material will break under a load of **43 kN**. Turingite should have 0 and 0 for its averages if your reset method is working properly.

12- Build and run your code. Correct all error. Your output should look something like this:



```
Atkinson_Lab12
The average melting point of Gentrynium is 143.13
The average breaking point of Gentrynium is 50.1533
Gentrynium will not break under a load of 43kN.
The average melting point of Wonderfonium is 141.403
The average breaking point of Wonderfonium is 54.1033
Wonderfonium will break under a load of 43kN.
The average melting point of Turingite is 0
The average breaking point of Turingite is 0
Turingite will not break under a load of 43kN.

Process returned 0 (0x0)   execution time : 0.003 s
Press ENTER to continue.
```

13- Submit your three files: Material.h, Material.cpp, and main.cpp through TRACS. You can leave when you're done.