1.

Passing a vector by value to a function… (careful about mixing Java/C# syntax).

If you pass an object by value to a function, its copy is created and that copy is actually passed to the function. If this object is a vector, then you could potentially be paying for creating a copy of a big object, assuming the vector has many elements in it (or each element is big). So a vector (or any object) should be passed by const reference if you don't want to modify it, or by reference if your intent is to modify it. In case of built in primitive types (int, double, etc) you can simply pass by value instead of a const reference since these types are small and its no more expensive to pass them by value as compared to passing by reference.

However, don't fall into the trap where you say that my class is real small, it only contains an integer field, so I am ok passing it by value… before you know it, your class fill grow (feature creep ☺) and you will then end up making all those pass by values to pass by references.

```
void Foo( vector <int> vi );         // This is expensive
void Foo( vector <int> & vi );       // This is fine if you want to modify vi
void Foo( const vector <int> & vi );// Don't plan to modify vi inside Foo
```

2.

See note on Return value optimization

If you return a big object by value, that also creates a copy. Instead you can pass in the object by reference and modify it there.

```
vector <int> Foo() {

      vector <int> vi;

      // fill up vi

      return vi;  // A copy of this will be created since its return by value

}
```

Instead, do something like:

```
void Foo(vector <int> & vi) {

      // fill up vi directly, no copy of this vector is created.

}
```

And if you want to know if the function Foo worked successfully, you can return a bool to indicate status, or throw an exception if there was an error.

Or another scenario that I saw:

```
std::string ReadFileIntoString( const std::string & fileName )
{
      std::string str;
      // Code to read the file into string
      return str;
}
```

This returns the sting by value, and create a copy of the string, and in this case, a copy of potentially a big string as it holds the file contents.
Consider passing in the string as a reference (not const) to the function.
So, something like:

```
string & ReadFileIntoString( const string & fileName, string & fileContents )
```

Or

```
void ReadFileIntoString( const string & fileName, string & fileContents )
```

NOTE on Return value optimization:
There is a feature called Return value optimization that will solve this problem for you, i.e., if you do return an object by value, the compiler makes it such that redundant copies are not made. So, if your compiler supports this (most of them really should), then you don't need to pass in the return value by reference.

3.

I saw a **lot** of assignments passing in string by value to constructors, or SetName kind of methods. I have the same comments here as for passing a vector by value I mentioned above.

4.

I saw that sometimes, the return value of a GetStudentVector () like method was correct (it was a const reference), but the problem was the code where this value was being received:

Example:

GetStudentCollection is declared as follows:

```
const vector<Student> &  GetStudentCollection() const
{
      return myStudentVector;
}
```

But the code that receives that value causes the problem:

```
vector<Student>  vecStudent = myClass.GetStudentCollection();
```

Note that the vecStudent is a **copy** of the vector returned from inside `GetStudentCollection`. So, even though the `GetStudentCollection` method was defined correctly, its usage caused the problem.

5.

You should make getName() and getID() kind of methods (also called getters) const methods since they do not modify the object they are called on.

6.

To better organize your code, you should separate parsing and processing of the data.

7.

A lot of assignments used some math to go from upper to lower case (or vice versa). Use the tolower or toupper macros instead.

8.

With any solution that you write (or any code that you write), you should be very sensitive to the algorithmic complexity of your code. In other words, if there is an N square loop, you should question if you can do that in a simpler fashion. If your algorithm is N square, then your processing will be very slow for large values of N.

I saw some N square loops in this assignment, where you can get away with linear (ie, order N) complexity.

9.

```
std::string whiteSpace = " \t\r\n";
```

I saw cases such as the one above, where a string is defined only for search purposes … in other words, this string is not expected to be modified. In this case, you should think of doing the following:

a. Make the string a const (So your code does not accidentally modify it, and if at all there is some optimization that the compiler can do for a const object, you would get that benefit).
b. Define it as a static (So you don't pay the price for its construction every time).

c. And since it is a const, you don't have to worry about multiple threads changing the value of the same object (since we said it should be a static).
d. Lastly, give it a name based on some convention that indicates it is a const.

Given above, you can think of doing something like:

```
static const std::string kWhiteSpace = " \t\r\n";
```

10.

```
void WriteToDisk(const std::string & input)
{
	int filesize = input.size();

	char * buffer = new char[filesize];

	for(int i = 0; i < filesize; i++)
	{
		buffer[i] = input[i];
	}

	std::ofstream fout("FileName.txt", std::ios::binary);

	fout.write(buffer, filesize);

	delete [] buffer;
}
```

I would consider changing the following:

a. Don't allocate buffer, instead use the `c_str()` member of string class.
b. Check for successful open of file for writing.
    a. Certain conditions can lead to the file not being opened for write. These can be (list below is not exhaustive):
        i. No write permission.
        ii. Out of disk space.
c. Check for any errors from the write call.
d. Change the return value to bool to indicate write success (or use exceptions).
e. Last statement could look like:
    `fout.write( input.c_str(), input.size() );`