CS 3005: Programming in C++

`void PPM::writeStream(std::ostream& os) const`

The `writeStream` method is used to send the PPM data to the output stream `os`. Since PPM file is a mixture of ASCII data for the header information, and binary data for the pixel information, we need to use a mixture of stream operations.

The `<<` Operator's Functionality

The `<<` (output operator or stream insertion operator) is used to send data to an output stream, possibly transforming the data for the stream. For example, to send a nice message to a stream you might write:

```
os << "Hello, friend.  How about a nice game of chess?";
```

The `<<` operator has two operands. The left-hand side must be an object whose type is `std::ostream` (or a type that inherits from `std::ostream`). The right-hand side is a variable (such as `x` or `y`) or a literal value (such as `3` or `"hello"`).

The operator's responsibility is to send the data of the right-hand side into the ostream of the left-hand side, using methods or functions that work on the ostream.

C++ comes with many versions of `<<` to send data of various types to an ostream. See the reference.

The type of the right-hand side value is important for the compiler to select the correct operator implementation. If the type is a built-in type (such as `int`) or a standard library type (such as `std::string`), then the compiler will correctly find one of the built-in `<<` implementation that matches the type. This implementation will convert the data to ASCII characters and send the characters to the ostream.

The `ostream::write()` Method's Functionality

If we want to send raw data (binary data as bytes) to the stream, instead of converting the data to ASCII first, we use the `.write()` method of the ostream class. In this case, we tell the write method where the data is located in memory, and how many bytes we want to have written to the stream. Here's an example to write the contents of an `int` variable to an ostream, `os`:

```
int x = 314159;
os.write((char *) &x, sizeof(x));
```

The `&x` is an expression to calculate the location (memory address) of the variable x. The `(char *)` is necessary to tell the `write()` function to treat the address as the location of a collection of bytes (the built-in `char` type is a single byte). The `sizeof()` operator will calculate how many bytes are required by its argument.

Writing a PPM File

The format of a PPM file is

```
P6 WIDTH HEIGHT MAX_COLOR_VALUE\n
BINARY REPRESENTATION OF COLORS FOR EACH PIXEL IN THE SAME ORDER AS THE COLOR FILE
```

where the meta data `P6 WIDTH HEIGHT MAX_COLOR_VALUE\n` is (ASCII)[https://en.wikipedia.org/wiki/ASCII] encoded, and the rest of the data is binary data.

So, we will use the `<<` operator to literally send `"P6"` to `os`. We'll follow this with a literal space `" "` as well. Next, we need the PPM's width and height sent to the stream as ASCII. So, again, the `<<` will be used to send the values. Don't forget to add spaces between them. The maximum color value needs the same treatment.

After the maximum color value send a literal newline character. This is a departure from using `std::endl`. Use `"\n"` here. Exactly one, with no trailing spaces sent to the stream.

Be sure that you put a space between each of the values in the meta data. Otherwise, the file will not be correctly formatted. Think of the difference between these two lines:

```
P6 200 150 255\n
```

```
P6200150255\n
```

This concludes the ASCII header of the file. The rest of the work will send the binary pixel data using `.write()`.

We use 1 byte per channel (that's one byte for red, one for green, and one for blue). We do not put any spaces or other data between these values. Spacing and separation are not an issue, because each pixel uses exactly these 3 bytes, so we know how to use the data.

To send binary data, we use the `write` method of the `ostream`.

Here is an example of writing a single byte:

```
unsigned char c = 17;
os.write((char *) &c, sizeof(c));
```

Of course, you don't want to write the number 17. You want to write the value of a channel. In fact you want to write the value of all channels. Remember the `getChannel()` method of the `PPM` class. You probably want to use 3 nested `for` loops to work over all rows, columns and channels.