| Lab | Manipulator | Name | Asa Hayes |
|---|---|---|---|
| | | OS | Windows 7 |
| | | Compiler | build.tamu.edu |

| | | |
|---|---|---|
| ***Fill in the blank***. *Please select the best response to each of the following questions. Be sure to* ***run all code on build.tamu.edu****; the visual studio compiler will provide different results (read wrong for the purpose of this assignment) for some of the questions* | | |
| **1** | A manipulator is an item designed to be used with the insertion operator `<<` or extraction operator `>>` to adjust the way that output appears. | **True** |
| **2** | Recall that a hexadecimal value can be provided to the output stream using the prefix 0x; likewise, an octal number can be provided using a leading 0. The stream will then output these numbers according to the base specified for integer output in the stream (default is decimal).<br><br>Provided the following code, determine its output:<br><br>`cout << 1234 << ' ' << 0x4d2 << ' ' << 02322 << '\n';` | **1234 1234 1234** |
| **3** | With respect to integer output manipulations, how does `oct` manipulate the output of an integer? | **uses base-8 (octal) notation** |
| **4** | With respect to integer output manipulations, how does `showbase` manipulate the output of an integer? | **prefix 0 for octal and ox for hexadecimal** |
| **5** | By default, a floating-point value is print using six digits using the general/`defaultfloat` format. | **True** |
| **6** | When a floating-point value is output using the general/`defaultfloat` format, the number is rounded to give the best approximation that can be printed using only six digits. | **True** |
| **7** | When a floating-point value is print using the general/`defaultfloat` format, the more appropriate format of scientific or fixed is used to present the most accurate representation of the it. | **True** |
| **8** | When using the manipulator `setprecision(x)` to set the floating-point precision, `x` is the total number of digits when the general format is used | **True** |
| **9** | When using the manipulator `setprecision(x)` to set the floating-point precision, `x` is the total number of digits after the decimal point when the scientific format is used | **True** |
| **10** | When using the manipulator `setprecision(x)` to set the floating-point precision, `x` is the total number of digits when the fixed format is used | **False** |

| | | | |
|---|---|---|---|
| | **11** | For floating-point values, this format uses fixed-point notation. | **fixed** |
| | **12** | For floating-point values, this format uses the mantissa and exponent notation. | **scientific** |
| | **13** | For floating-point values, this format chooses between `fixed` or `scientific` to give the numerically most accurate representation. | **general/defaultfloat** |
| | **14** | Field sizes established by `setw()` stick/persist. | **False** |
| | **15** | What is the output of the following code block?<br>*Feel free to compile and run to see what you get.*<br><br>`#include <iostream>`<br>`#include <ios>`<br>`#include <iomanip>`<br>`using namespace std;`<br><br>`int main() {`<br>`  double fvalue = 198.9987;`<br>`  cout << fixed << fvalue << " " << fvalue;`<br>`}` | **198.998700 198.998700** |
| | **16** | The `fixed` manipulator is a 'sticky' manipulator. | **True** |
| | **17** | The `setw()` manipulator provides a mechanism for setting the width that an integer value or string takes up on output.<br><br>If the value or string is larger than the field size specified, the output is clipped to fit within the field. | **False** |
| | ***Short Answers***. *Please respond to the following questions.* | | |

**18** What is the output of the following code? What does this code say about the `left` and `right` text manipulators? Is the same behavior exhibited by the `setw(n)` manipulator?

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

int main() {
    cout << setfill('*') << setw(21) << "" << setfill(' ') << endl;
    cout << setw(10) << right << "Name" << "|";
    cout << setw(10) << "Age" << endl;
    cout << setw(10) << left << "John" << "|" << setw(10) << 10 << endl;
    cout << setw(10) << "Smith" << "|" << setw(10) << 10 << endl;
    cout << setfill('*') << setw(21) << "" << endl;
}
```

```
*********************
      Name|          Age
John      |10
Smith     |10
*********************
```

The left and right manipulators change the starting point and orientation of the regular placement of the output elements. The behavior is the same as setw(), except from a specific direction instead of centered.

**19** What happens when the length of the string to be inserted exceeds that of the field specified by `setw(n)`? Why is this? Your response must include the code that you used to test this.

```
int main() {
   cout << setfill('*') << setw(21) << "" << setfill(' ') << endl;
   cout << setw(10) << right << "Name" << "|";
   cout << setw(10) << "Age" << endl;
   cout << setw(10) << left << "John" << "|" << setw(10) << 10 << endl;
   cout << setw(10) << "SmithSmithSmith" << "|" << setw(10) << 10 << endl;
   cout << setfill('*') << setw(21) << "" << endl;
}
```

When the string to be output is larger than the field size, the string is preserved at the cost of disrupting the ordering of the set.

**20** Run the following code on your machine, paying careful attention to when the output gets printed to the screen.

What do you observe?

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

int main(){
    cout << "starting exec... ";
    for(int i=0; i<std::numeric_limits<int>::max(); i++);
    cout << "done!";
}
```

The output of 'done!' does not happen for several seconds after running, about 10-11s. This is due to the contained loop running to the end of the numeric limit of an int.

**21** Run the following code on your machine, paying careful attention to when the output gets printed to the screen.

What differences do you notice between this block and that presented in question 20? Why do they occur?

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

int main(){
    cout << "starting exec... " << flush;
    for(int i=0; i<std::numeric_limits<int>::max(); i++);
    cout << "done!";
}
```

The output of 'done!' happens at about the same time compared to the last one, 10s. Because this didn't seem right, I researched what was supposed to have happened, and it should have waited until the loop was over and printed both statements at once. I was using the build.tamu.edu compiler, so I'm not sure why it did not.

| | | | | |
|---|---|---|---|---|
| | **21** | Save as a PDF and submit on Gradescope. | | |