Basic in Output Formatting with C++ Stream method- cout/cin

One big advantage is of this over C-I/O is "**type-safe**".

Proactive Learning:

- Stay inquisitive and experiment.
- Don't just read it.
- Type in samples, compile and run.
- Then, experiment with a few different things to display.

Cover:

Bare basic for reading standard I/O

1. Header files

```
#include <iostream>   // std::cin, std::cout
#include <iomanip>    // for set width and special format
using namespace std;  // for standard input and output stream
```

2. Fundamental cin and cout (get user input and display user output)

<< : called *insertion* –

e.g. cout << var : insert information from variable name "var" to the output stream.

>> : called *extraction* -

e.g. cin >> var : read as extract information from input stream and put in variable name "var".

Sample 1:

```
int a,b;
string name;

        cin >> name;
cin >> a;
cin >> b;
```

or ultimately, you can gather them on a same line, like:

```
cin >> a >> b;
```

Sample 2: with output

```
        cout << "Hello, " << name << ", you have entered " << a << "and" << b;
```

Common standard I/O formatting methods

1. I/O manipulator: Field Width – setw(n)

Sample 1:

```
int width = 10;
float amount = 5.5;
cout << setw(width) << amount;
```

or

```
cout << setw(width) << amount << "\n";
```

or

```
cout << setw(width) << amount<< endl;
```

Note: if you do not include "using namespace std", you will need to reference with "std::" , e.g.

```
std::cout << "Please, enter the amount: ";
```

Sample 2 - with loop:

```
const int max = 12;
const int width = 6;
for(int row = 1;row <= max;row++) {
        for(int col = 1;col <= max;col++) {
                cout << setw(width) << row * col;
        }
        cout << endl;
}
```

Notice how "setw(n)" controls the field width, so each number is printed inside a field that stays the same width regardless of the width of the number itself.

However, setw(n)" is volatile. *"setw(n)" only works for a single subsequent variable.* You must apply "setw(n)" for each variable.

2. I/O manipulator: Justification in Field

Choices are left and right. Numbers are by default right-justified.

```
setiosflags(ios::left);
setiosflags(ios::right);
```

Sample :

```
cout << setiosflags(ios::left) << 5.5;
cout << resetiosflags(ios::right)<< 5.5;
// need to reset in order to return to default, i.e right-justified.
```

3. I/O manipulator: Controlling Precision

There are two IO stream base :

- Fixed : write floating-point values in fixed-point notation
- Scientific : write floating-point values in scientific notation.

Sample 1:

```
double x = 800000.0/81.0;
setprecision(5);
cout << setiosflags(ios::fixed) << x;
```

then change the format to scientific notation:

```
cout << setiosflags(ios::scientific) << x;
```

Sample 2:

```
double a = 3.1415926534;
double b = 1996.1;
double c = 5.2e-10;

cout.precision(5);

cout << "default:\n";
cout << a << '\n' << b << '\n' << c << '\n';

cout << '\n';

cout << "fixed:\n" << fixed;
cout << a << '\n' << b << '\n' << c << '\n';

cout << '\n';

cout << "scientific:\n" << scientific;
cout << a << '\n' << b << '\n' << c << '\n';
```

Again, compile and run. Pay attention to the display and how each is made differently. Then, experiment with variations.

4. I/O manipulator: Choose a character to fill the leading spaces in a number

Sample :

```
int m = 1, d = 2, y = 2015;
cout << setfill('0')
cout << setw(2) << m << '/' << d << '/' << y << endl;
```

Compile and run. Then, try the one below:

```
cout << setfill('0');
cout << setw(2) << m << '/'
<< setw(2) << d << '/'
```

```
        << setw(4) << y << endl;
```

5. Number Bases other than 10

For base 8 and base 16:

```
        unsigned long x = 64206;
        cout << x
        << " in base 8 is \"" << oct << x << "\""
        << " and in base 16 is \"" << hex << x << "\"" << endl;
```

This stream formatting for different bases also works for input:

```
int x;
cin >> hex >> x;
```

Skip the following example if you are a beginner:

```
string convBase(unsigned long v, long base)
{
        string digits = "0123456789abcdef";
        string result;
        if((base < 2) || (base > 16)) {
                result = "Error: base out of range.";
        }
        else {
                do {
                        result = digits[v % base] + result;
                        v /= base;
                }
                while(v);
        }
        return result;
}

int main()
{
        unsigned long x = 64206;
        cout << "Hex:     " << convBase(x,16) << endl;
        cout << "Decimal: " << convBase(x,10) << endl;
        cout << "Octal:   " << convBase(x,8) << endl;
        cout << "Binary:  " << convBase(x,2) << endl;
        cout << "Test:    " << convBase(x,32) << endl;
        return 0;
}
```

6. IO Manipulators Summary <iomanip>

Header providing parametric manipulators:

setiosflags	setbase	setprecision
resetiosflags	setfill	setw

get_money get_time

put_money put_time

File IO

NOTE: The scope of the following simply shows very brief description for file I/O to allow you to start reading and writing to a text file, not meant to teach you how exactly the whole FILE- I/O mechanism works.

Simple Open – Read - Write

Basic steps:

1. Open the file.
2. Do all the reading or writing.
3. Close the file.

Open a file :

```
ifstream infp;   //  create input object
ofstream outfp;  //  create output object

infp.open(fname, ios::in);   // open a file for input mode
outfp.open(fname, ios::out);  // open a file for output mode
```

Sample:

```
ifstream infp;
infp.open("inputFilename.txt", ios::in);

if (!infp) {
  cerr << "Can't open input file " << "inputFilename.txt" << endl;
  exit(1);
}

outfp.open("outputFilename.tx", ios::out);

if (!outfp) {
  cerr << "Can't open output file " << "outputFilename.txt" << endl;
  exit(1);
}
```

Reading a file:

e.g. Data in inputfilename.txt shows:

```
amy 100
joey 50
```

To get the data and store in memory:

```
char name[10]; int score;

while (!infp.eof()) {
  infp >> name >> score;
```

```
  outfp << name << " " << score << endl;
}

Or

while (infp >> username >> score) {
  …
}
```

Close a file:

```
infp.close();
outfp.close();
```

Redirect I/O

First, you should know available access mode:

In C:

```
fopen (filename, mode);
```

mode	
"r"	read: Open file for input operations. The file must exist.
"w"	write: Create an empty file for output operations. If a file with the same name already exists, its contents are discarded and the file is treated as a new empty file.
"a"	append: Open file for output at the end of a file. Output operations always write data at the end of the file, expanding it. Repositioning operations (fseek, fsetpos, rewind) are ignored. The file is created if it does not exist.
"r+"	read/update: Open a file for update (both for input and output). The file must exist.
"w+"	write/update: Create an empty file and open it for update (both for input and output). If a file with the same name already exists its contents are discarded and the file is treated as a new empty file.
"a+"	append/update: Open a file for update (both for input and output) with all output operations writing data at the end of the file. Repositioning operations (fseek, fsetpos, rewind) affects the next input operations, but output operations move the position back to the end of file. The file is created if it does not exist.

To open a file as a *binary file*:

A `"b"` character has to be included in the *mode* string, e.g.:

"rb", "wb", "ab", "r+b", "w+b", "a+b") or

Additional specifier in C++ (`"x"`), that can be appended to any `"w"` specifier (to form `"wx"`, `"wbx"`, `"w+x"` or `"w+bx"/"wb+x"`). This subspecifier forces the function to fail if the file exists, instead of overwriting it.

In C++:

```
open (filename, mode);
```

Where `filename` is a string representing the name of the file to be opened, and `mode` is an optional parameter with a combination of the following flags:

mode	
`ios::in`	Open for input operations.
`ios::out`	Open for output operations.
`ios::binary`	Open in binary mode.
`ios::ate`	Set the initial position at the end of the file. If this flag is not set, the initial position is the beginning of the file.
`ios::app`	All output operations are performed at the end of the file, appending the content to the current content of the file.
`ios::trunc`	If the file is opened for output operations and it already existed, its previous content is deleted and replaced by the new one.

e.g.:
```
ofstream myfile;
myfile.open ("example.bin", ios::out | ios::app | ios::binary);
```

To Redirect stdout to a file: e.g.

```
freopen ("myfile.txt","w",stdout);
```

To redirect file data into a string: e.g.

```
string line;

ifstream infp;
infp.open(fname, "r");

while (infp >> line)
{
 …
}
```

To redirect stdout to a file: e.g.

Note: method rdbuf() returns a pointer to the *stream buffer* object currently associated with the stream, such as cout, cin, etc.

```
streambuf  *psbuf, *backup;
ofstream  ofp;

ofp.open( "outfilename.txt");
backup = cout.rdbuf();          // backup cout's stream buffer;
psbuf = ofp.rdbuf();            // get file's internal stream buffer
cout.rdbuf(psbuf);              // assign stream buffer to cout
cout << "this will be written to the file instead of stdout";
cout.rdbuf(backup);             // restore
```

```
ofp.close();
```

another sample segment for writing multiple values:

```
for (i=0; i < 10; i++)
        cout << i << ",\n" ;
```

To redirect variable values to a file: e.g.

```
int i;
char s;
ifstream infp;

infp.open("outfilename.txt");

while (infp >> i >> s )
{
     cout << i << endl;
}
```

To redirect standard input/output to a string stream:

Std::stringstream : Stream class to operate on strings.

```
using namespace std

stringstream buffer;
streambuf *old = cout.rdbuf(buffer.rdbuf());

cout << "go to the string buffer instead" <<  endl;

string text = buffer.str();
      // text will now contain "go to the string buffer instead\n"
```

Misc.

get_time (skip this if you are beginner)

You will need to add in #include <ctime> . You should review the *struct tm* data type before using this manipulator.

Sample 1:

```
  struct std::tm when;
  std::cout << "Please, enter the time: ";
  std::cin >> std::get_time(&when,"%R");   // extract time (24H format)

  if (std::cin.fail())
        std::cout << "Error reading time\n";
  else
  {
    std::cout << "The time entered is: ";
    std::cout << when.tm_hour << " hours and " << when.tm_min << "
minutes\n";
  }
```

put_time (skip this if you are beginner)

You need to know which format to use in order to print out time in a certain format. There are of these. You may view all online. Here is a sample online site:
http://www.cplusplus.com/reference/iomanip/put_time/

Sample 1:

```
        #include <chrono>       // std::chrono::system_clock

        using std::chrono::system_clock;
  std::time_t tt = system_clock::to_time_t (system_clock::now());

  struct std::tm * ptm = std::localtime(&tt);
  std::cout << "Now (local time): " << std::put_time(ptm,"%c") << '\n'
```