2 Tools for Stream I/O

- To control the format of the program's output
  - We use commands that determine such details as:
    - The spaces between items
    - The number of digits after a decimal point
    - The numeric style: scientific notation for fixed point
    - Showing digits after a decimal point even if they are zeroes
    - Showing plus signs in front of positive numbers
    - Left or right justifying numbers in a given space

2.1 Formatting Output to Files

- Format output to the screen with:

```
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
```

- Format output to a file using the out-file stream named out_stream with:

```
out_stream.setf(ios::fixed);
out_stream.setf(ios::showpoint);
out_stream.precision(2);
```

2.1.1 out_stream.precision(2);

- precision is a member function of output streams
  - After **out_stream.precision(2);**
    Output of numbers with decimal points...
    - will show a total of 2 significant digits 23. 2.2e7 2.2 6.9e-1 0.00069
      OR
    - will show 2 digits after the decimal point 23.56 2.26e7 2.21 0.69 0.69e-4
      - In both the fixed and scientific notations, the precision field specifies exactly how many digits to display after the decimal point.
- Calls to precision apply only to the stream named in the call

2.1.2 setf(ios::fixed);

- **setf** is a member function of output streams
  - **setf** is an abbreviation for set flags
    - A flag is an instruction to do one of two options
    - **ios::fixed** is a flag
  - After **out_stream.setf(ios::fixed);**
    All further output of floating point numbers...
    - Will be written in fixed-point notation, the way we normally expect to see numbers
- Calls to **setf** apply only to the stream named in the call

2.1.3 setf(ios::showpoint);

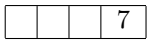
- After **out_stream.setf(ios::showpoint);**
  Output of floating point numbers...
  - Will show the decimal point even if all digits after the decimal point are zeroes

2.2 Creating Space in Output

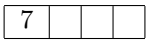
- The width function specifies the number of spaces for the next item
  - Applies only to the next item of output
- Example: To print the digit 7 in four spaces use

```
out_stream.width(4);
out_stream << 7 << endl;
```

  - Three of the spaces will be blank



(ios::right)



(ios::left)

2.2.1 Not Enough Width?

- What if the argument for width is too small?
  - Such as specifying

```
cout.width(3);
```

    when the value to print is 3456.45
- The entire item is always output
  - If too few spaces are specified, as many more spaces as needed are used

2.3 Unsetting Flags

- Any flag that is set, may be unset
- Use the **unsetf** function
  - Example:

```
cout.unsetf(ios::showpos);
```

    causes the program to stop printing plus signs on positive numbers

2.4 Manipulators

- A manipulator is a function called in a nontraditional way
- Manipulators in turn call member functions
  - Manipulators may or may not have arguments
  - Used after the insertion operator (<<) as if the manipulator function call is an output item

2.4.1 The setw Manipulator

- **setw** does the same task as the member function width
  - **setw** calls the width function to set spaces for output
- Example:

```
cout << "Start" << setw(4) << 10
        << setw(4) << setw(6) << 30;
```

produces:

```
Start   10   20     30
```

2.4.2 The setprecision Manipulator

- **setprecision** does the same task as the member function precision
- Example:

```
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout << "$" << setprecision(2) << 10.3 << endl
        << "$" << 20.5 << endl;
```

produces:

```
$10.30
$20.50
```

  - **setprecision** setting stays in effect until changed

2.4.3 Manipulator Definitions

- The manipulators **setw** and **setprecision** are defined in the **iomanip** library
- To use these manipulators, add these lines

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

2.4.4 sample15.cpp

```cpp
#include <iostream>
#include <iomanip>
int main (int argc, char *argv[]) {
    double number = 123.456;
    // print number in default
    std::cout << "number=" << number << std::endl;
    // set precision 2
    std::cout.precision(2);
    std::cout << "number=" << number << std::endl;
    // set fixed decimal point and alway show the point
    std::cout.setf(std::ios::fixed);
    std::cout.setf(std::ios::showpoint);
    std::cout.precision(2);
    std::cout << "number=" << number << std::endl;

    // use setprecision from "iomanip"
    std::cout << std::setprecision(1);
    std::cout << "number=" << number << std::endl;

    // set width
    std::cout.width(10);
    std::cout << number << std::endl;

    // set width left
    std::cout.setf(std::ios::left);
    std::cout.width(10);
    std::cout << number << number <<std::endl;

    // use setw from "iomanip"
    std::cout.unsetf(std::ios::left);
    std::cout << number << std::setw(10) << number <<std::endl;
    return 0;
}
```

```
number=123.456
number=1.2e+02
number=123.46
number=123.5
     123.5
123.5      123.5
123.5      123.5
Program ended with exit code: 0
```

2.5 The End of The File

- Input files used by a program may vary in length
  - Programs may not be able to assume the number of items in the file
- A way to know the end of the file is reached:
  - The boolean expression (in_stream >> next)
    - Reads a value from in_stream and stores it in next
    - True if a value can be read and stored in next
    - False if there is not a value to be read (the end of the file)

2.5.1 End of File Example

- To calculate the average of the numbers in a file

```
double next, sum = 0;
int count = 0;
while(in_stream >> next)
{
        sum = sum + next;
        count++;
}
double average = sum / count;
```

2.5.2 Using eof

- To calculate the average of the numbers in a file

```
double next, sum = 0;
int count = 0;
while (1)
{
        in_stream >> next;
        if (in_stream.eof()) break;
        sum = sum + next;
        count++;
}
```

- **in_stream.eof( )** becomes **true** when the program reads **past** the last line in the file