# 8.1 Output and input streams

## The ostream and cout streams



Programs need a way to output data to a screen, file, or elsewhere. An ***ostream***, short for "output stream," is a class that supports output, available via `#include <iostream>` and in namespace "std". ostream provides the ***<< operator***, known as the ***insertion operator***, for converting different types of data into a sequence of characters. That sequence is normally placed into a buffer, and the system then outputs the buffer at various times. The << operator returns a reference to the ostream that called the operator, and is evaluated from left to right like most operators, so << operators can appear in series.

The << operator is overloaded with functions to support the various standard data types, such as int, bool, float, etc., each function converting that data type to a sequence of characters. The operator may be further overloaded by the string library from `#include <string>` or by the programmer for programmer-created classes.

***cout*** is a predefined ostream object (declared as `ostream cout;` in the iostream library) that is pre-associated with a system's standard output, usually a computer screen.

Basic use of cout and the insertion operator were covered in an earlier section.

**PARTICIPATION ACTIVITY** 8.1.1: ostream supports output.

### Animation captions:

1. The insertion operator converts the string literal to characters, temporarily storing characters in an output buffer.
2. The system then writes the buffer's content to screen.

**PARTICIPATION ACTIVITY** 8.1.2: ostream and cout.



1) Characters written to cout are immediately written to a system's standard output.
   - True
   - False

2) To use cout, a program must include the statement `#include`

`<iostream>`.

- ○ True
- ○ False

3) << is known as the stream operator.

- ○ True
- ○ False



## The istream and cin streams

Programs need to receive input data, whether from a keyboard, touchscreen, or elsewhere. An ***istream***, short for "input stream," is a class that supports input. Available via `#include <iostream>`, istream provides the ***>> operator***, known as the ***extraction operator***, to extract data from a data buffer and write the data into different types of variables.

***cin*** is a predefined istream pre-associated with a system's standard input, usually a computer keyboard. The system automatically puts the standard input into a data buffer associated with cin. The >> operator skips leading whitespace and extracts as many characters as possible consistent with the target variable's type. The operator then stops at the next whitespace, converts the extracted characters to the target variable's type, and stores the result into the variable.

Basic use of cin and the extraction operator were covered in an earlier section.

**PARTICIPATION ACTIVITY** 8.1.3: istream and the extraction operator support input.

### Animation captions:

1. The system puts input characters into a buffer.
2. The extraction operator reads characters up to the next whitespace, converts to the target variable's data type, and stores the results into variable firstName.
3. The extraction operator skips leading whitespaces.

**PARTICIPATION ACTIVITY** 8.1.4: istream and cin.



1) cin is a predefined istream associated with the system's standard input.

- ○ True
- ○ False

2) A read from cin will directly read characters from the system's keyboard.

- True
- False

3) >> is known as the extraction operator.

- True
- False



4) If the istream data buffer currently has " Hi friend!" and firstString is a string variable, then `cin >> firstString` will store " Hi " into firstString.

- True
- False

5) If the istream data buffer currently has "friend!" and secondString is a string variable, then `cin >> secondString` will store "friend" into secondString.

- True
- False

6) If the istream data buffer currently has "Hey" and myChar is a char type, then `cin >> myChar` will store "Hey" into myChar.

- True
- False

7) If the istream data buffer currently has "21JumpStreet" and userVal is an int type, then `cin >> userVal` will store 21 into userVal.

- True
- False

©zyBooks 04/25/21 07:14 488201
xiang zhao
BAYLORCSI14301440Spring2021

Exploring further:

- ostream Reference Page from cplusplus.com
- More on ostreams from msdn.microsoft.com

- istream Reference Page from cplusplus.com
- More on istreams from msdn.microsoft.com

# 8.2 Output formatting



## Floating-point manipulators

A programmer can adjust the way that a program's output appears, a task known as output formatting. The main formatting approach uses manipulators. A ***manipulator*** is a function that overloads the insertion operator << or extraction operator >> to adjust the way output appears. Manipulators are defined in the iomanip and ios libraries in namespace std.

**PARTICIPATION ACTIVITY** 8.2.1: Floating-point manipulators.

### Animation content:

undefined

### Animation captions:

1. The ios and iomanip libraries define several floating-point manipulators.
2. When a floating-point number is output, the default number of digits to display is 6.
3. The setprecision() manipulator changes the total number of digits to display. Scientific notation (e with a power of 10) may result for smaller precision values.
4. setprecision() affects all subsequent floating-point number output, not just the next output.
5. The fixed manipulator uses fixed-point notation, and the precision (2) now applies to the number of decimal places.
6. The scientific manipulator turns scientific notation on.

Table 8.2.1: Floating-point manipulators.



| Manipulator | Description | Example |
|---|---|---|
| fixed | Use fixed-point notation.<br>From <iostream> | `// 12.340000`<br>`cout << fixed << 12.34;` |
| scientific | Use scientific notation. | |

| | | |
|---|---|---|
| | From <iostream> | `// 1.234000e+01`<br>`cout << scientific << 12.34;` |
| setprecision(p) | If stream has not been manipulated to fixed or scientific:<br>Sets max number of digits in number | `// 12.3`<br>`cout << setprecision(3) << 12.34;`<br><br>`// 12.34`<br>`cout << setprecision(5) << 12.34;` |
| | If stream has been manipulated to fixed or scientific:<br>Sets max number of digits in fraction only (after the decimal point).<br>From <iomanip> | `// 12.3`<br>`cout << fixed << setprecision(1) << 12.34;`<br><br>`// 1.2e+01`<br>`cout << scientific << setprecision(1) << 12.34;` |
| showpoint | Even if fraction is 0, show decimal point and trailing 0s.<br>Opposite is noshowpoint.<br>From <iostream> | `// 99`<br>`cout << setprecision(3) << 99.0;`<br><br>`// 99.0`<br>`cout << setprecision(3) << showpoint << 99.0;` |



Manipulators are always meant to be used with the << and >> operators. A common error is to have a statement like `setprecision(2);` rather than `cout << setprecision(2);`, which compiles fine but does not impact cout. Details on operator overloading are presented elsewhere in this material.

**PARTICIPATION ACTIVITY** 8.2.2: Output formatting for floating-point manipulators.

Use the code below to answer each question, and assume no manipulators have previously been applied.

```
double temp;
temp = 98.63;
// a floating-point manipulator
cout << temp;
```



1) Which causes the output to appear as "98.6"?

- `cout << fixed;`
- `cout << setprecision(3);`
- `cout << setprecision(2);`
- `cout << scientific <<`

```
setprecision(2);
```

2) Which causes the output to appear as "9.86e+01"?

- `cout << fixed;`
- `cout << setprecision(3);`
- `cout << setprecision(2);`
- `cout << scientific << setprecision(2);`

3) Which causes the output to appear as "99"?

- `cout << fixed;`
- `cout << setprecision(3);`
- `cout << setprecision(2);`
- `cout << scientific << setprecision(2);`



## Text-alignment manipulators

Some manipulators help align output.

**PARTICIPATION ACTIVITY** 8.2.3: Text-alignment manipulators.

### Animation content:

`undefined`

### Animation captions:

1. The setw() manipulator sets the number of characters for displaying the following item to 10. The left manipulator outputs the following item "Dog age" left-aligned within 10 characters.
2. The vertical bar | is output at the end of the 10 characters. setw() only affects the "Dog age" output.
3. The setw() and right manipulators output "Human age" right-aligned within 12 characters.
4. The setfill() manipulator sets the dash character to fill the empty space created when outputting an empty string in the next 23 characters.
5. setfill(' ') sets the empty space character back to the default space character.
6. The dog age values are output left-aligned within 10 spaces, and the human age values are output right-aligned within 12 spaces.
7. One final line of 23 dashes is output.

## Table 8.2.2: Text-alignment manipulators.

| Manipulator | Description | Example |
|---|---|---|
| setw(n) | Sets the number of characters for the next output item only<br>(does not persist, in contrast to other manipulators).<br>By default, the item will be right-aligned, and filled with spaces.<br>From <iomanip> | <br>`// "  Amy"`<br>`// " George"`<br>`cout << setw(7) << "Amy" << endl;`<br>`cout << setw(7) << "George" << endl;` |
| setfill(c) | Sets the fill to character c.<br>From <iomanip> | `// "****Amy"`<br>`cout << setfill('*') << setw(7) << "Amy";` |
| left | Changes to left alignment.<br>From <iostream> | `// "Amy    "`<br>`cout << left << setw(7) << "Amy";` |
| right | Changes back to right alignment.<br>From <iostream> | `// "    Amy"`<br>`cout << right << setw(7) << "Amy";` |

**PARTICIPATION ACTIVITY** 8.2.4: Output formatting for text manipulators.

Use the code below to answer each question, and assume no manipulators have previously been applied.

```
string str = "Amy";
```

1) Which statement prints "...Amy"?



- `cout << setfill('.') << str;`
- `cout << setw(6) << setfill('.') << str;`
- `cout << setw(6) << "." << str;`
- `cout << right << setw(6)`

```
    << str;
```

2) Which statement prints "Amy"?

- `cout << setfill('.') << str;`
- `cout << setw(6) << setfill('.') << str;`
- `cout << setw(6) << "." << str;`
- `cout << right << setw(6) << str;`



3) Which statement prints"  Amy"?

- `cout << setfill('.') << str;`
- `cout << setw(6) << setfill('.') << str;`
- `cout << setw(6) << "." << str;`
- `cout << right << setw(6) << str;`

## Buffer manipulators

Printing characters from the buffer to the output device (e.g., screen) requires a time-consuming reservation of processor resources. Once the resources are reserved, moving characters is fast, whether there is 1 character or 50 characters to print.

To preserve resources, the system may wait until the output buffer is full, or at least has a certain number of characters, before moving the characters to the output device. Or, with fewer characters in the buffer, the system may wait until the resources are not busy. Sometimes a programmer does not want the system to wait. Ex: In a very processor-intensive program, waiting could cause delayed and/or jittery output.

Two manipulators exist to send all buffer contents to the output device without waiting: endl and flush.



Table 8.2.3: Buffer manipulators.

| Manipulator | Description | Example |
|---|---|---|
| endl | Inserts a newline character '\n' into the output buffer | |

| | | |
|---|---|---|
| | and informs the system to flush the buffer.<br>From <iostream> | `// Insert newline and flush`<br>`cout << endl;` |
| flush | Informs the system to flush the buffer.<br>From <iostream> | `// Flush buffer`<br>`cout << flush;` |



A common error is to assume that a cout statement is never reached because the output had not been flushed when the program crashed. Ex: If the program crashes on the statement `cout << "value is " << someFunction();`, the words "value is" will not output because the buffer was not flushed.

Exploring further:

- More manipulators exist. See cplusplus.com

**PARTICIPATION ACTIVITY** 8.2.5: Buffer manipulators.

1) The text "test" is likely to immediately display on the screen.

```
cout << "test";
```

- True
- False

2) The text "test" and "this" are likely to immediately display on the screen.

```
cout << "test" << endl << "this\n";
```

- True
- False

3) The text "test" is likely to immediately display on the screen.

```
cout << "test" << flush;
```

- True
- False

**CHALLENGE ACTIVITY** 8.2.1: Output formatting.

Start

Type the program's output



```
#include <iostream>
#include <iomanip>

using namespace std;

int main() {
   float myFloat;

   myFloat = 12.3423;

   cout << setprecision(5) << myFloat << endl;
   cout << setprecision(6) << myFloat << endl;

   return 0;
}
```

```
12.342
12.3423
```

1 2 3 4 5

Check Next

**CHALLENGE ACTIVITY** 8.2.2: Output formatting: Printing a maximum number of digits.

Write a single statement that prints outsideTemperature with 4 digits. End with newline.
Sample output with input 103.45632:

```
103.5
```



```
1  #include <iostream>
2  #include <iomanip>
3  using namespace std;
4
5  int main() {
6     double outsideTemperature;
7
8     cin >> outsideTemperature;
9
10    /* Your solution goes here  */
11
12    return 0;
```

```
13 }
```

Run

View your last submission



**CHALLENGE ACTIVITY** 8.2.3: Output formatting: Printing a maximum number of digits in the fraction.

Write a single statement that prints outsideTemperature with 2 digits in the fraction (after the decimal point). End with a newline. Sample output with input 103.45632:

```
103.46
```

```
1  #include <iostream>
2  #include <iomanip>
3  using namespace std;
4
5  int main() {
6     double outsideTemperature;
7
8     cin >> outsideTemperature;
9
10    /* Your solution goes here  */
11
12    return 0;
13 }
```

Run

View your last submission



# 8.3 Input string stream

## Input string stream

Sometimes a programmer wishes to read input data from a string rather than from the keyboard. A new ***input string stream*** variable of type ***istringstream*** can be created that reads input from an associated string instead of the keyboard (standard input). istringstream is derived from istream. An istringstream can be used just like the cin stream as illustrated in the program below.

**PARTICIPATION ACTIVITY** 8.3.1: Reading a string as an input stream.



### Animation content:

undefined

### Animation captions:

1. The program uses #include <sstream> for access to the string stream class, which is in namespace std.
2. istringstream inSS(userInfo); declares a new string stream variable and initializes the string stream's buffer to a copy of string userInfo.
3. Then the program extracts data from stream inSS using >>, similar to extracting from cin.
4. The extracted data is output to the screen.

**PARTICIPATION ACTIVITY** 8.3.2: Input string streams.

1) Declare an istringstream variable named inSS that creates an input string stream using the string variable myStrLine.

Check Show answer

## Using getline() with string streams



A common use of string streams is to process user input line-by-line. The program below reads in a line as a string and then extracts individual data items from the string. The getline() function reads an input line into a string, and `inSS.str(lineString);` uses the str() function to initialize the stream's buffer to string lineString. Afterwards, the program extracts input from inSS using >>. The statement `inSS.clear();` is necessary to reset the state of the stream so that subsequent extractions start from the beginning of the input strings.

## Figure 8.3.1: Using a string stream to process a line of input text.

```
#include <iostream>
#include <string>
#include <sstream>
using namespace std;

int main() {
   istringstream inSS;       // Input string stream
   string lineString;        // Holds line of text
   string firstName;         // First name
   string lastName;          // Last name
   int    userAge;           // Age
   bool   inputDone;         // Flag to indicate
next iteration

   inputDone = false;

   // Prompt user for input
   cout << "Enter \"firstname lastname age\" on each
line" << endl;
   cout << "(\"Exit\" as firstname exits)." << endl
<< endl;

   // Grab data as long as "Exit" is not entered
   while (!inputDone) {

      // Entire line into lineString
      getline(cin, lineString);

      // Copies to inSS's string buffer
      inSS.clear();
      inSS.str(lineString);

      // Now process the line
      inSS >> firstName;

      // Output parsed values
      if (firstName == "Exit") {
         cout << "   Exiting." << endl;

         inputDone = true;
      }
      else {
         inSS >> lastName;
         inSS >> userAge;

         cout << "   First name: " << firstName <<
endl;
         cout << "   Last  name: " << lastName <<
endl;
         cout << "   Age:        " << userAge   <<
endl;
         cout << endl;
      }
   }

   return 0;
}
```

```
Enter "firstname lastname age" on
each line
("Exit" as firstname exits).

Mary Jones 22
   First name: Mary
   Last  name: Jones
   Age:        22

Sally Smith 14
   First name: Sally
   Last  name: Smith
   Age:        14

Exit
   Exiting.
```



©zyBooks 04/25/21 07:14 488201
xiang zhao
BAYLORCSI14301440Spring2021

**PARTICIPATION ACTIVITY** 8.3.3: Using getline() with string streams.

1) Which function is used to read an entire string from user input?
   - getline()
   - str()
   - clear()

2) Which function resets the string stream's state?
   - getline()
   - str()
   - clear()

3) Which function copies the input string into the string stream?
   - getline()
   - str()
   - clear()



## Reaching the end of a string stream

A programmer will not always know how much data exists in a user input string, so using multiple individual extractions (Ex: `inSS >> data1; inSS >> data2;`) is not useful for reading all of the input data. Input streams have a Boolean function called ***eof()*** or ***end of file*** that returns true or false depending on whether or not the end of the stream has been reached. An if statement or while loop can check if the end of input string stream has been reached by using the extraction operator. Ex: In the code

```
while (inSS >> data) {
   ...
}
```

the while statement implicitly calls inSS's eof() function, which returns false if more data exists in the string stream to be read and true if the end of string stream has been reached. When eof() returns false, `inSS >> data` resolves to true, causing the loop to continue. Conversely, when eof() returns true, `inSS >> data` resolves to false, meaning the end of the string stream has been reached, and the loop exits.



**PARTICIPATION ACTIVITY** 8.3.4: Reaching the end of a string stream.

### Animation content:

undefined

## Animation captions:

1. The user enters a single string, which contains any number of names.
2. The extraction in the while loop condition implicitly calls inSS's eof() function, which indicates if data exists in the string to be read or if the end of the string has been reached.
3. The program continues to loop and output each name.
4. When the end of the string is reached, the loop condition resolves to false and the loop exits.



**PARTICIPATION ACTIVITY** 8.3.5: Reaching the end of a string stream.

1) In the animation above, how many times does the while loop iterate?
   - 3
   - 4
   - 5

2) What indicates to a program that the end of an input string stream has been reached?
   - The string stream's eof() function
   - The extraction operator (>>)
   - getline()

3) If in the loop condition inSS >> data, inSS's eof() function returns true, what does the condition inSS >> data resolve to?
   - true
   - false

## Example: Phone number formats



The example below reads in phone numbers in two different formats and then presents the phone numbers in a standard 10-digit format. American phone numbers include an area code (3 digits), central office code (3 digits), and station number (4 digits). The program uses an input string stream to read each portion of the phone number, the ios good() function to determine if the area code entered consists solely of integers, and two dummy characters to read and determine if valid phone number delimiters were used. If the area code does not contain solely integers or the delimiters are not valid, the string stream is placed into an error state, and an error message is output. The program outputs all valid phone numbers in a standardized format.

## Figure 8.3.2: Input stream example: Phone number formats.

```
#include <iostream>
#include <sstream>
#include <string>
using namespace std;

int main() {
   istringstream inSS;       // Input string stream
   string lineString;        // Holds line of text
   int    areaCode;          // Area code (3 digits)
   int    officeCode;        // Central office code (3
digits)
   int    stationNum;        // Station number (4
digits)
   char   dummyChar1;
   char   dummyChar2;
   bool   isValidNumber;

   cout << "Enter a 10-digit phone number (or -1 to
exit):" << endl;
   getline(cin, lineString);

   while (lineString != "-1") {
      isValidNumber = false; // Set to false before
extracting phone number
      dummyChar1 = ' '; // Reset dummy chars to
character other than - and (
      dummyChar2 = ' ';

      // Copy input to inSS's string buffer
      inSS.clear();
      inSS.str(lineString);

      // Try extracting area code.
      inSS >> areaCode;
      if (inSS.good()) {
         // Number format should be ###-###-####
         inSS >> dummyChar1 >> officeCode >>
dummyChar2 >> stationNum;

         if (inSS.eof() && dummyChar1 == '-' &&
dummyChar2 == '-') {
            isValidNumber = true;
         }
      }
      else {
         // Number format should be (###) ###-####

         // Clear inSS state, and try extracting with
area code in ()
         inSS.clear();
         inSS >> dummyChar1 >> areaCode >> dummyChar2;
         if (inSS.good()  && dummyChar1 == '(' &&
dummyChar2 == ')') {
            // Extract office code, then -, and then
station number
            inSS >> officeCode >> dummyChar1 >>
stationNum;
            if (inSS.eof() && dummyChar1 == '-') {
               isValidNumber = true;
            }
         }
      }

      if (isValidNumber) {
```

```
Enter a 10-digit phone number
(or -1 to exit):
342-555-9084
   Standardized format: (342)
555-9084

*1778.555.2925
   Invalid phone number.

778.555.2925
   Invalid phone number.

(302)555-8927
   Standardized format: (302)
555-8927

-1
```

```
         cout << "   Standardized format: (" <<
areaCode << ") "
              << officeCode << "-" << stationNum <<
endl << endl;
      }
      else {
         cout << "   Invalid phone number." << endl <<
endl;
      }

      // Get next user input
      getline(cin, lineString);
   }

   return 0;
}
```



**PARTICIPATION ACTIVITY** 8.3.6: String stream error state.

1) Which phone number would result in "Invalid phone number." being printed to the screen?
   - 399+555@8493
   - (430) 555-2029
   - 224-555-7326

2) What function checks if the string stream is in an error state?
   - good()
   - eof()

Exploring further:

- stringstream Reference Page from cplusplus.com

**CHALLENGE ACTIVITY** 8.3.1: Input string streams.



Start

Type the program's output

```
#include <iostream>
#include <sstream>
#include <string>
using namespace std;

int main() {
   string objectInfo = "Book 10 19";
   istringstream objectISS(objectInfo);
   string object;
   int quantity;
   int price;

   objectISS >> object >> quantity >> price;

   cout << object << " x" << quantity << endl;
   cout << "Price: " << price;

   return 0;
}
```

```
Book x10
Price: 19
```



1 2

Check Next

**CHALLENGE ACTIVITY** 8.3.2: Reading from a string.

Write code that uses the input string stream inSS to read input data from string userInput, and updates variables userMonth, userDate, and userYear. Sample output if the input is "Jan 12 1992":

```
Month: Jan
Date: 12
Year: 1992
```

```
1  #include <iostream>
2  #include <sstream>
3  #include <string>
4  using namespace std;
5
6  int main() {
7     istringstream inSS;
8     string userInput;
9     string userMonth;
10    int userDate;
11    int userYear;
12
13    getline(cin, userInput);
14    inSS.str(userInput);
```

```
15    /* Your solution goes here  */
16
17    cout << "Month: " << userMonth << endl
```

Run

View your last submission



# 8.4 Output string stream

## Output string stream

An ***output string stream*** variable of type ***ostringstream*** can insert characters into a string buffer instead of the screen (standard output). ostringstream is derived from ostream. A program can insert characters into an ostringstream buffer using <<, just like the cout stream. The ostringstream member function ***str()*** returns the contents of an ostringstream buffer as a string.

**PARTICIPATION ACTIVITY** 8.4.1: Using an output string stream.

### Animation content:

undefined

### Animation captions:

1. The sstream library defines the ostringstream class, which is in namespace std.
2. ostringstream infoOSS; declares a new output string stream variable. The stream's buffer is initially empty.
3. After reading input from the keyboard, the program inserts data into infoOSS's buffer using <<, similar to inserting data into the cout stream.
4. The string " (minor)" is also inserted into the string stream buffer because the user entered an age < 21.
5. The str() function returns the stream's buffer data as a single string, which is then output to the screen.



**PARTICIPATION ACTIVITY** 8.4.2: Output string streams.

1) Given an ostringstream variable called

outSS, write a statement that appends the value of the int variable carSpeed to the stream's buffer.

Check Show answer



2) Write a statement that copies the contents of an output string stream called outSS to an existing string variable called myStr.

Check Show answer

## Example: Savings table

The example below uses an output string stream to create a savings table. The ProduceSavingsTable() function has 3 parameters: the starting amount, the annual percentage rate, and number of years. ProduceSavingsTable() uses several manipulators like fixed, setprecision, and setw to create a table with an ostringstream and returns the table as a single string.

Figure 8.4.1: Output string stream example.

```
#include <iostream>
#include <iomanip>
#include <string>
#include <sstream>
using namespace std;

string ProduceSavingsTable(double startAmount, double apr, int numYears) {
   // Column widths
   const int YEAR_COL_WIDTH = 5;
   const int BALANCE_COL_WIDTH = 10;

   ostringstream outSS;
   double interest;
   double balance = startAmount;
   int month;
   int totalMonths = numYears * 12;

   // Convert APR to monthly percentage rate and decimal number
   double mpr = apr / 12 * 0.01;

   // Display 2 decimal places
   outSS << fixed << setprecision(2);

   // Table heading
   outSS << setw(YEAR_COL_WIDTH) << "Year"
      << setw(BALANCE_COL_WIDTH) << "Balance" << endl;

   // Calculate interest and ending balance for each month
   for (month = 1; month <= totalMonths; ++month) {
      interest = balance * mpr;
      balance += interest;

      // Only output year number and balance at the end of the year
      if (month % 12 == 0) {
         outSS << setw(YEAR_COL_WIDTH) << month / 12
             << setw(BALANCE_COL_WIDTH) << balance << endl;
      }
   }

   // Return the table as a string
   return outSS.str();
}

int main() {
   string table;
   double startAmount;
   double apr;
   int years;

   // Get input values
   cout << "Starting amount?" << endl;
   cin >> startAmount;
   cout << "Annual Percentage Rate?" << endl;
   cin >> apr;
   cout << "Number of years?" << endl;
   cin >> years;

   cout << endl << "Savings over time:" << endl;
   table = ProduceSavingsTable(startAmount, apr, years);
   cout << table << endl;

   return 0;
}
```

```
Starting amount?
100
Annual Percentage Rate?
5
Number of years?
6

Savings over time:
 Year    Balance
    1     105.12
    2     110.49
    3     116.15
    4     122.09
    5     128.34
    6     134.90
```



**PARTICIPATION ACTIVITY** 8.4.3: Output string stream example.

1) Which manipulator sets the width of the Year and Balance columns?
   - setw
   - setprecision
   - fixed

2) Which modification would cause the balance amounts to round to the nearest dollar?
   - `outSS << scientific << setprecision(2);`
   - `outSS << fixed;`
   - `outSS << fixed << setprecision(0);`

3) Which statement can be added to ProduceSavingsTable() to produce a line of dashes across the table?
   - `cout << setfill('-') << setw(15) << "";`
   - `outSS << setw(15) << "-";`
   - `outSS << setfill('-') << setw(15) << "";`

**CHALLENGE ACTIVITY** 8.4.1: Output string streams.

Start



Type the program's output

```
#include <iostream>
#include <sstream>
#include <string>
using namespace std;

int main() {
   ostringstream objectOSS;
   string object;
   int quantity;
   int discount;

   object = "Shoes";
   quantity = 10;
   discount = 20;

   objectOSS << object << " x" << quantity << endl;
   objectOSS << discount << "% off";

   cout << objectOSS.str();

   return 0;
}
```

```
Shoes x10
20% off
```

1 2

Check Next

**CHALLENGE ACTIVITY** 8.4.2: Output using string stream.

Write code that inserts userItem into the output string stream itemsOSS until the user enters "Exit". Each item should be followed by a space. Sample output if user input is "red purple yellow Exit":



```
red purple yellow
```

```
1 #include <iostream>
2 #include <sstream>
3 #include <string>
```

```
4  using namespace std;
5
6  int main() {
7     string userItem;
8     ostringstream itemsOSS;
9
10    cout << "Enter items (type Exit to quit):" << endl;
11    cin >> userItem;
12
13    while (userItem != "Exit") {
14
15       /* Your solution goes here  */
16
17       cin >> userItem;
```



Run

View your last submission

# 8.5 File input

## Opening and reading from a file

Sometimes a program should get input from a file rather than from a user typing on a keyboard. To read file input, a programmer can create a new input stream that comes from a file, rather than the predefined input stream cin that comes from the standard input (keyboard). An input stream can then be used just like cin.

The `inFS.open(str)` function has a string parameter str that specifies the name of the file to open. The filename parameter can be a C++ string or a null-terminated C string. A program can also use a user-entered string as the filename, such as using `cin >> filename;`.

**PARTICIPATION ACTIVITY** 8.5.1: Input from a file.



### Animation content:

`undefined`

### Animation captions:

1. #include <fstream> (short for "file stream") enables use of the ifstream class.

2. A new stream variable, ifstream inFS, is declared. ifstream is short for input file stream, and is derived from istream.
3. inFS.open("numFile.txt"); opens the file for reading and associates the file with the inFS stream.
4. If the open fails, either because the file does not exist or is in use by another program, the program outputs an error message, and exits.
5. A successfully opened input stream can be used like the cin stream. inFS >> fileNum1; reads an integer into fileNum1.
6. When done using the stream, the program closes the file using inFS.close().



A common error is to type `cin >> num1;` when actually intending to get data from a file as in `inFS >> num1`. Another common error is a mismatch between the variable data type and the file data. If the data type is int but the file data is "Hello".

**PARTICIPATION ACTIVITY** 8.5.2: File input.

1) What is the error in the following code?

```
ifstream inFS;
int numBooks;
int numStudents = 40;
inFS >> numBooks;
cout << "Books per student: ";
cout << numBooks / numStudents;
```

- The file stream is not large enough.
- The file stream has not been properly opened.
- The >> operator cannot be used here.
- No error.

2) Which function checks that a file has been opened?

- open()
- close()
- is_open()



3) Which statement reads and stores data into integer variable inputNum from the file opened by the stream inFS?

- inFS >> inputNum;
- cin >> inputNum;

```
inFS.open(inputNum);
```

### File open() with C strings in older C++ versions

*Before C++11, earlier versions of the C++ standard required the inFS.open(str) function's str parameter to be a null-terminated C string, not a C++ string. To use a C++ string as the filename, the c_str() function that comes with C++ strings can be used. Ex: If filename is a C++ string, then filename.c_str() returns a null-terminated C string.*



## Reading until the end of the file

A program can read varying amounts of data in a file by using a loop that reads until the end of the file has been reached. The ***eof()*** function returns true if the previous stream operation reached the end of the file.

Errors may be encountered while attempting to read from a file, like the inability to read the file, reading corrupt data, etc. So, a program should check that each read was successful before using the variable to which the read data was assigned. The ***fail()*** function returns true if the previous stream operation had an error.

### Figure 8.5.1: Reading a varying amount of data from a file.

| Program | Example input file and output |
|---|---|
| | myfile.txt with variable number of integers:<br>`111`<br>`222`<br>`333`<br>`444`<br>`555` |
| | `Opening file myfile.txt.`<br>`Reading and printing numbers.`<br>`num: 111`<br>`num: 222`<br>`num: 333`<br>`num: 444`<br>`num: 555`<br>`Closing file myfile.txt.` |

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

int main() {
   ifstream inFS;    // Input file stream
   int fileNum;      // File data

   // Open file
   cout << "Opening file myfile.txt." << endl;
   inFS.open("myfile.txt");

   if (!inFS.is_open()) {
      cout << "Could not open file myfile.txt." <<
endl;
      return 1;
   }

   // Print read numbers to output
   cout << "Reading and printing numbers." << endl;

   inFS >> fileNum;
   while (!inFS.fail()) {
      cout << "num: " << fileNum << endl;
      inFS >> fileNum;
   }
   if (!inFS.eof()) {
      cout << "Input failure before reaching end of
file." << endl;
   }

   cout << "Closing file myfile.txt." << endl;

   // Done with file, so close it
   inFS.close();

   return 0;
}
```



**PARTICIPATION ACTIVITY** 8.5.3: File input.

1) Which statement determines if the following file read operation successfully read an integer?

```
inputFile >> vehicleCount;
```



- `if (inputFile.eof()) {`
- `if (!vehicleCount.eof()) {`
- `if (!inputFile.fail()) {`

## Example: Counting instances of a specific word

The following program uses both the extraction operator, eof(), and fail() to determine how many times a user entered word appears in a file. The number of words in the file is unknown, so the program extracts words until the end of the file is reached. The program exits if the stream extraction causes an error or after the word's frequency in the file is output.



Figure 8.5.2: How many times a word appears in a file.

**Program**

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main() {
   ifstream inFS;     // Input file stream
   string userWord;
   int wordFreq = 0;
   string currWord;

   // Open file
   cout << "Opening file wordFile.txt." << endl;
   inFS.open("wordFile.txt");

   if (!inFS.is_open()) {
      cout << "Could not open file wordFile.txt."
<< endl;
      return 1;
   }

   // Word to be found
   cout << "Enter a word: ";
   cin >> userWord;

   // Identify when a word matches the userWord
   // and increase wordFreq
   while (!inFS.eof()) {
      inFS >> currWord;
      if (!inFS.fail()) {
         if(currWord == userWord) {
            ++wordFreq;
         }
      }
   }

   cout << userWord << " appears in the file "
        << wordFreq << " times." << endl;

   // Done with file, so close it
   inFS.close();

   return 0;
}
```

**Example input file and output**

wordFile.txt with a list of words:

```
twenty
associable
twenty
unredacted
associable
folksay
twenty
```

```
Opening file wordFile.txt.
Enter a word: twenty
twenty appears in the file 3
times.
```

**PARTICIPATION ACTIVITY** 8.5.4: Counting instances of specific word example.

1) In the example above, how many times does the while loop iterate?
   - 3
   - 4
   - 7



2) What indicates that the end of the input file stream has been reached?
   - The eof() function
   - The extraction operator (>>)
   - The fail() function

3) If the user entered "associable" as the userWord, how many times would the while loop execute.
   - 2
   - 7

## Example: Business reviews

The following example reads a file with business reviews as the program starts and outputs data from the file at the end of the program. The number of reviews is unknown to the program, so the program continues to read until the end of the file. Each entry contains the username of the person who left the review and a 1 - 5 rating (1 being a low rating and 5 being a high rating). Upon completion, the program outputs the data from the file along with the average business rating.

Figure 8.5.3: Program that reads business reviews from a file and computes the average rating.

| Program | Example input file and output |
|---|---|
| `#include <iostream>`<br>`#include <fstream>`<br>`#include <string>`<br>`#include <vector>`<br>`using namespace std;`<br><br>`class Review {`<br>`public:`<br>`void SetUserName(string revUserName) {`<br>`userName = revUserName;` | Trattoria_Reviews.txt with a list of usernames and ratings: |

```
      }
      void SetRating(int revRating) {
         rating = revRating;
      }
      string GetUserName() const { return
userName; }
      int GetRating() const { return rating; }

   private:
      string userName = "NoName";
      int rating = -1;
};


void ReadReviews(string& restaurantName,
vector<Review>& reviewList) {
   ifstream inFS;     // Input file stream
   string userName;
   int userRating;
   Review currentReview;

   // Open file
   inFS.open("Trattoria_Reviews.txt");

   if (!inFS.is_open()) {
      cout << "Could not open file
Trattoria_Reviews.txt."<< endl;
   }

   getline(inFS, restaurantName);

   while (!inFS.eof()) {
      inFS >> userName;
      inFS >> userRating;

      if (!inFS.fail()) {
         currentReview.SetUserName(userName);
         currentReview.SetRating(userRating);
         reviewList.push_back(currentReview);
      }
   }

   // Close file when done reading
   inFS.close();
}

double CalcAvgRating(const vector<Review>&
reviewList) {
   int i;
   double ratingAvg = 0;

   for(i = 0; i < reviewList.size(); ++i) {
      ratingAvg += reviewList.at(i).GetRating();
   }
   return ratingAvg /= reviewList.size();
}

void DisplayReviews(const string& restaurantName,
                    const vector<Review>&
reviewList,
                    const double ratingAvg) {
   int i;

   cout << endl << restaurantName << endl;
   cout << "Average rating: " << ratingAvg <<
endl;
   cout << "--------------------" << endl;

   for(i = 0; i < reviewList.size(); ++i) {
      cout << "User name: " <<
```

```
Trattoria Italian Bistro

sunny8trophy
4

Angelcopter
2

Mogoodid24
5

Elixirnoel8626
5

gobbledygook
1

Friderday912
3
```



```
Trattoria Italian Bistro's
Average Rating
Average rating: 3.33333
--------------------
User name: sunny8trophy
   Rating: 4

User name: Angelcopter
   Rating: 2

User name: Mogoodid24
   Rating: 5

User name: Elixirnoel8626
   Rating: 5

User name: gobbledygook
   Rating: 1

User name: Friderday912
   Rating: 3
```

©zyBooks 04/25/21 07:14 488201
xiang zhao
BAYLORCSI14301440Spring2021

```
reviewList.at(i).GetUserName() << endl;
      cout << "   Rating: " <<
reviewList.at(i).GetRating() << endl;
      cout << endl;
   }
}

int main() {
   double ratingAvg;
   vector<Review> reviewList;
   string restaurantName;

   // Reads restaurant name and reviews from input
file at program start
   ReadReviews(restaurantName, reviewList);

   ratingAvg = CalcAvgRating(reviewList);
   DisplayReviews(restaurantName, reviewList,
ratingAvg);

   return 0;
}
```



**PARTICIPATION ACTIVITY** 8.5.5: Reading and using data from a file.

Refer to the program above.

1) Which statement reads a full line of multiple strings from an input file?
   - inFS >> lineString;
   - getline(inFS, lineString);

2) How is data read from the file stored in the program?
   - In an array
   - In a vector
   - The data is read and output immediately without being stored.



## Input stream errors

A ***stream error*** occurs when insertion or extraction fails, causing the stream to enter an error state. Ex: If a file has the string `two` but the program attempts to extract an integer, the extraction will fail and the stream will enter an error state.

An input stream may also enter an error state if a value extracted is too large (or small) to fit in the given variable. While in an error state, an input stream may: skip extraction, set the given variable to 0,

or set the given variable to the maximum (or minimum) value of that variable's data type.

A stream internally uses several 1-bit error flags to track the state of the stream. A program can check a stream's error state using several stream functions that return the current state. A stream's error state is cleared using clear().

### Table 8.5.1: Stream error state flags and functions to check error state.



| Flag | Meaning | Function |
|---|---|---|
| goodbit | Indicates no error flags are set and the stream is good. | ***good()*** returns true if no stream errors have occurred. |
| eofbit | Indicates if end-of-file reached on extraction. | ***eof()*** returns value of eofbit, if end-of-file reached on extraction. |
| failbit | Indicates a logical error for the previous extraction or insertion operation. | ***fail()*** returns true if either failbit or badbit is set, indicating an error for the previous stream operation. |
| badbit | Indicates an error occurred while reading or writing the stream, and the stream is bad. Further operations on the stream will fail. | ***bad()*** returns true if badbit is set, indicating the stream is bad. |

**PARTICIPATION ACTIVITY** 8.5.6: Check for errors while reading a file.

## Animation content:

undefined

## Animation captions:

1. After the input stream opens the file, the program uses the good() function to ensure the stream is in an error free state.
2. The program expects the file to contain reviews in the format of a string followed by an integer.
3. After reading "Vancity93", the program tries to extract an integer, but the next character is the letter 't'. The extraction fails, and the stream's error state is updated by setting the goodbit to 0 and the failbit to 1.
4. The while loop terminates because inFS.good() now returns false. Then, as the end-of-file was not reached before the stream error occurred, the program outputs an error message and

exits.

**PARTICIPATION ACTIVITY** 8.5.7: Stream error functions.

1) Which returns whether a logical error occurred with file input stream inFS?
   - inFS(fail)
   - inFS.fail()
   - fail(inFS)

2) Which function wouldn't be used to check cout's error state?
   - good()
   - bad()
   - eof()



Exploring further:

- fstream Reference Page from cplusplus.com

# 8.6 C++ example: Parsing and validating input files

The following program reads input from the teams.txt file, which contains a baseball team name on one line followed by an optional line with the number of wins and losses for the season. Some win/loss lines only have a single number representing the number of wins. The file may have any number of team name and win/loss lines.



Figure 8.6.1: Program that reads from teams.txt.

| | teams.txt |
|---|---|

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main() {
   ifstream teamFS;
   string teamName;
   int numWins;
   int numLosses;

   teamFS.open("teams.txt");

   if (!teamFS.is_open()) {
      cout << "Could not open file teams.txt." << endl;
      return 1;
   }

   // Read first team name
   getline(teamFS, teamName);

   // Read until end-of-file
   while (!teamFS.fail()) {
      // Attempt to read wins
      teamFS >> numWins;

      if (teamFS.fail()) {
         // Win/loss line missing
         cout << teamName << " has no wins or losses"
<< endl;
      }
      else {
         // Attempt to read losses
         teamFS >> numLosses;

         if (teamFS.fail()) {
            // No losses provided
            cout << teamName << " has " << numWins << "
wins" << endl;
         }
         else {
            // Win and losses provided
            cout << teamName << " win average is "
                 << static_cast<double>(numWins) /
(numWins + numLosses) << endl;
         }

         // Remove newline
         teamFS.ignore();
      }

      // Clear the error state
      teamFS.clear();

      // Attempt to read next team
      getline(teamFS, teamName);
   }

   teamFS.close();

   return 0;
}
```

```
Arkansas Travelers
5 11
Vancouver Canadians
10 3
Columbus Clippers
Norfolk Tides
6
Rocky Mountain Vibes
8 8
```



```
Arkansas Travelers win
average is 0.3125
Vancouver Canadians win
average is 0.769231
Columbus Clippers has no
wins or losses
Norfolk Tides has 6 wins
Rocky Mountain Vibes win
average is 0.5
```

©zyBooks 04/25/21 07:14 488201
xiang zhao
BAYLORCSI14301440Spring2021

**PARTICIPATION ACTIVITY** 8.6.1: Parsing input files.

1) What is output when the win/loss line is missing?
   - *Team* has no wins or losses
   - *Team* has *W* wins
   - *Team* win average is *A*

2) What is output when the loss number is missing?
   - *Team* has no wins or losses
   - *Team* has *W* wins
   - *Team* win average is *A*

3) Which istream member function is called to verify teams.txt is opened successfully?
   - fail()
   - clear()
   - is_open()

4) Which istream member function is called to verify a team name, win number, or loss number was successfully read?
   - fail()
   - clear()
   - ignore()

5) What is the last line output if `teamFS.clear()` is removed from the program?
   - Vancouver Canadians win average is 0.769231
   - Columbus Clippers has no wins or losses
   - Rocky Mountain Vibes win average is 0.5

# 8.7 File output

## Opening and writing to a file



An ***ofstream***, short for "output file stream", is a class that supports writing to a file. The ofstream class inherits from ostream.

After declaring a variable of type ofstream, a file is opened using the ofstream's open() function. The ofstream's is_open() function is commonly called to check if the file opened successfully. If so, data can be written to the file using the << operator. When all desired data is written, the ofstream's close() function is called to close the file.

Table 8.7.1: Basic steps for opening and writing a file.

| Action | Sample code |
|---|---|
| Open the file helloWorld.txt for writing | `ofstream outFS;`<br>`outFS.open("helloWorld.txt");` |
| Check to see if the file opened successfully | `if (!outFS.is_open()) {`<br>`  // Do not proceed to code that writes to the file`<br>`}` |
| Write the string "Hello World!" to the file | `outFS << "Hello World!" << endl;` |
| Close the file after writing all desired data | `outFS.close();` |

**PARTICIPATION ACTIVITY** 8.7.1: Opening and writing to a file.



1) ofstream's open function takes _____.

- 0 arguments
- a string for a file name as an argument

2) If is_open() returns false, execution

_____ proceed to code that writes to the file.

- should
- should not

3) Data can be written to the file _____.

- before calling open()
- after the file is opened successfully, but before the file is closed
- at any time



## Example: Writing a text file

Calling an ofstream's open() function makes an attempt to open the file. The file may fail to open due to things like file permissions or a full disk. The is_open() should be called to check if the file opened successfully. is_open() returns true if the file is open, false otherwise.

The file is created and is initially empty, if opened successfully. Data is written to the file, then the file is closed.

**PARTICIPATION ACTIVITY** 8.7.2: Writing to an output text file.

### Animation content:

```
undefined
```

### Animation captions:

1. An ofstream named outFS is declared, then myoutfile.txt is opened for writing. The file is initially empty.
2. Since the file opened successfully, outFS.is_open() returns true.
3. Two lines of text are written to the file, then the file is closed.



**PARTICIPATION ACTIVITY** 8.7.3: Writing a text file.

1) In the code above, the statement `outFS.close()` is not executed if is_open() returns false.

- True

False

2) Since no data is written, the code below never creates the myoutfile.txt file on disk.

```
outFS.open("myoutfile.txt");
outFS.close()
```

- True
- False



3) An ofstream should be closed using the close() function.

- True
- False

## Example: Writing a simple HTML file

HTML files are used for web pages and consist of text content. Therefore, an HTML file can be written similar to a text file.

**PARTICIPATION ACTIVITY** 8.7.4: Writing simple html to a file and to the console.

### Animation content:

undefined

### Animation captions:

1. Output file simple.html is opened for writing.
2. Since ofstream inherits from ostream and outFS is an ofstream, outFS can be passed by reference as an argument to writeHTMLFile().
3. writeHTMLFile() writes HTML content to the file, then the file is closed.
4. cout can also be passed to writeHTMLFile(), since cout is an ostream instance. The same content is written to the console.



**PARTICIPATION ACTIVITY** 8.7.5: Writing a simple HTML file.

1) writeHTMLFile can write to either outFS or cout because both are instances of a class that inherits from _____.

- ofstream
- ostream
- string

2) The _____ operator allows writing a string to an ostream.

- >>
- <<
- +



3) A better implementation of writeHTMLFile would close the stream.

- True
- False

# 8.8 C++ example: Saving and retrieving program data

The following program reads restaurant reviews from the reviews.txt file into a vector of strings. The user is prompted to enter additional reviews, which are appended to the string vector. Each time a new review is entered, the complete list of reviews are output to the console.

Figure 8.8.1: Program that reads restaurant reviews from reviews.txt.

reviews.txt

```
The salads are really good.
Service was a little slow, but the
food was decent.
My favorite place in town!
```

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
using namespace std;

void ReadReviews(vector<string>
&reviewList) {
   ifstream reviewsFS;
   string review;

   reviewsFS.open("reviews.txt");

   if (!reviewsFS.is_open()) {
      cout << "Could not open file
reviews.txt." << endl;
   }
   else {
      getline(reviewsFS, review);
      while (!reviewsFS.fail()) {
         reviewList.push_back(review);
         getline(reviewsFS, review);
      }

      reviewsFS.close();
   }
}

void DisplayReviews(const vector<string>
&reviewList) {
   cout << endl << "Reviews:" << endl;
   for (int i = 0; i < reviewList.size();
i++) {
      cout << i + 1 << ". " <<
reviewList.at(i) << endl;
   }
   cout << endl;
}

int main() {
   vector<string> reviewList;
   string newReview;

   // Read reviews from file into reviews
vector and display
   ReadReviews(reviewList);
   DisplayReviews(reviewList);

   cout << "Enter new review or QUIT:" <<
endl;
   getline(cin, newReview);
   while (newReview != "QUIT") {
      // Add new reviews to the vector and
display new list
      reviewList.push_back(newReview);
      DisplayReviews(reviewList);

      cout << "Enter new review or QUIT:"
<< endl;
      getline(cin, newReview);
   }

   return 0;
}
```

```
Reviews:
1. The salads are really good.
2. Service was a little slow, but the
food was decent.
3. My favorite place in town!

Enter new review or QUIT:
Prices are good for the amount of food
you get.

Reviews:
1. The salads are really good.
2. Service was a little slow, but the
food was decent.
3. My favorite place in town!
4. Prices are good for the amount of
food you get.

Enter new review or QUIT:
QUIT
```



©zyBooks 04/25/21 07:14 488201
xiang zhao
BAYLORCSI14301440Spring2021

**PARTICIPATION ACTIVITY** 8.8.1: Saving and retrieving program data.

1) How many reviews appear on each line in reviews.txt?
   - 1
   - 2
   - All



2) If the user typed a second review in the example run above instead of QUIT, what would be the size of the reviews vector sent to DisplayReviews()?
   - 3
   - 4
   - 5

3) If the user runs the program above, enters 2 reviews, quits, and runs the program a second time, how many reviews will initially be displayed?
   - 3
   - 4
   - 5

**PARTICIPATION ACTIVITY** 8.8.2: Save the reviews.

The program is not saving new user reviews, so a new function SaveReviews() should be called after the user enters QUIT to save the reviews vector to reviews.txt.



```
void SaveReviews(const vector<string> &reviews) {
   __A__ reviewsFS;
   reviewsFS.__B__;

   for (int i = 0; i < reviews.size(); i++) {
      __C__ << reviews.at(i) << endl;
   }

   reviewsFS.__D__;
}
```

Enter the correct code that replaces the letters in the SaveReview() function above.

1) A's replacement?

Check Show answer

2) B's replacement?



Check Show answer

3) C's replacement?

Check Show answer

4) D's replacement?

Check Show answer

# 8.9 Overloading stream operators

## Overloading the << operator

The C++ << operator is known as the ***insertion operator***. A C++ class can overload the insertion operator by creating a member function named operator<<.

Ex: A VoteCounter class may count a vote for a candidate by using the << operator. The candidate's name is passed as a string argument, and the VoteCounter object is returned. So the member function declaration is `VoteCounter& operator<<(const string& candidateName);`.



Figure 8.9.1: VoteCounter class demo.

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

class VoteCounter
{
private:
```

```
   int ACount, BCount, CCount;
public:
   VoteCounter() { ACount = BCount = CCount = 0; }
   VoteCounter& operator<<(const string& candidateName);
   string GetResults();
};

VoteCounter& VoteCounter::operator<<(const string& candidateName) {
   if ("Candidate A" == candidateName)
      ACount++;
   else if ("Candidate B" == candidateName)
      BCount++;
   else if ("Candidate C" == candidateName)
      CCount++;

   return *this;
}

string VoteCounter::GetResults() {
   if (ACount > BCount) {
      if (ACount > CCount)
         return "Candidate A wins!";
      else if (ACount < CCount)
         return "Candidate C wins!";
      else
         return "Tie between candidates A & C!";
   }
   else if (ACount < BCount) {
      if (BCount > CCount)
         return "Candidate B wins!";
      else if (BCount < CCount)
         return "Candidate C wins!";
      else
         return "Tie between candidates B & C!";
   }
   else {
      if (BCount == CCount)
         return "Tie between all 3 candidates!";
      else
         return "Tie between candidates A & B!";
   }
}

int main() {
   const string candidateNames[] = {
      "Candidate A", "Candidate B", "Candidate C"
   };
   const int voteIndices[] = {
      2, 1, 0, 1, 2, 0, 0, 1, 2, 1, 0, 1, 0, 0, 1, 2, 0, 0, 1
   };
   const int voteCount = sizeof(voteIndices) / sizeof(int);

   // Cast the votes
   cout << "Counting votes..." << endl;
   VoteCounter counter;
   for (int i = 0; i < voteCount; i++) {
      int voteIndex = voteIndices[i];
      counter << candidateNames[voteIndex];
   }

   // Reveal the winner
   cout << counter.GetResults() << endl;

   return 0;
}
```



```
Counting votes...
Candidate A wins!
```

**PARTICIPATION ACTIVITY** 8.9.1: VoteCounter << operator.

1) The statement `counter << "candidate c";` would count a vote for candidate C.
   - True
   - False



2) The statement `counter << "Candidate A" << "Candidate B";` would count 2 votes, one for candidate A and another for candidate B.
   - True
   - False

3) If the operator<< member function's return type were changed to void and the return statement removed, a compiler error would occur.
   - True
   - False

## Overloading the >> operator

The C++ >> operator is known as the ***extraction operator***. A C++ class can overload the extraction operator by creating a member function named operator>>.

Ex: The WaitingLine class uses the insertion operator to add a string to the back of the line and the extraction operator to remove a string from the front of the line.

Figure 8.9.2: WaitingLine class.

```
class WaitingLine
{
public:
   WaitingLine& operator<<(const string& name) {
      // Add the name to the end of the line
      line.push(name);

      cout << name << " enters the back of the line" << endl;

      return *this;
   }
   WaitingLine& operator>>(string& frontName) {
      // Copy the name at the front of the line to frontName, then remove
      frontName = line.front();
      line.pop();

      return *this;
   }

   queue<string> line;
};
```



**PARTICIPATION ACTIVITY** 8.9.2: The insertion and extraction operators add and remove from the WaitingLine object.

## Animation content:

undefined

## Animation captions:

1. Three strings are inserted into the waiting line using the insertion operator: Lion, Tiger, and Bear.
2. Lion and Tiger are removed using the extraction operator.
3. Duck and Goose are added to the back of the line with the insertion operator.
4. The remaining strings are removed from the waiting line with the extraction operator.

**PARTICIPATION ACTIVITY** 8.9.3: Overloading the extraction operator.



1) WaitingLine's extraction operator adds a string to the front of the line.

   - True
   - False

2) WaitingLine's insertion operator adds a

string to the front of the line.

- True
- False

3) WaitingLine's extraction operator removes an empty string if the line is empty.

- True
- False



## Extending cin and cout

By default, a programmer-defined C++ class does not work with cin and cout. Statements like `cin >> line1` or `cout << line1`, where line1 is a WaitingLine object, cause a compiler error. But the functionality of cin and cout can be extended by implementing certain friend functions in the C++ class.

Figure 8.9.3: WaitingLine class with stream friend functions.

```
class WaitingLine
{
public:
   WaitingLine& operator<<(const string& name) {
      // Add the name to the end of the line
      line.push(name);

      cout << name << " enters the back of the line" << endl;

      return *this;
   }

   WaitingLine& operator>>(string& frontName) {
      // Copy the name at the front of the line to frontName, then remove
      frontName = line.front();
      line.pop();

      return *this;
   }

   friend ostream& operator<<(ostream& out, const WaitingLine& line) {
      out << "(front)";
      queue<string> lineCopy = line.line;
      while (!lineCopy.empty()) {
         string lineItem = lineCopy.front();
         lineCopy.pop();
         out << " " << lineItem;
      }
      out << " (back)";
      return out;
   }

   friend istream& operator>>(istream& in, WaitingLine& line) {
      string inString;
      in >> inString;
      line << inString;
      return in;
   }

   queue<string> line;
};
```



**PARTICIPATION ACTIVITY** 8.9.4: Using cin and cout with the WaitingLine class.

## Animation content:



undefined

## Animation captions:

1. Since friend istream& operator>> is implemented in the WaitingLine class, cin >> line1 can be used to get input from the user and add to the line.
2. A second, hard-coded string is added to the list. cout is then used to display the full list.

**PARTICIPATION ACTIVITY** 8.9.5: Using cin and cout with the WaitingLine class.

1) The statement `cin >> line1` calls the _____ function.
   - friend ostream& operator<<
   - friend istream& operator>>
   - WaitingLine& operator>>



2) Objects other than cout can be used to write WaitingLine contents. The statement `file1 << line1` would work provided file1 is an instance of _____.
   - ostream
   - istream
   - WaitingLine

3) In the animation above, the user could have typed `one two` instead of `one`, thus adding _____, to line1.
   - two strings, "one" and "two"
   - one string, "one two"
   - one string, "one"

# 8.10 LAB: Warm up: Parsing strings

(1) Prompt the user for a string that contains two strings separated by a comma. (1 pt)

- Examples of strings that can be accepted:
  - Jill, Allen
  - Jill , Allen
  - Jill,Allen



Ex:

```
Enter input string:
Jill, Allen
```

(2) Print an error message if the input string does not contain a comma. Continue to prompt until a valid string is entered. *Note: If the input contains a comma, then assume that the input also contains two strings.* (2 pts)

Ex:

```
Enter input string:
Jill Allen
Error: No comma in string.

Enter input string:
Jill, Allen
```



(3) Extract the two words from the input string and remove any spaces. Store the strings in two separate variables and output the strings. (2 pts)

Ex:

```
Enter input string:
Jill, Allen
First word: Jill
Second word: Allen
```

(4) Using a loop, extend the program to handle multiple lines of input. Continue until the user enters q to quit. (2 pts)

Ex:

```
Enter input string:
Jill, Allen
First word: Jill
Second word: Allen

Enter input string:
Golden , Monkey
First word: Golden
Second word: Monkey

Enter input string:
Washington,DC
First word: Washington
```

```
Second word: DC

Enter input string:
q
```

LAB ACTIVITY | 8.10.1: LAB: Warm up: Parsing strings | 7 / 7



main.cpp

Load default template...

```
1  #include <iostream>
2  #include <string>
3  #include <sstream>
4  using namespace std;
5
6  int main() {
7     string line;
8
9     string firstWord;
10    string secondWord;
11    istringstream iss;
12
13    do {
14       cout << "Enter input string:" << endl;
15       getline(cin, line);
16
17       if (line != "q") {
```

Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

JillAllen

Run program

Input (from above) → main.cpp (Your program) → Outp

Program output displayed here



Signature of your work What is this?

```
4/15.. R--------|3|1|1|3|5|5|7|7 ..4/15
```

# 8.11 LAB*: Program: Data visualization

(1) Prompt the user for a title for data. Output the title. (1 pt)



Ex:

```
Enter a title for the data:
Number of Novels Authored
You entered: Number of Novels Authored
```

(2) Prompt the user for the headers of two columns of a table. Output the column headers. (1 pt)

Ex:

```
Enter the column 1 header:
Author name
You entered: Author name

Enter the column 2 header:
Number of novels
You entered: Number of novels
```

(3) Prompt the user for data points. Data points must be in this format: *string, int*. Store the information before the comma into a string variable and the information after the comma into an integer. The user will enter `-1` when they have finished entering data points. Output the data points. Store the string components of the data points in a vector of strings. Store the integer components of the data points in a vector of integers. (4 pts)

Ex:

```
Enter a data point (-1 to stop input):
Jane Austen, 6
Data string: Jane Austen
Data integer: 6
```



(4) Perform error checking for the data point entries. If any of the following errors occurs, output the appropriate error message and prompt again for a valid data point.

- If entry has no comma
  - Output: **`Error: No comma in string.`** (1 pt)
- If entry has more than one comma
  - Output: **`Error: Too many commas in input.`** (1 pt)
- If entry after the comma is not an integer
  - Output: **`Error: Comma not followed by an integer.`** (2 pts)



Ex:

```
Enter a data point (-1 to stop input):
Ernest Hemingway 9
Error: No comma in string.

Enter a data point (-1 to stop input):
Ernest, Hemingway, 9
Error: Too many commas in input.

Enter a data point (-1 to stop input):
Ernest Hemingway, nine
Error: Comma not followed by an integer.

Enter a data point (-1 to stop input):
Ernest Hemingway, 9
Data string: Ernest Hemingway
Data integer: 9
```

(5) Output the information in a formatted table. The title is right justified with a setw() value of 33. Column 1 has a setw() value of 20. Column 2 has a setw() value of 23. (3 pts)

Ex:

```
        Number of Novels Authored
Author name          |       Number of novels
-----------------------------------------------
Jane Austen          |                        6
Charles Dickens      |                       20
Ernest Hemingway     |                        9
Jack Kerouac         |                       22
F. Scott Fitzgerald  |                        8
Mary Shelley         |                        7
Charlotte Bronte     |                        5
Mark Twain           |                       11
Agatha Christie      |                       73
```

```
Ian Flemming        |                    14
J.K. Rowling        |                    14
Stephen King        |                    54
Oscar Wilde         |                     1
```

(6) Output the information as a formatted histogram. Each name is right justified with a setw() value of 20. (4 pts)



Ex:

```
         Jane Austen ******
     Charles Dickens ******************
    Ernest Hemingway *********
        Jack Kerouac **********************
 F. Scott Fitzgerald ********
        Mary Shelley *******
    Charlotte Bronte *****
          Mark Twain ***********
     Agatha Christie
**************************************************************************

        Ian Flemming **************
        J.K. Rowling **************
        Stephen King
**************************************************
         Oscar Wilde *
```

**LAB ACTIVITY** | 8.11.1: LAB*: Program: Data visualization | 17 / 17

main.cpp — Load default template...

```
#include <iostream>
#include <string>
#include <vector>
#include <sstream>
#include <iomanip>
using namespace std;


void printTable(string title, string header1, string header2, vector<string>& col1, vecto
   cout << setw(33) << title << endl;
   cout << left << setw(20) << header1 << '|';
   cout << right << setw(23) << header2 << endl;
   cout << setfill('-') << setw(44) << "" << endl;
```

```
14    cout << setfill(' ');
15
16    for (unsigned int i = 0; i < col1.size(); i++) {
17       cout << left << setw(20) << col1[i] << '|';
```

Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.



Enter program input (optional)

```
Number of Novels Authored
Author name
Number of novels
Ernest Hemingway 9
Ernest, Hemingway, 9
Ernest Hemingway, nine
Ernest Hemingway, 9
```

Run program | Input (from above) → main.cpp (Your program) → Outp

Program output displayed here

Signature of your work What is this?

```
4/15.. R|0|0|1|1|2|0----------|9|0|2|10|10|0|10|13|13|13|17
 ..4/15
```

# 8.12 LAB: Parsing dates

Complete main() to read dates from input, one date per line. Each date's format must be as follows: March 1, 1990. Any date not following that format is incorrect and should be ignored. Use the substr() function to parse the string and extract the date. The input ends with -1 on a line alone. Output each correct date as: 3/1/1990.



Ex: If the input is:

```
March 1, 1990
April 2 1995
```

```
7/15/20
December 13, 2003
-1
```

then the output is:

```
3/1/1990
12/13/2003
```



**LAB ACTIVITY** | 8.12.1: LAB: Parsing dates | 10 / 10

main.cpp — Load default template...

```
1  #include <iostream>
2  #include <string>
3
4  using namespace std;
5
6  int DateParser(string month) {
7     int monthInt = 0;
8
9
10    if (month == "January")
11       monthInt = 1;
12    else if (month == "February")
13       monthInt = 2;
14    else if (month == "March")
15       monthInt = 3;
16    else if (month == "April")
17       monthInt = 4;
```

**Develop mode** | **Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

```
March 1, 1990
April 2 1995
7/15/20
December 13, 2003
-1
```



**Run program** Input (from above) → **main.cpp** (Your program) → Outp

Program output displayed here

Signature of your work What is this?

```
4/16.. F--------------------|10|10 ..4/16
```



©zyBooks 04/25/21 07:14 488201
xiang zhao
BAYLORCSI14301440Spring2021