==========

STRINGSTREAMS

==========

String Streams

- We saw how a stream can be connected to a file.
- A stream can also be connected to a string.
- With stringstreams you can perform input/output from/to a string.
- This allows you to convert numbers
  (or any type with the << and >> stream operators overloaded) to and from strings.
- To use stringstreams =>
  - `#include <sstream>`
- The `istringstream` class reads characters from a string
- The `ostringstream` class writes characters to a string.

Stringstream uses

- A very common use of string streams is:
  - to accept input one line at a time and then to analyze it further.
    - by using stringstreams you can avoid mixing `cin >>` ... and `getline()`
    - *see examples in the following pages*
  - to use standard output manipulators to create a formatted string

istringstream

- Using an `istringstream`,
  you can read numbers that are stored in a string by using the >> operator:

```
string input = "March 25, 2014";
istringstream instr(input); //initializes 'instr' with 'input'

string month, comma;
int day, year;
instr >> month >> day >> comma >> year;
```

- Note that this input statement yields `day` and `year` as integers.
  Had we taken the string apart with `substr`, we would have obtained only strings.
- Converting strings that contain digits to their integer values is such a common operation
  that it is useful to write a helper function for that purpose:

```
int string_to_int(string s)
{
   istringstream instr;
   instr.str(s); // ALTERNATIVE way to initialize 'instr' with 's'
                 // to the initialization mode used above
   int n;
   instr >> n;
   return n;
}
```

ostringstream

- By writing to a string stream, you can convert numbers to strings.
- By using the << operator, the number is converted into a sequence of characters.

```
ostringstream outstr;
outstr << setprecision(5) << sqrt(2);
```

- To obtain a string from the stream, call the `str` member function.
  - `string output = outstr.str();`
- Example: (builds the string "`January 23, 1955`")

```
string month = "January";
int day = 23;
int year = 1955;
ostringstream outstr;
outstr << month << " " << day << "," << year;
string output = outstr.str();
```

- Converting an integer into a string is such a common operation that is useful to have a helper function for it.

```
string int_to_string(int n)
{
   ostringstream outstr;
   outstr << n;
   return outstr.str();
}
```

String ←→ Number conversion in C++11

- **C++11** introduced some standard library functions that can directly convert basic types to `std::string` objects and vice-versa.
- These functions are declared in declared in `<string>`.
- `std::to_string()` converts basic numeric types to strings.
  - Example:

```
int number = 123;
string text = to_string(number);
```

- The set of functions
  - `std::stoi, std::stol, std::stoll` - convert to integral types
  - `std::stof, std::stod, std::stold` - convert to floating-point values.
  - Example:

```
text = "456"
number = stoi(number);
```

```
/**
READ TIME IN SEVERAL FORMATS
ex:
21:30
9:30 pm
10 am
and show it in "military format" (HH:MM) and "am/pm format" (HH:MM am/pm)
*/
#include <iostream>
#include <string>
#include <sstream>
using namespace std;

/**
Converts an integer value to a string, e.g. 3 -> "3".
@param s an integer value
@return the equivalent string
*/
string int_to_string(int n)
{
      ostringstream outstr;
      outstr << n;
      return outstr.str();       //convert stringstream into string
}
/**
Reads a time from standard input
in the format hh:mm or hh:mm am or hh:mm pm
@param hours filled with the hours
@param minutes filled with the minutes
*/
void read_time(int &hours, int &minutes)
{
      string line;
      string suffix;
      char ch;

      getline(cin, line);

      istringstream instr(line);  //initialize stringstream from string
                                  // ALTERNATIVE:
                                  // istringstream  instr;
                                  // instr.str(line);

     instr >> hours;

      minutes = 0;

      instr.get(ch);    // do {instr.get(ch);} while (ch==' ');  // EFFECT ?
                        // try with 18:45 and 18: 45 and 18 :45 and 18 : 45
      if (ch == ':')
            instr >> minutes;
      else
            instr.unget(); // OR instr.putback(ch);

      instr >> suffix;
      if (suffix == "pm")
            hours = hours + 12;
}
```

```
/**
Computes a string representing a time.
@param hours the hours (0...23)
@param minutes the minutes (0...59)
@param military
 true for military format,
 false for am/pm format,
*/
string time_to_string(int hours, int minutes, bool military)
{
      string suffix;
      string result;

      if (!military)
      {
            if (hours < 12)
                  suffix = "am";
            else
            {
                  suffix = "pm";
                  hours = hours - 12;
            }
            if (hours == 0) hours = 12;
      }

      result = int_to_string(hours) + ":";
      if (minutes < 10) result = result + "0";
      result = result + int_to_string(minutes);

      if (!military)
            result = result + " " + suffix;

      return result;
}


int main()
{
      int hours;
      int minutes;

      do
      {
            cout << "Please enter the time\n";
            cout << "HH[:MM] or HH[:MM] am or HH[:MM] pm (0:0 => END): ";

            read_time(hours, minutes);

            cout << "Military time: "
                  << time_to_string(hours, minutes, true) << "\n";
            cout << "Using am/pm:   "
                  << time_to_string(hours, minutes, false) << "\n";
            cout << endl;

      } while (hours!=0 || minutes!=0);  // TO DO by students

      return 0;
}
```

```
/*
Read fractions and do arithmetic operations with them

STRINGSTREAMS
By using STRINGTREAMS you can avoid mixing cin << ... and getline(cin, ...)
You may always use getline()
*/

/*
TO DO:
Fraction sumFractions(Fraction f1, Fraction f2)
Fraction subtractFractions(Fraction f1, Fraction f2)
Fraction divideFractions(Fraction f1, Fraction f2)
*/

#include <iostream>
#include <iomanip>
#include <cctype>
#include <string>
#include <sstream>

using namespace std;

struct Fraction
{
	int numerator;
	int denominator;
};

/*  // READING / WRITING    DIRECTLY FROM / TO    cin / cout
bool readFraction(Fraction &f)
{
	char fracSymbol;
	int numerator;
	int denominator;
	bool success;

	cout << "n / d ? ";
	cin >> numerator >> fracSymbol >> denominator;
	if (cin.fail())
	{
		cin.clear();
		success = false;
	}
	else
		if (fracSymbol == '/')
		{
			f.numerator = numerator;
			f.denominator = denominator;
			success = true;
		}
		else
			success = false;

	cin.ignore(1000,'\n');

	return success;
}
*/
```

```
bool readFraction(Fraction &f)
{
  string fractionString;
  char fracSymbol;
  int numerator;
  int denominator;
  bool success;

  cout << "n / d ? ";
  getline(cin,fractionString);

  istringstream fractionStrStream(fractionString);
  if (fractionStrStream >> numerator >> fracSymbol >> denominator)
    if (fracSymbol == '/')
    {
      f.numerator = numerator;
      f.denominator = denominator;
      success = true;
    }
    else                // TO DO: write these tests in a different way
      success = false; // suggestion: initialize 'success'
  else
    success = false;
  return success;
}

Fraction multiplyFractions(Fraction f1, Fraction f2)
{
      Fraction f;

      f.numerator = f1.numerator * f2.numerator;
      f.denominator = f1.denominator * f2.denominator;
      return f;

}

void showFraction(Fraction f)
{
      cout << f.numerator << "/" << f.denominator;
}

int main()
{
      Fraction f1, f2, f3;

      cout << "Input 2 fractions:\n";
      if (readFraction(f1) && readFraction(f2))
      {
            f3 = multiplyFractions(f1,f2);

            cout << "Product: ";
            showFraction(f3);
      }
      else
      {
            cout << "Invalid fraction\n";
      }
      cout << endl;

      return 0;
}
```