```
      while (more1 && more2)
        if (n1 < n2) more1 = copy(fout, fin1, n1);
        else more2 = copy(fout, fin2, n2);
      while (more1)
        more1 = copy(fout, fin1, n1);
      while (more2)
        more2 = copy(fout, fin2, n2);
      fout << endl;
    }
```

The `more()` function is used to read the data from the input files. Each call attempts to read one integer from the `fin` file to the reference parameter `n`. It returns `true` if it is successful, otherwise `false`. The `copy()` function writes the value of `n` to the `fout` file and then calls the `more()` function to read the next integer from the `fin` file into `n`. It also returns `true` if and only if it is successful.

The first two calls to the `more()` function read 22 and 20 into `n1` and `n2`, respectively. Both calls return `true` which allows the main `while` loop to begin. On that first iteration, the condition `(n1 < n2)` is false, so the `copy()` function copies 20 from `n2` into the `combined.dat` file and then calls the `more()` function again which reads 30 into `n2`. On the second iteration, the condition `(n1 < n2)` is true (because 22 < 30), so the `copy()` function copies 22 from `n1` into the `combined.dat` file and then calls the `more()` function again which reads 25 into `n1`. The next iteration writes 25 to the output file and then reads 40 into `n1`. The next iteration writes 30 to the output file and then reads 33 into `n2`. This process continues until 85 is written to the output file from `n2` and the next call to `more()` fails, assigning `false` to `more2`. That stops the main `while` loop. Then the second `while` loop iterates three times, copying the last three integers from `north.dat` to `combined.dat` before it sets `more1` to `false`. The last loop does not iterate at all.

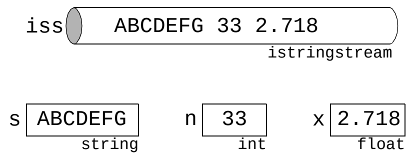
Note that file objects (`fin1`, `fin2`, `fout`) are passed to function the same way any other objects are passed. However, they must always be passed by reference.

9.6 STRING STREAMS

A *string stream* is a stream object that allows a `string` to be used as an internal text file. This is also called *in-memory I/O*. String streams are quite useful for buffering input and output. Their types `istringstream` and `ostringstream` are defined in the `<sstream>` header file.

EXAMPLE 9.8 Using an Output String Stream

This program creates four objects: a character string `s`, an integer `n`, a floating-point number `x`, and an output string stream `oss`:



```
    #include <iostream>
    #include <sstream>
    #include <string>
    using namespace std;
    void print(ostringstream&);
    int main()
    { string s="ABCDEFG";
      int n=33;
      float x=2.718;
      ostringstream oss;
```

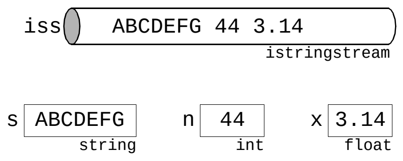
```
  print(oss);
  oss << s;
  print(oss);
  oss << " " << n;
  print(oss);
  oss << " " << x;
  print(oss);
}
void print(ostringstream& oss)
{ cout << "oss.str() = \"" << oss.str() << "\"" << endl;
}
```

```
oss.str() = ""
oss.str() = "ABCDEFG"
oss.str() = "ABCDEFG 33"
oss.str() = "ABCDEFG 33 2.718"
```

The output string stream object `oss` acts like the output stream object `cout`: the values of the string `s`, the integer `n`, and the number `x` are written to it by means of the insertion operator `<<`.

While the internal object `oss` is like an external text file, its contents can be accessed as a `string` object by the call `iss.str()`.

EXAMPLE 9.9 Using an Input String Stream



This program is similar to the one in Example 9.8 except that it reads from an input string stream `iss` instead of writing to an output string stream.:

```
void print(string&,int,float,istringstream&);
int main()
{ string s;
  int n=0;
  float x=0.0;
  istringstream iss("ABCDEFG 44 3.14");
  print(s,n,x,iss);
  iss >> s;
  print(s,n,x,iss);
  iss >> n;
  print(s,n,x,iss);
  iss >> x;
  print(s,n,x,iss);
}

void print(string& s, int n, float x, istringstream& iss)
{ cout << "s = \"" << s << "\", n = " << n << ", x = " << x
       << ", iss.str() = \"" << iss.str() << "\"" << endl;
}
```

```
s = "", n = 0, x = 0, iss.str() = "ABCDEFG 44 3.14"
s = "ABCDEFG", n = 0, x = 0, iss.str() = "ABCDEFG 44 3.14"
s = "ABCDEFG", n = 44, x = 0, iss.str() = "ABCDEFG 44 3.14"
s = "ABCDEFG", n = 44, x = 3.14, iss.str() = "ABCDEFG 44 3.14"
```

The input string stream object `iss` acts like the input stream object `cin`: values for the string `s`, the integer `n`, and the number `x` are read from it by means of the extraction operator `>>`. But the `iss` object also acts like an external file: reading from it does not change its contents.

Review Questions

9.1 What is the difference between a C-string and a C++ `string`?
9.2 What is the difference between formatted input and unformatted input?
9.3 Why can’t whitespace be read with the extraction operator?
9.4 What is a stream?
9.5 How does C++ simplify the processing of strings, external files, and internal files?
9.6 What is the difference between sequential access and direct access?
9.7 What do the `seekg()` and `seekp()` functions do?
9.8 What do the `read()` and `write()` functions do?

Problems

9.1 Describe what the following code does:

```
char cs1[] = "ABCDEFGHIJ";
char cs2[] = "ABCDEFGH";
cout << cs2 << endl;
cout << strlen(cs2) << endl;
cs2[4] = ’X’;
if (strcmp(cs1, cs2) < 0) cout << cs1 << " < " << cs2 << endl;
else cout << cs1 << " >= " << cs2 << endl;
char buffer[80];
strcpy(buffer, cs1);
strcat(buffer, cs2);
char* cs3 = strchr(buffer, ’G’);
cout << cs3 << endl;
```

9.2 Describe what the following code does:

```
string s = "ABCDEFGHIJKLMNOP";
cout << s << endl;
cout << s.length() << endl;
s[8] = ’!’;
s.replace(8, 5, "xyz");
s.erase(6, 4);
cout << s.find("!");
cout << s.find("?");
cout << s.substr(6, 3);
s += "abcde";
string part(s, 4, 8);
string stars(8, ’*’);
```

9.3 Describe what happens when the code

```
string s;
int n;
float x;
cin >> s >> n >> x >> s;
```

executes on each of the following inputs:
a. `ABC 456 7.89 XYZ`
b. `ABC 4567 .89 XYZ`
c. `ABC 456 7.8 9XYZ`
d. `ABC456 7.8 9 XYZ`
e. `ABC456 7 .89 XYZ`
f. `ABC4 56 7.89XY Z`
g. `AB C456 7.89 XYZ`
h. `AB C 456 7.89XYZ`

9.4 Trace the execution of the merge program in Example 9.7 on page 218 on the following two data files:

```
north.dat                         south.dat
27 35 38 52 55 61 81 87           31 34 41 45 49 56 63 74 92 95
```

Show each value of the variables `n1`, `n2`, `more1`, and `more2`, as they change.

9.5 Write a program that reads full names, one per line, and then prints them in the standard telephone directory format. For example, the input

```
Johann Sebastian Bach
George Frederic Handel
Carl Phillipp Emanuel Bach
Joseph Haydn
Johann Christian Bach
Wolfgang Amadeus Mozart
```

would be printed as:

```
Bach, Johann S.
Handel, George F.
Bach, Carl P. E.
Haydn, Joseph
Bach, Johann C.
Mozart, Wolfgang A.
```

9.6 Write a program that counts and prints the number of lines, words, and letter frequencies in its input. For example, the input:

```
Two roads diverged in a yellow wood,
And sorry I could not travel both
And be one traveler, long I stood
And looked down one as far as I could
To where it bent in the undergrowth;
```

would produce the output:

```
The input had 5 lines, 37 words,
and the following letter frequencies:
    A: 10   B: 3    C: 2    D: 13   E: 15   F: 1    G: 3    H: 4
    I: 7    J: 0    K: 1    L: 8    M: 0    N: 12   O: 20   P: 0
    Q: 0    R: 11   S: 5    T: 11   U: 3    V: 3    W: 6    X: 0
    Y: 2    Z: 0
```

9.7 Implement and test the following function:

```
void reduce(string& s);
// Changes all capital letters in s to lowercase
// and removes all non-letters from the beginning and end.
// EXAMPLE: if s == "’Tis,", then reduce(s) makes it "tis"
```

Hint: First write and test the following three boolean functions:

```
bool is_uppercase(char c);
bool is_lowercase(char c);
bool is_letter(char c);
```

9.8 Modify your program from Problem 9.6 so that it counts the frequencies of words instead of letters. For example, the input

```
[I] then went to Wm. and Mary college, to wit in the spring of
1760, where I continued 2 years.  It was my great good fortune,
and what probably fixed the destinies of my life that Dr. Wm.
Small of Scotland was then professor of Mathematics, a man
profound in most of the useful branches of science, with a happy
talent of communication, correct and gentlemanly manners, & an
enlarged & liberal mind.  He, most happily for me, became soon
```

```
attached to me & made me his daily companion when not engaged in
the school; and from his conversation I got my first views of the
expansion of science & of the system of things in which we are
placed.
```

would produce the output

```
The input had 11 lines and 120 words,
with the following frequencies:
              i:  3           then:  2           went:  1
             to:  3             wm:  2            and:  4
           mary:  1        college:  1            wit:  1
             in:  4            the:  6         spring:  1
             of: 11               :  6          where:  1
      continued:  1          years:  1             it:  1
            was:  2             my:  3          great:  1
           good:  1        fortune:  1           what:  1
       probably:  1          fixed:  1      destinies:  1
           life:  1           that:  1             dr:  1
          small:  1       scotland:  1      professor:  1
    mathematics:  1              a:  2            man:  1
       profound:  1           most:  2         useful:  1
       branches:  1        science:  2           with:  1
          happy:  1         talent:  1  communication:  1
        correct:  1    gentlemanly:  1        manners:  1
             an:  1       enlarged:  1        liberal:  1
           mind:  1             he:  1        happily:  1
            for:  1             me:  3         became:  1
           soon:  1       attached:  1           made:  1
            his:  2          daily:  1      companion:  1
           when:  1            not:  1        engaged:  1
         school:  1           from:  1   conversation:  1
            got:  1          first:  1          views:  1
      expansion:  1         system:  1         things:  1
          which:  1             we:  1            are:  1
         placed:  1
```

9.9 Write a program that right-justifies text. It should read and echo a sequence of left-justified lines and then print them in right-justified format. For example, the input

```
Listen, my children, and you shall hear
Of the midnight ride of Paul Revere,
On the eighteenth of April, in Seventy-five;
Hardly a man is now alive
Who remembers that famous day and year.
```

would be printed as

```
     Listen, my children, and you shall hear
        Of the midnight ride of Paul Revere,
On the eighteenth of April, in Seventy-five;
                   Hardly a man is now alive
     Who remembers that famous day and year.
```

9.10 Implement and test the following function:

```
string Roman(int n);
//  Returns the Roman numeral equivalent to the Hindu-Arabic
//  numeral n.
//  PRECONDITIONS: n > 0, n < 3888
//  EXAMPLES: Roman(1776) returns "MDCCLXXVI",
//    Roman(1812) returns "MDCCCXII", Roman(1945) returns "MCMXLV"
```

9.11 Implement and test the following function:

```
int HinduArabic(string s);
//  Returns the Hindu-Arabic numeral equivalent to the Roman
//  numeral given in the string s.
//  PRECONDITIONS: s contains a valid Roman numeral
//  EXAMPLES: HindArabic("MDCCLXXVI") returns 1776,
//    HindArabic("MDCCCXII") returns 1812
```

Note that this is the inverse of the `Roman()` function in Problem 9.10. [Hint: Write an auxiliary function `int v(string s, int i)` that returns the digit for the Roman numeral character `s[i]`; *e.g.*, `v("MDCCCXII", 1)` returns `500`.]

9.12 Implement Algorithm G.1 on page 403 to convert decimal numerals to hexadecimal:

```
string hexadecimal(int n);
//  Returns the hexadecimal numeral that represents n.
//  PRECONDITION: n >= 0
//  POSTCONDITION: each character in the returned string is a
//    hexadecimal digit and that string is the dexadecimal
//    equivalent of n
//  EXAMPLE: hexadecimal(11643) returns "2d7b"
```

[Hint: Write an auxiliary function `char c(int k)` that returns the hexadecimal character for the hexadecimal digit `k`; *e.g.*, `c(14)` returns `’e’`.]

9.13 Implement Algorithm G.2 on page 403 to convert hexadecimal numerals to decimal:

```
int decimal(string s);
//  Returns the decimal numeral that represents the hexadecimal
//  numeral stored in the string s.
//  PRECONDITION: s.length() > 0 and each s[i] is a hexadecimal
//    digit
//  POSTCONDITION: the returns value is the decimal equivalent
//  EXAMPLE: decimal("2d7b") returns 11643
```

Note that this is the inverse of the `hexadecimal()` function in Problem 9.12. [Hint: Write an auxiliary function `int v(string s, int i)` that returns the decimal digit for the hexadecimal character `s[i]`; *e.g.*, `v("2d7b", 3)` returns `12`.]

9.14 Implement and test the following function:

```
void reverse(string& s);
//  Reverses the string s.
//  POSTCONDITION: s[i] <--> s[len-i-1]
//  EXAMPLE: reverse(s) changes s = "ABCDEFG" into "GFEDCBA"
```

[Hint: Use a temporary string.]

9.15 Implement and test the following function:

```
bool is_palindrome(string s);
//  Returns true iff s is a palindrome
//  EXAMPLES: is_palindrome("RADAR") returns true,
//    is_palindrome("ABCD") returns false
```

9.16 Modify the program in Example 9.7 on page 218 so that it merges the two sorted files of names shown at the top of the next page, writing the resulting sorted lines both to a file named `Presidents.dat` and to `cout`:

[Hint: Use `getline(fin, s)`.]