EXAMPLE: DRY BONES!

The Old Testament book of Ezekiel is a book of vivid images that chronicle the siege of Jerusalem by the Babylonians and the subsequent forced relocation (known as the *exile*) of the Israelites following Jerusalem's fall. Chapter 37 describes a powerful vision of Ezekiel in which a valley of dry bones becomes reconnected as he prophesies. Tendons, muscle, and skin regrow on the reconnected bones and the dead skeletons return to life forming a vast army that fills the valley. This vision of new life arising from dry bones provided the homesick Israelites with hope that the" dry bones" of defeated Israel would "come back to life" and they could one day be free of their oppressors, the Babylonians.

Before the American Civil War, many black men and women in the United States found themselves in a situation similar to that of the Israelites, having been forcibly relocated from their homelands and brought to the United States to serve as slaves. One way that they were able to keep their hopes alive was through singing songs that spoke of freedom. However, songs that expressed such hopes directly could result in a beating for the singer, and so these sentiments had to be cleverly encoded in lyrics that would not catch the attention of the slaves' overseers.

One way this was done was by singing **spirituals**—songs whose lyrics referred to stories from the Bible that carried themes of hope and freedom. Songs like *Go Down Moses; Swing Low, Sweet Chariot; Wade in the Water;* and *Bright Canaan* are examples of spirituals that, in addition to their overt spiritual message, carried a second level of meaning that provided these men and women with the hope that they could one day be free of oppression.

One of the most subtle spirituals is *Dry Bones*. It makes no direct references to freedom, but uses the imagery of scattered bones being reconnected to encode the same message of hope conveyed in Ezekiel 37. With its message of freedom carefully hidden, such a song could be sung in the presence of even the harshest overseer:

```
Ezekiel cried, "Dem dry bones!"
Ezekiel cried, "Dem dry bones!"
Ezekiel cried, "Dem dry bones!"
Oh, hear the word of the Lord.
The foot bone connected to the leg bone,
The leg bone connected to the knee bone,
The knee bone connected to the thigh bone,
The thigh bone connected to the back bone,
The back bone connected to the neck bone,
The neck bone connected to the head bone,
Oh, hear the word of the Lord!
Dem bones, dem bones gonna walk aroun'
Dem bones, dem bones, gonna walk aroun'
Dem bones, dem bones, gonna walk aroun'
Oh, hear the word of the Lord.
The head bone connected to the neck bone,
The neck bone connected to the back bone,
The back bone connected to the thigh bone,
```

```
The thigh bone connected to the knee bone,
The knee bone connected to the leg bone,
The leg bone connected to the foot bone,
Oh, hear the word of the Lord!
Dem bones, dem bones gonna walk aroun'
Dem bones, dem bones, gonna walk aroun'
Dem bones, dem bones, gonna walk aroun'
Oh, hear the word of the Lord.
```

The structure of the song is interesting, because it can be partitioned into the following steps, which together comprise an algorithm for printing the song:

a. Print the `"Ezekiel cried"` variation of the chorus.
b. Print the *bone lyrics* from foot to head.
c. Print the `"Dem bones"` variation of the chorus.
d. Print the *bone lyrics* from head to foot.
e. Print the `"Dem bones"` variation of the chorus.

What makes the structure interesting is that while steps b and d are similar, they proceed in reverse order; that is, with the bones reversed (i.e., in LIFO order). So in step b, the lines have the form

```
The X bone connected to the Y bone,
```

while in step d, the corresponding lines are in reverse order, with the bones reversed:

```
The Y bone connected to the X bone,
```

Because of this reversal, the actions performed in steps b-d can be described using recursion. More precisely, step c occurs in the middle, and so provides us with an anchor case:

`X ==  "head"` → Print the `"Dem bones"` variation of the chorus

and every other case can be performed recursively, using this strategy :

1. Identify the next bone `Y`.
2. Print `"The X bone connected to the Y bone,"`.
3. Recursively print the lyrics for bone `Y`.
4. Print `"The Y bone connected to the X bone,"`.

The function `printBoneLyrics()` in Code15-1 encodes this recursive logic. The output statement

```
cout << "The " << bone             //  do 'downward' lyric
     << " bone connected to the "  //  before recursion
     << nextBone << " bone,\n";    //  (winding)
```

generates the lyrics for the current bone, and the function call

```
   printBoneLyrics(nextBone);         //  do rest recursively
```

then recursively does the same for each of the bones "below" the current bone. Once the anchor case has been reached, the recursion then unwinds, executing the statement

```
   cout << "\nThe " << nextBone       //  do 'upward' lyric
        << " bone connected to the "  //  after recursion
        << bone << " bone,";          //  (unwinding)
```

which generates the lyrics in the reverse order, with the bones reversed.

The first call to `printChorus()` in `main()` performs step a of the algorithm; the recursive function `printBoneLyrics()` performs steps b–d; and the second call to `printChorus()` performs the final step e.

Code 15-1 Dry Bones!

```
/* dryBones.cpp displays the lyrics of the song "Dry Bones."
 *
 * Output: lyrics of "Dry Bones"
 ************************************************************/

#include <iostream>                        // cin, cout, <<, >>
#include <string>                          // string class
using namespace std;

void printChorus(const string& variation);
void printLastLine();
void printBoneLyrics(const string & bone);

int main()
{
   printChorus("Ezekiel cried, \"Dem dry bones!\"\n");
   printBoneLyrics("foot");
   printLastLine();
   printChorus("Dem bones, dem bones gonna walk aroun'\n");
}

/* printChorus displays the lyrics of the chorus of the song, of
 * which there are two variations.
 *
 * Receive: variation, a string
 * Output:  the chorus with the specified variation.
 ************************************************************/

void printChorus(const string& variation)
{
   cout << variation << variation << variation;
   printLastLine();
}
```

```
/* printLastLine displays the last line of each verse and the chorus.
 *
 * Output:  the last line
 ****************************************************************/

void printLastLine()
{
   cout << "Oh, hear the word of the Lord!\n\n";
}

/* printBoneLyrics displays the lyrics for a given bone.
 *
 * Receive: bone, a string.
 * Output:  the lyrics for that bone and (recursively) the lyrics
 *          for the bones "beneath" it (during winding) and then
 *          for the bones "above" it (during unwinding).
 ****************************************************************/

string getNext(const string& aBone);

void printBoneLyrics(const string& bone)
{
   if (bone == "head")                  // Anchor: bone == head
                                        //  sing chorus variation
      printChorus("Dem bones, dem bones gonna walk aroun'\n");

   else
   {                                    // Ind-Step: bone < head
      string nextBone = getNext(bone);  //  find next body part

      cout << "The " << bone            //  do 'upward' lyric
           << " bone connected to the "  //  before recursion
           << nextBone << " bone,\n";    //  (winding)

      printBoneLyrics(nextBone);        //  do rest recursively

      cout << "The " << nextBone        //  do 'downward' lyric
           << " bone connected to the "  //  after recursion
           << bone << " bone,\n";        //  (unwinding)
   }
}

/* getNext() gets the next bone.
 *
 * Receive:      aBone, a string.
 * Precondition: aBone is a valid bone (in the song).
 * Return:       the bone above aBone.
 ****************************************************************/

string getNext(const string& aBone)
{
   if (aBone == "foot")
```

```
      return "leg";
   else if (aBone == "leg")
      return "knee";
   else if (aBone == "knee")
      return "thigh";
   else if (aBone == "thigh")
      return "back";
   else if (aBone == "back")
      return "neck";
   else if (aBone == "neck")
      return "head";
   else
      cerr << "\n*** GetNext(): "
           << aBone << " is unknown!" << endl;
   return "";
}
```

Like the Towers of Hanoi example in Section 8.6, this example demonstrates that recursion is not limited to functions such as `power()` that return numerical values. Instead, recursion can be applied to solve any of the wide variety of problems whose solutions are inherently recursive. In the next section, we see how recursion can actually simplify the operations on certain kind of containers.