12.4 Case Study: Geological Classification

Objects are often organized into groups that share similar characteristics. To illustrate:

- A doctor might be described as an internist, a pediatrician, a surgeon, a gynecologist, a family practitioner, or some other specialty, according to his or her area of training.
- Members of the animal kingdom are organized into groups (called phyla) according to whether or not they have vertebrae, the relative positions of their nervous and digestive systems, and their outer covering.
- The elements are organized into groups according to the number of electrons in the outermost shell of one of their atoms.

These are just a few of the many situations in which we **classify** objects. By classifying objects into groups according to their characteristics, objects that are in some way similar become related by their group membership.

The library developed in this section illustrates how enumerations can be used to create software simulations of classifications from the real world. The problem we will consider is a simplified rock-classification problem.

In geology, rocks are classified according to the nature of their origin. More precisely, a given rock is described as:

- *Igneous*, if it is volcanic in origin (i.e., formed as the result of solidifying magma);
- *Metamorphic*, if the rock was formed under conditions of high temperature and pressure; and
- *Sedimentary*, if the rock was formed from the laying down of deposits of sediment.

Igneous rocks include basalt, granite, and obsidian. Metamorphic rocks include marble, quartzite, and slate. Sedimentary rocks include dolomite, limestone, sandstone, and shale.

Knowing the different categories of rocks can make outdoor activities (such as backpacking and canoeing) more interesting. For example, if one is hiking through a valley whose walls contain layers of sandstone, then one can conclude that the walls of the valley were probably once under water and may contain fossils of water creatures. By contrast, a valley whose walls consist of granite means that there was once a volcano in the vicinity, and finding it can make for an interesting diversion in the hike.

It is relatively easy to write a program that, given the name of a rock, describes some of its characteristics. The following is an example of such a program.

Case Study 12.4-1 A Rock Classification Program.

```
// ----- Figure 12-16 in the text.

/* geology.cpp allows a user to retrieve information about rocks.
 *
 * Receive: the name of a rock
 * Output:  the known information about that rock
 ****************************************************************/

#include <iostream>                       // cin, cout
using namespace std;
#include "Rock.h"                         // class Rock

int main()
{
   cout << "This program provides information about "
           "specific rocks.\n";

   Rock aRock;                            // input variable
   char response;                         // query response

   do
   {
      cout << "\nEnter the name of a rock: ";
      cin >> aRock;

      if ( cin.good() )
         cout << endl << aRock
              << " is classified as a(n) " << aRock.kind()
              << " rock, and\n its texture is " << aRock.texture()
              << endl;
      else
      {
         cerr << "That kind of rock is not supported." << endl;
         cin.clear();
      }

      cout << "\nEnter 'c' to continue, anything else to quit: ";
      cin >> response;
   }
   while (response == 'c');
}
```

Sample run:

```
This program provides information about specific rocks.

Enter the name of a rock: sandstone

Sandstone is classified as a(n) SEDIMENTARYrock, and
 its texture is COARSE.
```

```
Enter 'c' to continue, anything else to quit: c

Enter the name of a rock: obsidian

Obsidian is classified as a(n) IGNEOUS rock, and
 its texture is FINE.

Enter 'c' to continue, anything else to quit: q
```

The interesting thing about the preceding program is that it allows the user to communicate in real-world terms (the names of rocks), rather than through some artificial mechanism (such as a numeric representation of rocks). This is accomplished by using the techniques of the last section to create the type `Rock` as a class that uses an enumeration `RockName` whose values are the names of common rocks,

Because it might be useful for other geological programs to be able to use the class `Rock`, it should be stored in a library to facilitate its reuse. The following is a header file for such a library:

Case Study 12.4-2 Class `Rock` Header File.

```
// ----- Figure 12-17 in the text.

/* Rock.h provides the declaration of class Rock.
 *
 * Class Invariant:
 *    The value of myRockName is one of the following:
 *       BASALT, DOLOMITE, GRANITE, LIMESTONE, MARBLE,
 *       OBSIDIAN, QUARTZITE, SANDSTONE, SHALE, SLATE
 ************************************************************/

#ifndef ROCK
#define ROCK

#include <iostream>                          // istream, ostream
#include <cassert>                           // assert
using namespace std;
#include "RockKind.h"                        // class RockKind
#include "RockTexture.h"                     // class RockTexture

enum RockName {ROCK_UNDERFLOW = -1,
               BASALT, DOLOMITE, GRANITE,          // recognized
               LIMESTONE, MARBLE, OBSIDIAN,        //  names of
               QUARTZITE, SANDSTONE, SHALE, SLATE, //  rocks
               NUMBER_OF_ROCKS,
               ROCK_OVERFLOW = NUMBER_OF_ROCKS};
```

```
class Rock
{
 public:                                          // constructors:
   Rock();                                        //  default value
   Rock(RockName initialRock);                    //  explicit value

   void read(istream & in);                       // input
   void print(ostream & out) const;               // output

   RockKind kind() const;                         // igneous, ...
   RockTexture texture() const;                   // coarse, ...

   Rock operator++();                             // prefix ++
   Rock operator++(int);                          // postfix ++
   Rock operator--();                             // prefix --
   Rock operator--(int);                          // postfix --

                                                  // relationals
   friend bool operator== (const Rock& left, const Rock& right);
   friend bool operator!= (const Rock& left, const Rock& right);
   friend bool operator<  (const Rock& left, const Rock& right);
   friend bool operator>  (const Rock& left, const Rock& right);
   friend bool operator<= (const Rock& left, const Rock& right);
   friend bool operator>= (const Rock& left, const Rock& right);

 private:
   RockName myRockName;
};


// -------- Function input -------------------------------------
inline istream& operator>>(istream& in, Rock& theRock)
{
   theRock.read(in);
   return in;
}


// -------- Function output ------------------------------------
inline ostream& operator<<(ostream& out, const Rock& theRock)
{
   theRock.print(out);
   return out;
}


// -------- Initialize me (default value) ----------------------
inline Rock::Rock()
{
   myRockName = BASALT;
}
```

```
// -------- Initialize me (explicit-value) ------------------------
inline Rock::Rock(RockName initialRock)
{
   assert(BASALT <= initialRock && initialRock < NUMBER_OF_ROCKS);
   myRockName = initialRock;
}


//--------- Compare me and anotherRock using equality -------------
inline bool operator==(const Rock& left, const Rock& right)
{
   return left.myRockName == right.myRockName;
}


// ---  Inline definitions for operators !=, <, >, <=, and >= are
// ---  similar and are omitted here to save space.

#endif
```

Note that this class depends upon the classes `RockKind` and `RockTexture`, each of which has its own header file. Since there are three real-world kinds of rocks, a `RockKind` class can be created with an enumeration of its own, whose values are the various kinds of rocks as shown below.

Case Study 12.4-3 Class `RockKind` Header File.

```
// ----- Figure 12-18 in the text.

/* RockKind.h declares class RockKind.
 *
 *    The value of myRockKindName is one of the following:
 *       IGNEOUS, METAMORPHIC, SEDIMENTARY,
 ************************************************************/

#ifndef ROCK_KIND
#define ROCK_KIND

enum RockKindName {KIND_UNDERFLOW = -1,
                   IGNEOUS, METAMORPHIC, SEDIMENTARY,
                   NUMBER_OF_KINDS,
                   KIND_OVERFLOW = NUMBER_OF_KINDS};

#include <iostream>                    // istream, ostream
using namespace std;
```

```
class RockKind
{
 public:                                // constructors:
   RockKind();                          //  default value
   RockKind(RockKindName initialRock); //  explicit value

   void read(istream& in);              // input
   void print(ostream& out) const;      // output

   RockKind operator++();               // prefix ++
   RockKind operator++(int);            // postfix ++
   RockKind operator--();               // prefix --
   RockKind operator--(int);            // postfix --

   friend bool operator==(const RockKind& left, const RockKind& right);
   friend bool operator!=(const RockKind& left, const RockKind& right);
   friend bool operator<(const RockKind& left, const RockKind& right);
   friend bool operator>(const RockKind& left, const RockKind& right);
   friend bool operator<=(const RockKind& left, const RockKind& right);
   friend bool operator>=(const RockKind& left, const RockKind& right);

 private:
   RockKindName myRockKindName;
};


// -------- Non-member input -----------------------------------
inline istream& operator>>(istream& in, RockKind& theRockKind)
{
   theRockKind.read(in);
   return in;
}


// -------- Non-member output ----------------------------------
inline ostream& operator<<(ostream& out, const RockKind& theRockKind)
{
   theRockKind.print(out);
   return out;

}


// -------- Initialize me (default-value) ------------------------
inline RockKind::RockKind()
{
   myRockKindName = IGNEOUS;
}

// -------- Initialize me (explicit-value) -----------------------
#include <cassert>
```

```
using namespace std;
inline RockKind::RockKind(RockKindName initialRockKindName)
{
   assert(IGNEOUS <= initialRockKindName &&
           initialRockKindName < NUMBER_OF_KINDS);
   myRockKindName = initialRockKindName;
}

// --------- Compare me and anotherRockKind using == ---------------
inline bool operator==(const RockKind& left, const RockKind& right)
{
   return left.myRockKindName == right.myRockKindName;
}

// ---  Inline definitions for operators !=, <, >, <=, and >= are
// ---  essentially the same and are omitted here to save space.

#endif
```

A class `RockTexture` can be created using a similar `RockTextureName` enumeration and is left as an exercise.

The implementation file for class `RockKind` contains the definitions of the nontrivial `RockKind` function members. The following is part of this file.

Case Study 12.4-4 Class `RockKind` Implementation File.

```
// ----- Figure 12-19 in the text.

/* RockKind.cpp defines the nontrivial members of class RockKind.
 ***************************************************************/

#include <string>
#include <cctype>
using namespace std;

#include "RockKind.h"

// -------- Read a value into me from in --------------------------
void RockKind::read(istream& in)
{
   string rockKindString;
   in >> rockKindString;

   for (int i = 0; i < rockKindString.size(); i++)
      if (islower(rockKindString[i]))
         rockKindString[i] = toupper(rockKindString[i]);

   if (rockKindString== "IGNEOUS")
      myRockKindName = IGNEOUS;
   else if (rockKindString== "METAMORPHIC")
```

```
      myRockKindName = METAMORPHIC;
   else if (rockKindString== "SEDIMENTARY")
      myRockKindName = SEDIMENTARY;
   else
   {
      cerr << "\n*** Read: Rock kind is unknown\n"
           << endl;
      in.setstate(ios::failbit);
   }
}

// -------- Display me via out -------------------------
void RockKind::print(ostream& out) const
{
   switch(myRockKindName)
   {
      case IGNEOUS:
                        out << "IGNEOUS";
                        break;
      case METAMORPHIC:
                        out << "METAMORPHIC";
                        break;
      case SEDIMENTARY:
                        out << "SEDIMENTARY";
                        break;
   }
}


// -------- Increment me (prefix) --------------------------
RockKind RockKind::operator++()
{
   switch(myRockKindName)
   {
      case IGNEOUS:
                        myRockKindName = METAMORPHIC;
                        break;
      case METAMORPHIC:
                        myRockKindName = SEDIMENTARY;
                        break;
      case SEDIMENTARY:
                        myRockKindName = KIND_OVERFLOW;
                        break;
   }
   return RockKind(myRockKindName);
}

// -------- Increment me (postfix) -------------------------
RockKind RockKind::operator++(int)
{
   RockKindName savedRockKindName = myRockKindName;
   switch(myRockKindName)
```

```
   {
      case IGNEOUS:
                        myRockKindName = METAMORPHIC;
                        break;
      case METAMORPHIC:
                        myRockKindName = SEDIMENTARY;
                        break;
      case SEDIMENTARY:
                        myRockKindName = KIND_OVERFLOW;
                        break;
    }
   return RockKind(savedRockKindName);
}

// -------- Decrement me (prefix) --------------------------------
// -------- Decrement me (postfix) -------------------------------

// Definitions of operator--() and operator--(int) are similar to those
// of operator++() and operator++(int) and are omitted to save space.
```

An implementation file for class `RockTexture` is similar, and left as an exercise.

The following is part of the implementation file *Rock.cpp* containing definitions of several of the nontrival `Rock` function members.

Case Study 12.4-5 Class `Rock` Implementation File.

```
// ----- Figure 12-20 in the text.

/* Rock.cpp defines the nontrivial members of class Rock.
 ************************************************************/

#include "Rock.h"

// ------- Read a value into me from in -------------------------
#include <cctype>
#include <string>
using namespace std;

void Rock::read(istream& in)
{
   string rockString;
   in >> rockString;

   for (int i = 0; i < rockString.size(); i++)
      if (islower(rockString[i]))
         rockString[i] = toupper(rockString[i]);

   if (rockString == "BASALT")
```

```
      myRockName = BASALT;
   else if (rockString == "DOLOMITE")
      myRockName = DOLOMITE;
                  .
                  .
                  .
      myRockName = DOLOMITE;
                else if (rockString == "SLATE")
      myRockName = SLATE;
   else
   {
      cerr << "\n*** Rock::read(): " << rockString
           << " is not recognized!\n" << endl;
      in.setstate(ios::failbit);
   }
}


// -------- Display my value via out -----------------------------
void Rock::print(ostream& out) const
{
   switch(myRockName)
   {
      case BASALT:    out << "BASALT";
                      break;
      case DOLOMITE:  out << "DOLOMITE";
                      break;
                           .
                           .
                           .
      case SLATE:     out << "SLATE";
                      break;
   }
}


// -------- My kind (igneous, metamorphic, ...) ---------------
RockKind Rock::kind() const
{
   switch (myRockName)              // if the rock is...
   {
      case BASALT: case GRANITE:    //   any of these, then
      case OBSIDIAN:                //      its an igneous rock
         return RockKind(IGNEOUS);
      case MARBLE: case QUARTZITE:  //   any of these, then
      case SLATE:                   //     it's a metamorphic rock
         return RockKind(METAMORPHIC);
      case DOLOMITE: case LIMESTONE: //   any of these, then
      case SANDSTONE: case SHALE:    //     it's a sedimentary rock
         return RockKind(SEDIMENTARY);
   }
}
```

```
// -------- My texture (coarse, fine, ...) ------------------
RockTexture Rock::texture() const
{
   // ... See Exercises ...
}


// -------- Increment my value (prefix) ----------------------
Rock Rock::operator++()
{
   switch(myRockName)
   {
      case BASALT:
                     myRockName = DOLOMITE;
                     break;
      case DOLOMITE:
                     myRockName = GRANITE;
                     break;
                           .
                           .
                           .
      case SLATE:
                     myRockName = ROCK_OVERFLOW;
   }
   return Rock(myRockName);
}


// -------- Increment my value (postfix) ---------------------
Rock Rock::operator++(int)
{
   RockName savedRockName = myRockName;

   switch(myRockName)
   {
      case BASALT:
                     myRockName = DOLOMITE;
                     break;
      case DOLOMITE:
                     myRockName = GRANITE;
                     break;
                           .
                           .
                           .
      case SLATE:
                     myRockName = ROCK_OVERFLOW;
   }

   return Rock(savedRockName);
}

// -------- Decrement me (prefix) -------------------------------
```

```
// -------- Decrement me (postfix) -----------------------------

// Definitions of operator--() and operator--(int) are similar to those
// of operator++() and operator++(int) and are omitted to save space.
```

EXERCISES 12.4

1. Implement the class `RockTexture`, that supports the use of the three real-world rock textures: `COARSE`, `INTERMEDIATE`, and `FINE`.
2. Implement the `Rock` method `texture()`, assuming the following:

   granite, sandstone, dolomite, and limestone are coarse in texture

   basalt, shale, slate, and quartzite are intermediate in texture

   obsidian and marble are fine in texture.