Solutions to Exercises in Chapter 5

5.1 In principle, the solution to the 'millenium bug' was simply to change the date representation to allow *four* digits for the year number.

In practice, the problem was enormous. In thousands of programs totalling billions of lines of code, it was necessary to locate and modify every declaration defining the date representation, and every statement whose effect depended on the date representation. Thousands of files and databases containing date values had to be translated to the new date representation. The problem was compounded by the fact that many of the programs were ill-structured and undocumented.

5.2 A possible `Person` ADT is shown in Program S5.1. This assumes that a person's forename, gender, and year of birth can never be changed.

5.4 A possible extended contract for a `Date` ADT is shown in Program S5.2.

5.5 Array and SLL implementations of the `String` ADT are outlined in Programs S5.3 and S5.4, respectively. Program S5.4 assumes a class `CharNode` that is similar to `SLLNode` (Program 4.4) except that the element is of type **`char`**.

Each operation's time complexity corresponds to Table 5.14.

Figure S5.5 illustrates the effect of the `concat` operation of Program S5.4. The operation makes a copy of **`this`**'s SLL, then links the resulting SLL's last node to `that`'s first node. Thus the resulting SLL shares nodes with `that`'s SLL, rather than containing copies of these nodes. The operation makes only n_1 copies, so its time requirement does not depend on n_2. (*Note:* This implementation of `concat` is possible only because `String` objects are immutable.)

5.6 A possible contract for a `Text` ADT is shown in Program S5.6.

(*Note:* Cut would be achieved by `clipboard = text.copy(start, finish)` followed by `text.delete(start, finish)`. Paste would be achieved by `text.insert(cursor, clipboard)`.)

The simplest representation for a text would be a long array or SLL of characters, where each line-terminator is represented by a suitable control character (such as CR or LF).

5.7 A possible contract for a `TimeOfDay` ADT is shown in Program S5.7.

A time-of-day can be represented by a triple (hours, minutes, seconds), or by the number of seconds since midnight. The latter representation makes the `difference` and `plus` operations easier to implement.

5.9 A possible contract for a `Complex` ADT is outlined in Figure S5.8.

A complex number can be represented either by a pair of Cartesian coordinates (*x*, *y*) or by a pair of polar coordinates (*r*, *?*). Cartesian coordinates make it easier to implement the `plus` and `minus` operations; polar coordinates make it easier to implement the `times` and `over` operations.

5.10 A possible `Course` ADT is shown in Program S5.9. This assumes that the course-code is fixed, but all other details can be changed.

```
public class Person {

   // Each Person value consists of a person's surname, forename, gender,
   // and year of birth.

   private String surname, forename;
   private boolean female;
   private int yearOfBirth;

   /////////////  Constructor  /////////////

   public Person (String surname, String forename,
                  boolean female, int yearOfBirth) { … }

   /////////////  Accessors  /////////////

   public String getSurname ()   { return this.surname; }

   public String getForename ()  { return this.forename; }

   public boolean isFemale ()    { return this.female; }

   public int getYearOfBirth ()  { return this.yearOfBirth; }

   /////////////  Transformer  /////////////

   public void changeName (String newSurname) {
      this.surname = newSurname;
   }

}
```

Program S5.1 Implementation of a `Person` ADT (changes from Program 5.2 italicized).

```
public class Date {

   // Each Date value is a past, present, or future date.

   private …;

   /////////////  Constructor  /////////////

   public Date (int y, int m, int d);
   // Construct a date with year y, month m, and day-in-month d.

   /////////////  Accessors  /////////////

   public int getYear ();
   // Return this date's year number.

   public int getMonth ();
   // Return this date's month number.

   public int getDay ();
   // Return this date's day-in-month number.

   public int difference (Date that);
   // Return the number of days between this date and that.

   /////////////  Transformer  /////////////

   public void advance (int n);
   // Advance this date by n days (or retard this date if n < 0).

}
```

Program S5.2 An extended contract for the `Date` ADT (changes from Program 5.10 italicized).

```
public class String {

   // Each String value is an immutable string of characters, of any length,
   // with consecutive indices starting at 0.

   private char[] chars;

   /////////////  Constructor  /////////////

   public String (char[] cs) {
      chars = cs;
   }

   /////////////  Accessors  /////////////

   public int length () {
      return chars.length;
   }

   public char charAt (int i) { … }

   public bool equals (String that) { … }

   public int compareTo (String that) {
      int minLength =
            (this.chars.length <= that.chars.length ?
               this.chars.length : that.chars.length);
      for (int i = 0; i < minLength; i++) {
         int comp = this.chars[i] - that.chars[i];
         if (comp != 0)  return comp;
      }
      return (this.chars.length - that.chars.length);
   }

   /////////////  Transformers  /////////////

   public String substring (int i, int j) { … }

   public String concat (String that) {
      char allChars[] = new char[this.chars.length
            + that.chars.length];
      int i = 0, j = 0, k = 0;
      while (i < this.chars.length)
         allChars[k++] = this.chars[i++];
      while (j < that.chars.length)
         allChars[k++] = that.chars[j++];
      return new String(allChars);
   }

}
```

Program S5.3 Implementation of the `String` ADT using arrays (outline).

```
public class String {

   // Each String value is an immutable string of characters, of any length,
   // with consecutive indices starting at 0.

   private CharNode first;
   private int length;

   ////////////// Constructors //////////////

   public String (char[] cs) {
      length = cs.length;
      first = null;
      for (int i = cs.length-1; i >= 0; i--)
         first = new SLLNode(cs[i], first);
   }

   private String () {
      first = null;  length = 0;
   }

   ////////////// Accessors //////////////

   public int length () { … }

   public char charAt (int i) { … }

   public bool equals (String that) { … }

   public int compareTo (String that) {
      for (SLLNode curr1 = first, curr2 = that.first;
           curr1 != null && curr2 != null;
           curr1 = curr1.succ, curr2 = curr2.succ) {
         int comp = curr1.element - curr2.element;
         if (comp != 0)  return comp;
      }
      return (this.length - that.length);
   }

   ////////////// Transformers //////////////

   public String substring (int i, int j) { … }

   public String concat (String that) {
      String all = new String();
      all.length = this.length + that.length;
      if (this.first == null)
         all.first = that.first;
      else {
         CharNode last =
              new CharNode(this.first.element, null);
         all.first = last;
         for (CharNode curr = this.first.succ;
              curr != null; curr = curr.succ) {
            last.succ = new CharNode(curr.element, null);
            last = last.succ;
         }
         last.succ = that.first;
      }
   }

}
```

Program S5.4 Implementation of the `String` ADT using SLLs (outline).

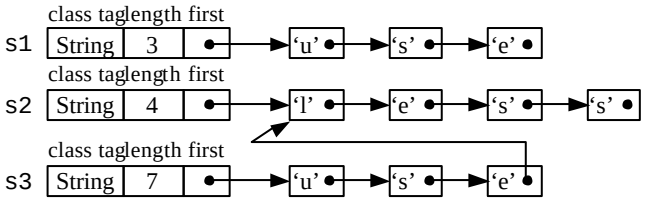

Figure S5.5 Effect of `s3 = s1.concat(s2)` when `String` objects are represented by SLLs.

```
public class Text {

   // Each Text value is a sequence of characters subdivided into lines. The characters
   // and line-terminators have consecutive position numbers, starting at 0.

   private …;

   /////////////  Constructors  /////////////

   public Text ();
   // Construct an empty text.

   /////////////  Accessors  /////////////

   public void display ();
   // Display this text on the screen.

   public void save (String filename);
   // Save this text to the named file.

   /////////////  Transformers  /////////////

   public void load (String filename);
   // Load the contents of the named file into this text.

   public void insert (int p, char c);
   // Insert c just before the character at position p in this text.

   public void insert (int p, Text that);
   // Insert that text just before the character at position p in this text.

   public void delete (int p);
   // Delete the character at position p in this text.

   public void delete (int p1, int p2);
   // Delete all characters at positions p1 through p2 in this text.

   public Text copy (int p1, int p2);
   // Return a copy of all characters at positions p1 through p2 in this text.

}
```

Program S5.6 Contract for a `Text` ADT.

```
public class TimeOfDay {

   // Each TimeOfDay value is a time-of-day, accurate to 1 second.

   private …;

   /////////////  Constructors  /////////////

   public TimeOfDay (int s);
   // Construct a time-of-day that is s seconds since midnight.

   public TimeOfDay (int h, int m, int s);
   // Construct a time-of-day that is h hours, m minutes, and s seconds since midnight.

   /////////////  Accessors  /////////////

   public int getHour ();
   // Return the hours part of this time-of-day.

   public int getMinute ();
   // Return the minutes part of this time-of-day.

   public int getSecond ();
   // Return the seconds part of this time-of-day.

   public int difference (TimeOfDay that);
   // Return the number of seconds between this time-of-day and that.

   /////////////  Transformer  /////////////

   public TimeOfDay plus (int s);
   // Return this time-of-day advanced by s seconds (or retarded if s < 0).

}
```

Program S5.7 Contract for a `TimeOfDay` ADT.

```
public class Complex {

   // Each Complex value is a complex number.

   private …;

   /////////////  Constructors  /////////////

   public Complex (float x, float y);
   // Construct a complex number with real part x and imaginary part y.

   /////////////  Accessors  /////////////

   public float getRealPart ();
   // Return the real part of this complex number.

   public float getImagPart ();
   // Return the imaginary part of this complex number.

   …

   /////////////  Transformers  /////////////

   public Complex plus (Complex that);
   // Return the sum of this complex number and that.

   public Complex minus (Complex that);
   // Return the difference of this complex number and that.

   public Complex times (Complex that);
   // Return the product of this complex number and that.

   public Complex over (Complex that);
   // Return the quotient of this complex number and that.

   …

}
```

Program S5.8 Contract for a `Complex` ADT (outline).

```
public class Course {

   // Each Course value is a course description, consisting of a course-code, title,
   // names one or more instructors, and names of zero or more teaching assistants.

   private …;

   /////////////  Constructor  ////////////

   public Course (String code, String title,
                  String[] instructors);
   // Construct a course description with the given details but no teaching assistants.

   /////////////  Accessors  ////////////

   public String getCode ();
   // Return this course's course-code.

   public String getTitle ();
   // Return this course's title.

   public String[] getInstructors ();
   // Return the names of this course's instructors.

   public String[] getAssistants ();
   // Return the names of this course's teaching asistants.

   /////////////  Transformers  ////////////

   public void changeTitle (String newTitle);
   // Change this course's title to newTitle.

   public void addInstructor (String name);
   // Add an instructor with name to this course.

   public void removeInstructor (String name);
   // Remove the instructor with name from this course.

   public void addAssistant (String name);
   // Add an assistant with name to this course.

   public void removeAssistant (String name);
   // Remove the assistant with name from this course.

}
```

Program S5.9 Contract for a `Course` ADT.