Laboratory 6

Problem Solving

Objective

- Gain experience with Polya's problem-solving methodology.
- Study algorithm development using top-down design and object-oriented design.

References

No software needed (unless you want to use a word processor for writing).

Textbook reference: Chapter 6, pp. 152–186

Background

You should read Chapter 6, "Problem Solving and Algorithm Design," thoroughly and study the examples of algorithm design it contains.

Activity

Algorithm design using reliable and disciplined methodologies was not done in the early days of computing. Programmers learned from each other, from studying someone else's code, and just by sheer intelligence and fortitude. But in the 1960s, when software projects started to balloon to millions of lines of code, this lack of clear methods prompted people to invent methodologies. Top-down design was one of the earliest. Object-oriented design became prevalent by the late 1980s and early 1990s with widespread use of C++ and other object-oriented languages.

However, designing solutions to problems using computers has always remained difficult. Because computers are used to solve such a wide variety of problems, there isn't one perfect recipe for cooking up a suitable program. However, there are some guidelines and strategies, which are discussed thoroughly in Chapter 6 of your textbook.

This lab is an exercise in thinking, discussing, and writing. No computer software is involved, unless you care to write using a word processor. Follow the directions and consult your textbook frequently. It might be good to work with another person, or even with a small group, because different people read different things in the problem and thus allow you to gain a wider perspective.

Exercise 1

Name ______________________________ Date ____________________

Section ______________________________

1) Suppose that you have been asked to write an algorithm, which will ultimately be implemented on a computer, to plan a vacation for a family traveling by car. The family wants to visit a number of cities in order to see relatives, to see historical attractions, and to have lots of fun at an amusement park. Write down all the items of information that the computer will need. (Hint: Think about money, time, and routes.)

2) What would a solution look like? What should the computer deliver as output to the family?

3) Is there one perfect solution, or might the family like to choose from several solutions? What would the trade-offs between different solutions likely be?

4) Let's now apply top-down design to this problem. Here's the top-level main module:

Ask for a starting city and destination. Then find routes between them with associated costs. Choose the best routes.

Decompose this by attaching some boxes underneath it that give the main phases of solving the algorithm.

5) One of the difficulties of translating a top-down design written with charts or pseudo-code into a computer program is recognizing which of the actions you list can be done directly by the computer, and which need further refining and breaking down into more primitive actions. Which of the actions in the list below sound like they might be "doable" by present-day computers?

- Find a highway route from Philadelphia to New York.
- Add the hotel cost to the running total of how much money has been spent.
- Ask the family what city they are starting their trip from.
- Order tickets for the amusement park.
- Put gas into the car's tank.
- Print a list of cities in the order in which they should be visited.
- Book a hotel room in New York by e-mail.
- Sort the list of cities by priority number.
- Decide whether more historical attractions than amusement parks should be visited.
- Look up the distance from city A to city B.

6) Suppose you have a distance table that gives the distance between any two cities, such as the one shown below. If there is no number in the intersection, the cities are not directly connected by a road. (All cities and distances are purely fictitious.)

	Alcrombie	Balooska	Chimichanga	Del Roy Point	Eberley
Alcrombie		50	87	20	
Balooska	50			39	77
Chimichanga	87			12	
Del Roy Point	20	39	12		100
Eberley		77		100	

Try to write a pseudo-code algorithm that takes in the name of two cities and finds a route between them. You will need to use a `while` loop (see textbook pp. 257–258 and 257). You can also check off names on the table.

What kinds of pitfalls lie in wait for your algorithm? What kinds of special cases would it have to deal with?

How could you test your algorithm without actually jumping in the car and driving the proposed route?

Exercise 2

Name ______________________________ Date ____________________

Section ______________________________

1) You have so many music CDs now that you no longer know what you have, where they are, or who has borrowed them. The other day at the mall you bought a really cool CD, only to discover back at home that you already owned it. That's it! Time to get organized.

   Use object-oriented design to design a program that will let you keep track of your collection. First, use brainstorming and filtering (textbook p. 179) to begin discovering what classes you need.

   Below is the beginning of a description of the required program. Complete it with a reasonable amount of detail. (See the bottom of p. 171 in your textbook for the address list example.)

   ```
   Create a CD catalog that includes all of my CDs with information
   about what music is on them,...
   ```

2) Circle the nouns and underline the verbs in your completed description.

3) Pick out three classes from the above description and fill out the following CRC cards (see textbook pp. 180–183). At this point, focus mostly on the class name and the responsibilities.

Class Name:	Superclass:	Subclass:

Responsibilities	Collaborations

Class Name:	Superclass:	Subclass:

Responsibilities	Collaborations

Class Name:	Superclass:	Subclass:

Responsibilities	Collaborations

4) Classes are related in one of three ways: *containment*, *inheritance*, or *collaboration* (textbook pp. 173–174). Next to each of the following pairs of classes, write the name of the appropriate relationship.

____________________	engine	automobile
____________________	mechanic	automobile
____________________	4-wheel vehicle	automobile

5) Now do the same for several of the classes that you identified in your CD catalog problem. Try to find one example of each relationship.

6) Choose a class from your CD catalog problem and answer the following questions related to information hiding, abstraction, and naming things (textbook pp. 186–188). You may use a different class for each question.

   a) What details are irrelevant to your program and should be omitted?

   b) List a class that you have named, but that does not really have a physical existence as one thing. (For example, a *nation* is a collection of people and institutions; a *sports team* is a collection of people. We often think of nations and teams as real things, but what happens when one of its parts leaves? Is it still the same thing?)

   c) Identify some action (responsibility) for a class that is a procedural abstraction. That is, it represents a lot of smaller actions.

Deeper Investigation

So far in the history of computing, people have written algorithms and programs, and computers have executed them. Computers devise solutions only in a few vary narrow subfields. Why do you suppose this is so? What is it about solving a problem, devising a plan, and writing an algorithm that is so hard that only humans usually do it?

Suppose that in the year 2052 computers routinely consult with humans who need a problem solved. The computers carry on some sort of conversation with the human, and then write the algorithm and execute it. What kinds of skills will computers need to do this 50 years down the road? What kinds of skills will humans need to be understandable? What are some of the dangers and pitfalls of having computers solve problems and write programs, instead of humans?