CS-160 Lab #6: Algorithms and Programming Intro

General

Read through the "Background" section below, then copy and paste the questions out of the "Assignment" section into your word processor and answer the questions. Turn in the questions using the instructions posted on the class web site.

At the top of the every document that you create (word processing or source files) include:

// Your name
// CS-160, Lab #X (replace the X with the Lab #)
// xxxx Term, 20xx (i.e. Winter Term, 2007)

For ALL Word processing documents, you must submit your documents in one of the following formats: MS-Word (NOT Works), RTF (most word processors can save in this format), or Open Document (used by the freely available Open Office suite). They will be returned ungraded if submitted in any other format.

Concepts

This lab will help you review the concepts presented in the algorithms and programming modules. These modules introduce you to several concepts important for understanding and expressing algorithms. You will start with an introduction to a basic problem solving technique ("enumeration" or sometimes known as "brute force"). Then you will gain some experience with reading and writing ***psuedocode*** which is one method of expressing an algorithm. Finally you will use the "Scratch" programming environment and tutorials to give you some experience with implementing algorithms as programs.

Background

Note: Original source of the background contained below is from the "CS160 Worksheets" by Daniel Balls of the CS department at Oregon State University; updated and revised by Mitch Fry (CS, Chemeketa Community College).

Problem-Solving By Enumeration

(aka "brute force" techniques)

The ability to solve problems is important in many disciplines, including computer programming. Computer programmers seek to obtain effective and efficient to perform desired tasks. Let's elaborate on three important terms in the previous sentence: algorithm, effective, and efficient. "An algorithm is an ordered set of unambiguous, executable steps that defines a terminating process" (Brookshear, *Computer Science: An Overview*, 2007). An algorithm is effective if it finds the solution (or a close enough approximation). An algorithm is efficient if the number of steps it performs is reasonable.

In this worksheet and the ones to follow, we will explore several important problem-solving strategies. It should be noted that there is no single set of rules that, if followed, always leads to an effective and efficient algorithm. In fact, problem solving relies so much on creativity and ingenuity that some people regard computer programming as an art form! Looking for better algorithms can be a lot of fun. With this in mind, think of the following worksheets as an opportunity to develop your own creative problem-solving techniques.

The first technique we are going to explore is called *enumeration*. Some types of problems have many possible solutions that are partially correct, but only one that satisfies all the criteria outlined. Enumeration means listing each of the possibilities and then systematically eliminating those that don't fit the requirements set out in the statement of the problem.

To illustrate problem-solving by enumeration , let's look at an example. Boronoff, Pavlow, Revistsky, and Sukarek are individuals who work in different fields of the arts and humanities: one is a painter, one is a writer, one a dancer, and one is a singer. Using the information below, determine the profession of each of these four people.

1) *Boronoff and Revitsky were both in the audience the night the singer made his debut on the concert stage.*
2) *The writer's biography of Sukarek was a best seller. He also wants to write Boronoff's biography, but hasn't met him yet.*
3) *Both Pavlow and the writer have sat for portraits by the painter.*
4) *Boronoff has never heard of Revitsky.*

To solve this problem by the enumeration technique, we will begin by listing all possible combinations of the individuals and their professions:

Dancer	Singer	Painter	Writer
Boronoff	Pavlow	Revitsky	Sukarek
Boronoff	Pavlow	Sukarek	Revitsky
Boronoff	Sukarek	Revitsky	Pavlow
Boronoff	Sukarek	Pavlow	Revitsky
Boronoff	Revitsky	Pavlow	Sukarek
Boronoff	Revitsky	Sukarek	Pavlow
Pavlow	Boronoff	Revitsky	Sukarek
Pavlow	Boronoff	Sukarek	Revitsky
Pavlow	Sukarek	Revitsky	Boronoff
Pavlow	Sukarek	Boronoff	Revitsky
Pavlow	Revitsky	Boronoff	Sukarek
Pavlow	Revitsky	Sukarek	Boronoff

Dancer	Singer	Painter	Writer
Revitsky	Pavlow	Boronoff	Sukarek
Revitsky	Pavlow	Sukarek	Boronoff
Revitsky	Sukarek	Boronoff	Pavlow
Revitsky	Sukarek	Pavlow	Boronoff
Revitsky	Boronoff	Pavlow	Sukarek
Revitsky	Boronoff	Sukarek	Pavlow
Sukarek	Pavlow	Boronoff	Revitsky
Sukarek	Pavlow	Revitsky	Boronoff
Sukarek	Revitsky	Boronoff	Pavlow
Sukarek	Revitsky	Pavlow	Boronoff
Sukarek	Boronoff	Pavlow	Revitsky
Sukarek	Boronoff	Revitsky	Pavlow

Now we'll use the clues to eliminate those possibilities that do not meet the first criterion. Since Boronoff and Revitsky were both in the audience when the singer was on stage, neither of them can be the singer. Therefore we will cross out (i.e. eliminate) each row in which either Boronoff or Revitsky is the singer, as shown below.

Dancer	Singer	Painter	Writer
Boronoff	Pavlow	Revitsky	Sukarek
Boronoff	Pavlow	Sukarek	Revitsky
Boronoff	Sukarek	Revitsky	Pavlow
Boronoff	Sukarek	Pavlow	Revitsky
~~Boronoff~~	~~Revitsky~~	~~Pavlow~~	~~Sukarek~~
~~Boronoff~~	~~Revitsky~~	~~Sukarek~~	~~Pavlow~~
~~Pavlow~~	~~Boronoff~~	~~Revitsky~~	~~Sukarek~~
~~Pavlow~~	~~Boronoff~~	~~Sukarek~~	~~Revitsky~~
Pavlow	Sukarek	Revitsky	Boronoff
Pavlow	Sukarek	Boronoff	Revitsky
~~Pavlow~~	~~Revitsky~~	~~Boronoff~~	~~Sukarek~~
~~Pavlow~~	~~Revitsky~~	~~Sukarek~~	~~Boronoff~~

Dancer	Singer	Painter	Writer
Revitsky	Pavlow	Boronoff	Sukarek
Revitsky	Pavlow	Sukarek	Boronoff
Revitsky	Sukarek	Boronoff	Pavlow
Revitsky	Sukarek	Pavlow	Boronoff
~~Revitsky~~	~~Boronoff~~	~~Pavlow~~	~~Sukarek~~
~~Revitsky~~	~~Boronoff~~	~~Sukarek~~	~~Pavlow~~
Sukarek	Pavlow	Boronoff	Revitsky
Sukarek	Pavlow	Revitsky	Boronoff
~~Sukarek~~	~~Revitsky~~	~~Boronoff~~	~~Pavlow~~
~~Sukarek~~	~~Revitsky~~	~~Pavlow~~	~~Boronoff~~
~~Sukarek~~	~~Boronoff~~	~~Pavlow~~	~~Revitsky~~
~~Sukarek~~	~~Boronoff~~	~~Revitsky~~	~~Pavlow~~

The second piece of information lets us know that the writer must be someone other than Boronoff or Sukarek. Therefore we will eliminate any row in which Boronoff or Sukarek is the writer, as shown below.

Dancer	Singer	Painter	Writer
~~Boronoff~~	~~Pavlow~~	~~Revitsky~~	~~Sukarek~~
Boronoff	Pavlow	Sukarek	Revitsky
Boronoff	Sukarek	Revitsky	Pavlow
Boronoff	Sukarek	Pavlow	Revitsky
~~Boronoff~~	~~Revitsky~~	~~Pavlow~~	~~Sukarek~~
~~Boronoff~~	~~Revitsky~~	~~Sukarek~~	~~Pavlow~~
~~Pavlow~~	~~Boronoff~~	~~Revitsky~~	~~Sukarek~~
~~Pavlow~~	~~Boronoff~~	~~Sukarek~~	~~Revitsky~~
~~Pavlow~~	~~Sukarek~~	~~Revitsky~~	~~Boronoff~~
~~Pavlow~~	~~Sukarek~~	~~Boronoff~~	~~Revitsky~~
~~Pavlow~~	~~Revitsky~~	~~Boronoff~~	~~Sukarek~~
~~Pavlow~~	~~Revitsky~~	~~Sukarek~~	~~Boronoff~~

Dancer	Singer	Painter	Writer
~~Revitsky~~	~~Pavlow~~	~~Boronoff~~	~~Sukarek~~
~~Revitsky~~	~~Pavlow~~	~~Sukarek~~	~~Boronoff~~
Revitsky	Sukarek	Boronoff	Pavlow
~~Revitsky~~	~~Sukarek~~	~~Pavlow~~	~~Boronoff~~
~~Revitsky~~	~~Boronoff~~	~~Pavlow~~	~~Sukarek~~
~~Revitsky~~	~~Boronoff~~	~~Sukarek~~	~~Pavlow~~
Sukarek	Pavlow	Boronoff	Revitsky
~~Sukarek~~	~~Pavlow~~	~~Revitsky~~	~~Boronoff~~
~~Sukarek~~	~~Revitsky~~	~~Boronoff~~	~~Pavlow~~
~~Sukarek~~	~~Revitsky~~	~~Pavlow~~	~~Boronoff~~
~~Sukarek~~	~~Boronoff~~	~~Pavlow~~	~~Revitsky~~
~~Sukarek~~	~~Boronoff~~	~~Revitsky~~	~~Pavlow~~

The third sentence informs us that Pavlow is neither the writer nor the painter and we will eliminate the rows in which Pavlow is the writer or the painter, as shown below.

Dancer	Singer	Painter	Writer
~~Boronoff~~	~~Pavlow~~	~~Revitsky~~	~~Sukarek~~
Boronoff	Pavlow	Sukarek	Revitsky
~~Boronoff~~	~~Sukarek~~	~~Revitsky~~	~~Pavlow~~
~~Boronoff~~	~~Sukarek~~	~~Pavlow~~	~~Revitsky~~
~~Boronoff~~	~~Revitsky~~	~~Pavlow~~	~~Sukarek~~
~~Boronoff~~	~~Revitsky~~	~~Sukarek~~	~~Pavlow~~
~~Pavlow~~	~~Boronoff~~	~~Revitsky~~	~~Sukarek~~
~~Pavlow~~	~~Boronoff~~	~~Sukarek~~	~~Revitsky~~
~~Pavlow~~	~~Sukarek~~	~~Revitsky~~	~~Boronoff~~
~~Pavlow~~	~~Sukarek~~	~~Boronoff~~	~~Revitsky~~
~~Pavlow~~	~~Revitsky~~	~~Boronoff~~	~~Sukarek~~
~~Pavlow~~	~~Revitsky~~	~~Sukarek~~	~~Boronoff~~

Dancer	Singer	Painter	Writer
~~Revitsky~~	~~Pavlow~~	~~Boronoff~~	~~Sukarek~~
~~Revitsky~~	~~Pavlow~~	~~Sukarek~~	~~Boronoff~~
~~Revitsky~~	~~Sukarek~~	~~Boronoff~~	~~Pavlow~~
~~Revitsky~~	~~Sukarek~~	~~Pavlow~~	~~Boronoff~~
~~Revitsky~~	~~Boronoff~~	~~Pavlow~~	~~Sukarek~~
~~Revitsky~~	~~Boronoff~~	~~Sukarek~~	~~Pavlow~~
Sukarek	Pavlow	Boronoff	Revitsky
~~Sukarek~~	~~Pavlow~~	~~Revitsky~~	~~Boronoff~~
~~Sukarek~~	~~Revitsky~~	~~Boronoff~~	~~Pavlow~~
~~Sukarek~~	~~Revitsky~~	~~Pavlow~~	~~Boronoff~~
~~Sukarek~~	~~Boronoff~~	~~Pavlow~~	~~Revitsky~~
~~Sukarek~~	~~Boronoff~~	~~Revitsky~~	~~Pavlow~~

After these three rows have been eliminated, we are left with only two rows: one in which Boronoff is the dancer and Revitsky is the writer (option A), and one in which Boronoff is the painter and Revitsky is the writer (option B). It is clear that Revitsky is the writer, and, according to the third piece of information, Revitsky sat for a portrait by the painter (who is Boronoff if option B is true). The last clue, however, states that Boronoff has never heard of Revitsky, which means option B cannot be true. Thus Boronoff must be the dancer, Pavlow is the singer, Revitsky is the writer, and Sukarek is the painter.

PSEUDOCODE STANDARD

(Adopted from Dr. John Dalbey, Cal Poly Computer Science Department)

Pseudocode is a kind of structured english for describing algorithms. It allows the designer to focus on the logic of the algorithm without being distracted by details of language syntax. At the same time, the pseudocode needs to be complete. It describe the entire logic of the algorithm so that implementation becomes a rote mechanical task of translating line by line into source code.

In general the vocabulary used in the pseudocode should be the vocabulary of the problem domain, not of the implementation domain. The pseudocode is a narrative for someone who knows the requirements (problem domain) and is trying to learn how the solution is organized.

Extract the next word from the line (good)
set word to get next token (poor)

Append the file extension to the name (good)
name = name + extension (poor)

FOR all the characters in the name (good)
FOR character = first to last (ok)

Note that the logic must be decomposed to the level of a single loop or decision. Thus "Search the list and find the customer with highest balance" is too vague because it takes a loop AND a nested decision to implement it. It's okay to use "Find" or "Lookup" if there's a predefined function for it such as String.indexOf().

Each textbook and each individual designer may have their own personal style of pseudocode. Pseudocode is not a rigorous notation, since it is read by other people, not by the computer. There is no universal "standard" for the industry, but for instructional purposes it is helpful if we all follow a similar style. However, in ALL pseudocode styles, ***INDENTATION is VITAL*** in communicating the blocks of operations that are nested within each of the structures described in the following sections.

The "structured" part of pseudocode is a notation for representing six specific structured programming constructs: SEQUENCE, WHILE, IF-THEN-ELSE, REPEAT-UNTIL, FOR, and CASE. Each of these constructs can be embedded inside any other construct. These constructs represent the logic, or flow of control in an algorithm.

It has been proven that three basic constructs for flow of control are sufficient to implement any "proper" algorithm.

1) **SEQUENCE** is a linear progression where one task is performed sequentially after another.
2) **WHILE** is a loop (repetition) with a simple conditional test at its beginning.
3) **IF-THEN-ELSE** is a decision (selection) in which a choice is made between two alternative courses of action.

Although these constructs are sufficient, it is often useful to include three more constructs:

4) **REPEAT-UNTIL** is a loop with a simple conditional test at the bottom.
5) **CASE** is a multiway branch (decision) based on the value of an expression. CASE is a generalization of IF-THEN-ELSE.
6) **FOR** is a "counting" loop.

SEQUENCE

Sequential control is indicated by writing one action after another, each action on a line by itself, and all actions aligned with the same indent. The actions are performed in the sequence (top to bottom) that they are written.

Example

Example (non-computer)
Brush teeth
Wash face
Comb hair
Smile in mirror

Example

READ height of rectangle
READ width of rectangle
COMPUTE area as height times width
Common Action Keywords
Several keywords are often used to indicate common input, output, and processing operations.
Input: READ, OBTAIN, GET
Output: PRINT, DISPLAY, SHOW
Compute: COMPUTE, CALCULATE, DETERMINE
Initialize: SET, INIT
Add one: INCREMENT, BUMP

WHILE

The WHILE construct is used to specify a loop with a test at the top. The beginning and ending of the loop are indicated by two keywords WHILE and ENDWHILE. The general form is:

```
WHILE condition
      sequence
ENDWHILE
```

The loop is entered only if the condition is true. The "sequence" is performed for each iteration. At the conclusion of each iteration, the condition is evaluated and the loop continues as long as the condition is true.

Example

```
WHILE Population < Limit
      Compute Population as Population + Births – Deaths
ENDWHILE
```

Example

```
WHILE employee.type NOT EQUAL manager AND personCount <
numEmployees
      INCREMENT personCount
      CALL employeeList.getPerson with personCount RETURNING
      employee
ENDWHILE
```

IF-THEN-ELSE

Binary choice on a given Boolean condition is indicated by the use of four keywords: IF, THEN, ELSE, and ENDIF. The general form is:

```
IF condition THEN
      sequence 1
ELSE
      sequence 2
ENDIF
```

The ELSE keyword and "sequence 2" are optional. If the condition is true, sequence 1 is performed, otherwise sequence 2 is performed.

Example

```
IF HoursWorked > NormalMax THEN
      Display overtime message
ELSE
      Display regular time message
ENDIF
```

REPEAT-UNTIL

This loop is similar to the WHILE loop except that the test is performed at the bottom of the loop instead of at the top. Two keywords, REPEAT and UNTIL are used. The general form is:

```
REPEAT
      sequence
UNTIL condition
```

The "sequence" in this type of loop is always performed at least once, because the test is peformed after the sequence is executed. At the conclusion of each iteration, the condition is evaluated, and the loop repeats if the condition is false. The loop terminates when the condition becomes true.

CASE

A CASE construct indicates a multiway branch based on conditions that are mutually exclusive. Four keywords, CASE, OF, OTHERS, and ENDCASE, and conditions are used to indicate the various alternatives. The general form is:

```
CASE expression OF
      condition 1 : sequence 1
      condition 2 : sequence 2
      ...
      condition n : sequence n
      OTHERS: default sequence
ENDCASE
```

The OTHERS clause with its default sequence is optional. Conditions are normally numbers or characters indicating the value of "expression", but they can be English statements or some other notation that specifies the condition under which the given sequence is to be performed. A certain sequence may be associated with more than one condition.

Example

```
CASE  Title  OF
        Mr      : Print "Mister"
        Mrs     : Print "Missus"
        Miss    : Print "Miss"
        Ms      : Print "Mizz"
        Dr      : Print "Doctor"
ENDCASE
```

Example

```
CASE  grade  OF
        A       : points = 4
        B       : points = 3
        C       : points = 2
        D       : points = 1
        F       : points = 0
ENDCASE
```

FOR

This loop is a specialized construct for iterating a specific number of times, often called a "counting" loop. Two keywords, FOR and ENDFOR are used. The general form is:

```
FOR iteration bounds
      sequence
ENDFOR
```

In cases where the loop constraints can be obviously inferred it is best to describe the loop using problem domain vocabulary.

Example

```
FOR each month of the year (good)
FOR month = 1 to 12 (ok)

FOR each employee in the list (good)
FOR empno = 1 to listsize (ok)
```

NESTED CONSTRUCTS

The constructs can be embedded within each other, and this is made clear by use of indenting. Nested constructs should be clearly indented from their surrounding constructs.

Example

```
SET total to zero
REPEAT
      READ Temperature
      IF Temperature > Freezing THEN
          INCREMENT total
      END IF
UNTIL Temperature < zero
Print total
```

In the above example, the IF construct is nested within the REPEAT construct, and therefore is indented.

INVOKING SUBPROCEDURES

Use the CALL keyword. For example:

```
CALL AvgAge with StudentAges
CALL Swap with CurrentItem and TargetItem
CALL Account.debit with CheckAmount
CALL getBalance RETURNING aBalance
CALL SquareRoot with orbitHeight RETURNING nominalOrbit
```