TEAM PROBLEM

Averaging Numbers

Do not mark this sheet. Write all work on looseleaf.

PROBLEM A computer program must calculate the average of all numbers entered as input.

Question What individual tasks must an averaging program do?

Before writing the complete averaging algorithm, focus on one part of the problem.

EXERCISE 1 How will the computer know when the user has finished entering all the numbers to be averaged?

a. Explain in English (not "computerese") a method for handling the problem of knowing how many numbers are to be averaged.
b. Give a *second* method for solving this problem.
c. Compare the two methods. Which method is more user-friendly and why?
d. Optional for extra credit: If possible, give a *third* method for solving this problem.

EXERCISE 2 Use one of the methods you listed in Exercise **1** above as part of a complete algorithm for solving the problem stated above (averaging numbers). Note these points.

a. List the steps of the algorithm in *English*.

Examples Get a number from the user.
Print the average.

b. Make sure the steps of the algorithm are in the correct logical order and are detailed enough so that a programmer could write the program from your description without having to ask any questions.

Note: It is not necessary that you know how to program every step of the algorithm from your present knowledge of programming.

Make a "scratch" copy of your algorithm first. Ask the teacher to scan your scratch copy to see if you are on the right track. Then, after revising and completing it, write a neat copy for submission.

Remember You are *not* writing a *program* for a computer. You are writing an *algorithm* for a human to read.

TEAM PROBLEM

Putting Numbers into Ascending Order

Do not mark this sheet. Write all work on looseleaf and trace paper.

PROBLEM

Input any three numbers. Print the numbers in *ascending* order.

EXERCISE 1

Draw a flow chart for a program that does what the problem above states. If two or more numbers are equal, print the equal numbers in any order among themselves.

EXERCISE 2

Trace your flow chart from Exercise **1**. Use the set of input listed below. If the flow chart does not produce the correct output for a set of input, revise it until it handles all input sets correctly.

a. 2, 3, 4
b. 2, 4, 3
c. 3, 2, 4
d. 4, 2, 3
e. 3, 4, 2
f. 4, 3, 2
g. 2, 2, 2
h. 3, 2, 2
i. 2, 3, 2
j. 2, 2, 3
k. 3, 3, 2
l. 3, 2, 3
m. 2, 3, 3

Suggestion Divide the trace in two. Let one team member trace using sets **a-g** and let the other team member test sets **h-m**.

TEAM PROBLEM

String Searches

PROBLEM Search a block of text for the occurrence of a given "target" string.

Two string-searching algorithms will be compared: "brute force" and "Boyer-Moore" (1977).

EXAMPLE 1 Use a "brute-force" method to determine if "pain" is contained in the string "The rain in Spain falls". List the steps in the process.

Solution

a. Compare the last (fourth) character of "pain" to the fourth character of "The rain in Spain".

```
pain
The rain in Spain falls
   ^                    "n" ≠ " ": no match
```

b. Shift "pain" one position to the right and test for a match again.

```
 pain
The rain in Spain falls
    ^                   "n" ≠ "r": no match
```

c. Keep shifting one position to the right and testing until a match is found or the end of the string being searched is reached.

```
  pain
The rain in Spain falls
     ^                  "n" ≠ "a": no match
        ...   (jump ahead)
    pain
The rain in Spain falls
       ^                One character matches; compare the previous character
    pain
The rain in Spain falls
      ^                 Two characters match; compare the previous character
    pain
The rain in Spain falls
     ^                  Three characters match; compare previous character
    pain
The rain in Spain falls
    ^                   No match; continue tsting one position to the right of
                        "rain" (where the first match was found)
     pain
The rain in Spain falls
         ^              "n" ≠ " ": no match
        ...   (jump ahead)
              pain
The rain in Spain falls
                 ^      "n" ≠ "i": no match
               pain
The rain in Spain falls
                  ^     One character matches; move left to see if the previous
                        characters in the two strings match
               pain
The rain in Spain falls
```

```
                  ^      Two characters match
               pain
The rain in Spain falls
                 ^       Three characters match
               pain
The rain in Spain falls
                  ^      Four characters match; SUCCESS after 20 comparisons
```

<continue over>

EXERCISE 1 Apply the brute-force method to determine if "lain" is in "falls mainly in the plain". Follow Example **1** but do *not* skip steps by putting three dots (...).

EXAMPLE 2 Use the Boyer-Moore algorithm to determine if "pain" is contained in the string "The rain in Spain falls".

Solution

a. Compare the "n" of "pain" to the fourth character of "The rain in Spain falls".

```
pain
The rain in Spain falls
   ^                  No match; since " " is not in "pain", shift four positions to the right
```

b.

```
    pain
The rain in Spain falls
       ^              One character matches; compare the previous character
```

c.

```
    pain
The rain in Spain falls
      ^               Two characters match; compare the previous character
```

d.

```
    pain
The rain in Spain falls
     ^                Three characters match; compare previous character
```

e.

```
    pain
The rain in Spain falls
    ^                 No match; shift right four places from the position of the first match of the sequence
```

f.

```
        pain
The rain in Spain falls
          ^           No match; shift right four places since " " is not in "pain"
```

g.

```
            pain
The rain in Spain falls
                ^     No match; shift right one place since "i" is one position from the end of "pain"
```

h.

```
              pain
The rain in Spain falls
                  ^   One character matches; compare the previous character
```

i.

```
              pain
The rain in Spain falls
                 ^    Two characters match; compare the previous character
```

j.

```
              pain
The rain in Spain falls
                ^     Three characters match; compare previous characters
```

k.

```
              pain
The rain in Spain falls
                ^     SUCCESS after only 11 comparisons
```

EXERCISE 2 Use Boyer-Moore to determine if "lain" is in "falls mainly in the plain". Write out all the steps as in Example **2** above. How many fewer comparisons are required by this method than the brute force method of Exercise **1**?

TEAM PROBLEM

Swapless, Pushless Sorting

Do not mark this sheet. Write all work on looseleaf.

PROBLEM

Sort a list of integers into descending order *without* swapping any values or pushing values up or down.

EXERCISE 1

Develop an algorithm for the problem above. Keep in mind the following.

1. Assume the list of integers is already in memory in an array.
2. Express the algorithm in *English*. You may write it in paragraph form or in outline form.
3. The teacher will be glad to look at what you have and tell you whether you are on the right track, whether it is specific enough, and so on.

EXERCISE 2 (optional for extra credit)

Write, run, and debug a program that implements the sorting algorithm you developed for Exercise **1**.

EXERCISE 3 (optional for extra credit)

Develop a sorting algorithm that does not involve swapping numbers, pushing numbers up or down, or copying numbers in the array to other locations.

TEAM PROBLEM

Dealing Cards

Do not mark this sheet. Write all work on looseleaf.

PROBLEM Develop several algorithms for dealing a deck of cards and storing them in the computer's memory.

EXAMPLE One algorithm for dealing a 52-card deck is as follows.

I. Set up a 52 x 2 array and clear all its cells to 0.

II. Repeat these steps until all 52 cards have been dealt.

- **A.** Generate a random integer from 1 through 13; this integer represents the rank of the card (1 = ace, 2 = deuce, ..., 11 = jack, 12 = queen, 13 = king).
- **B.** Generate a random integer from 1 through 4 to represent the suit of the card (1 = clubs, 2 = diamonds, 3 = hearts, 4 = spades).
- **C.** Do a sequential search of the array to determine if this "card" (combination of two integers) has already been dealt.
  1. If it is has not been dealt, store the rank and suit numbers in the next available row of the array.
  2. If the card has been dealt, repeat steps **A**, **B**, and **C**.

EXERCISE 1 Outline an algorithm for dealing a 52-card deck and storing the deal in the computer. The algorithm must be *essentially different* from the one in the example above.

Examples of "essentially different" are using a one-dimensional array instead of a two-dimensional array and/or filling the position in the array in a different order from the example above (but not just back to front instead of front to back).

EXERCISE 2 Do *either* of the following.

a. Outline another algorithm for dealing a deck of cards. This algorithm must be significantly different from both the example above and your algorithm for Exercise **1**.

b. Write a program to implement the algorithm you outlined in Exercise **1**. Print the cards as they are "dealt" or after all 52 have been stored. Submit a listing of the program and the output of a sample run.

EXERCISE 3 (extra credit) Do the other part of Exercise **2**.

TEAM PROBLEM

Text File Compression

Do not mark this sheet. Write all work on the separate answer sheet.

EXERCISE 1 The text file below has been condensed using the LZ algorithm. Use the substitution table to recreate the original file. Note: _ means a space.

```
SCREEN 0
WIDTH 40
CLS
?% '? TOP]\
?~   '? MIDDLEΛ\
?% '? BOTTOM]\
&
# ?%
∏#[?S THREE ROWS^
}R{
? "XXXXXXXXXXXX"
@R
& #
# ?~
∏#[?S AΛ BAR^
}F{
? "X"
@F
& #
```

Token	String	Token	String
~	DOWN	?	PRINT
%	ACROSS	}	FOR_
{	_=_1_TO_3	&	END
#	SUB	^	_OF_X'S.
\	_BAR.	@	NEXT_
[	PROGRAM_	]	_HORIZONTAL
∏	'THIS_	Λ	_VERTICAL

EXERCISE 2 Condense the following file as much as possible. Use at least *ten* tokens. List the table of replacements and the contents of the condensed file. Calculate the number of characters saved by condensing the file using your scheme.

```
Storage of floating-point numbers is based on a standard
notation. The floating-point number is normalized; that is,
converted to a form consisting of a mantissa times the
proper power of ten. The exponent of ten is the
characteristic. So a floating-point number is stored as two
integers, one representing the mantissa and another for the
characteristic.
```

TEAM PROBLEM

Text File Compression

Answer Sheet

Name(s) _____

EXERCISE 1 Write the unzipped file here. (Note: You may prefer to type and print the file.)

```
SCREEN 0
WIDTH 40
CLS
```

(over for Exercise **2**)

EXERCISE 2 Complete your substitution table here. (Note: Only *ten* tokens are required. However, space is available in the table for additional substitutions.)

Token	String	Token	String

Write the condensed file here. (Note: You may prefer to type and print the file.)

A. Characters in original file = _____

B. Characters in condensed file = _____

C. Characters in substitution table = _____

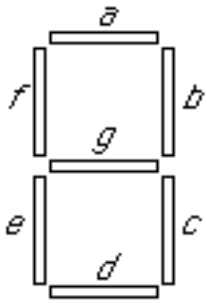
Characters saved: **A – (B + C)** = _____

TEAM PROBLEM

Seven Segment Display

Do not mark this sheet. Write all work on the separate sheets.

Most calculators, digital clocks, and timers use the "seven segment display" format. In this setup, as the diagram at the right shows, there are seven segments that can be lit in different combinations to form the numerals 0 through 9. For example, "1" is formed by lighting segments *b* and *c*; "2" consists of segments *a*, *b*, *g*, *e*, and *d*. "9" is composed of segments *a*, *b*, *c*, *d*, *f*, and *g*.



PROBLEM

Design circuitry to run a seven-segment display for one digit. The input consists of a four-bit digit (where each bit is an input line). The outputs are *a*, *b*, *c*, *d*, *e*, *f*, and *g* of the seven segment diagram (1 = light the segment, 0 = do not light the segment).

EXERCISE 1

Draw up the truth table for the problem. Note: There are only *ten* rows of input in the table corresponding to the bits for the digits 0 through 9. The first two rows and the last row of the table look like this. Copy these and then complete the other seven rows.

	In put				Out puts						
Digit	**b_3**	**b_2**	**b_1**	**b_0**	**a**	**b**	**c**	**d**	**e**	**f**	**g**
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...
9	1	0	0	1	1	1	1	1	0	1	1

EXERCISE 2

From the truth table (which is in effect *seven* truth tables in one) write *seven* Boolean expressions, one for *a*, one for *b*, and so on. Note: In most cases it is easier to use the *complement* approach.

EXERCISE 3

Use maps to simplify the expressions, if possible. Draw the seven minimal circuits, one for each output segment. *Note:* Include the maps with your solution.