APS105F: Computer Fundamentals
Lab Quiz, October 29, 1997

Instructions

- Place your student card on the desk top, so that it can be checked by the examiner.
- The only aid allowed is an original edition of the "Turing Reference Manual", the "On-line Turing Reference Manual", or other book that has a manual for Turing. If you are using "C" then you are allowed only a reference manual or other book that includes a "C" reference. You may not bring or use a photocopy of that manual, any other books, notes, written materials, or a calculator. You may reuse any programs in your own directories in building your program for this quiz.
- You must not communicate in any way (oral, written, or electronic) with anyone other than the examiner and his or her assistants. You should only talk to them if you believe there is an error or other problem in the statement of a question. They will not give you any advice.
- There are two questions requiring you to submit a Turing or C program. You will compose these programs in files called "`part1`" and "`part2`", respectively.
- **Insert your name and student number as comments at the beginning of each file.**
- You will submit these programs for grading by typing the commands "`SUBMITPART1`" and "`SUBMITPART2`", respectively. Note that these commands are Unix commands and they are typed as one word (no blanks) in block capitals. If you are in "oot", you must save your Turing file and exit to UNIX before running this command. The command will check that you have an appropriate file in your current directory.
- You may submit revised versions of your programs at any time before the end of the quiz by again typing the submit command. However, if you run this command before your quiz starts, or after the examiner has announced your quiz is over, you will receive a grade of zero.
- You may use the back of the pages for your rough work and pseudo-code design.
- The examiner will announce when there are 10 minutes and 5 minutes remaining in the quiz. It is your responsibility to submit your program before the end of the two hour quiz period. It is strongly advised that you do not leave this submission to the last minute.
- It is highly recommended that you save the file you are working on periodically; i.e. as soon as you have typed more than you are willing to type in again.
- Marks will be taken off for **poor programming style** and **obviously inefficient algorithms**.
- For the remaining questions do the work in the space provided or the back of other sheets. Clearly indicate if your work continues elsewhere.
- Complete and sign this sheet, and hand in the complete paper at the end of the quiz.

Name (print)______________________________

Login ID (print)______________________________

Student Number______________________________

Signature______________________________

Marks:	2 [5]	3 [25]	4 [10]	5 [10]	6 [10]	Total [60]

[Do this Now!] 1. Create two dummy text files called "`part1`" and "`part2`". The files can have anything or nothing in them for this test.

Test out the two submission programs, "`SUBMITPART1`" and "`SUBMITPART2`", and make sure that they work. Ask for assistance if you have difficulty.

You should submit versions of each program as you go along. Only your most recent submission will be kept. Submissions will be disabled when the end of the test is announced. Therefore, make sure that the submit programs work for you **NOW!!** and submit your programs well in advance of the end of the test.

[5 marks] 2. Write a recursive function that returns the value of the following function:

$$F(n) = \begin{cases} n & \text{for } n = 0, 1, 2 \\ F(n-1) + F(n-3) & n > 2 \end{cases}$$

What to do

Write a program that constantly prompts for a value for n. The program should output the value of F for the function above. When the input is less than zero, then the program terminates.

Save your program in a file called "`part1`" and submit it using "`SUBMITPART1`".

Assumptions

Assume that only legal inputs are entered.

[25 marks] 3. In the file "`/usr/copy/aps105/PATHin`" you will find an N by N matrix of dots and **P**'s. A smaller 6 by 6 sample is shown below. Note that the first line of the file gives the value of N.

```
6
.  .  .  P  .  .
.  .  .  P  P  .
P  P  .  .  P  .
.  P  .  P  P  .
.  P  P  P  .  .
.  .  .  .  .  .
```

What to do

(a) Copy the PATHin file to your own directory so that your program can read it in the same directory that it is running in. Leave the name of the file as PATHin.

(b) Write a procedure, called `PATHread`, to read the `PATHin` file and store each element into a two-dimensional PATH array of characters called **MAP**. Each character in the file is separated by two spaces and you do not have to store the spaces. A better solution will allow dynamic sizing of the array, where the array indices would run from 1 to N in Turing or 0 to N-*1* for C. For example, the **P** in the top row of the sample, would be stored in the element `MAP(1,4)` when using Turing. The `PATHread` procedure should take as parameters the array to be filled and the name of the file to read.

(c) Write a procedure that takes a PATH array like **MAP** as an argument and prints it on the screen in a format similar to the file.

(d) Write a procedure that will trace through the PATH array and print out a sequence of directions as if the array represented a map. Assume that the start of the path is on the west (left) side. Your program will have to find the entrance and then print out the directions from there using the letters **N**, **S**, **E**, and **W** for the compass directions, which correspond to *up*, *down*, *right*, and *left* for the example above. An example output of the procedure for the sample array is shown below. Note that the coordinates of the entrance and exit must also be printed.

```
The entrance is at coordinate (3,1).

The path goes: ESSEENENNWN.

The exit is at coordinate (1,4).
```

(e) Your complete program should read in the `PATHin` file, print it out on the screen, and then trace through the file to print out the directions.

Save your program in a file called "`part2`" and submit it using "`SUBMITPART2`".

Assumptions

The input file is in the correct format, with an entrance on the west side. The path does not ever split so there is only one route through the map. We will test your program with a different PATHin file.

Restrictions

Do not use any fancy formating commands in your output, such as the Turing `locate` function.

[2 marks] 4. (a) Sort the following complexities from slowest growth to fastest growth.

$$\begin{array}{lllll} O(n) & O(n^3) & O(n^n) & O(\log n) & O(n^2\log n) \\ O(n\log n) & O(2^n) & O(\sqrt{n}) & O(n\sqrt{n}) & O(n^{\log n}) \end{array}$$

[4 marks] (b) What are the complexities of the algorithms with the following pseudo code structures:

```
for i : 1 .. N

    for j : i .. N

    end for

end for
```

Answer: $\boxed{}$

```
for i : 1 .. N

    k := i;
    while(k > 1)
        for j : 1 .. k


        end for
        k := k/5
    end while

end for
```

Answer: $\boxed{}$

[4 marks] (c) What is the complexity of the function in Question 2. Use the back of this sheet for your work.

Answer: $\boxed{}$

5. Consider the following *pseudo code*:

```
var N : int initialize to 5

var A : 20 by 20 array of int

procedure t (B : 20 by 20 array of int, N : int)

    for i = 1 to N
        declare variable A initialized to N
        loop
            B (i, A) = i * A
            A = A - 1
            exit when A = 0
        end loop
    end for

end t

t (A,N)

for i = 1 to N
    for j = 1 to N
        put A (i, j), " " on the same line
    end for
    start a new line
end for
```

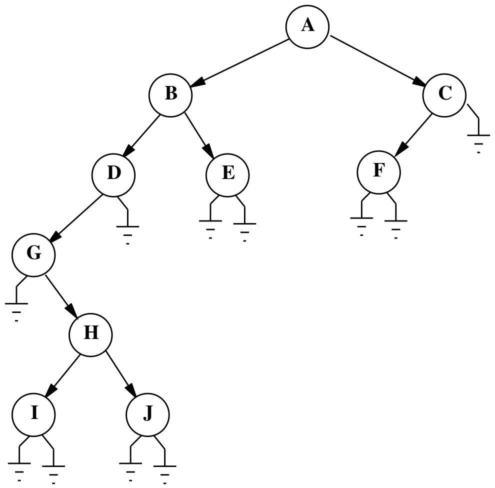
[5 marks] (a) Draw lines, as was done in class, to indicate the scope of all variables and the procedure. Use Turing-like scoping rules.

[5 marks] (b) Trace through several iterations of this program by hand showing how the variables are changed and explaining what the program is doing. Do this by building a table or use some other scheme that helps to clearly explain how the program works. For each array element, just show the values of the elements that are changing. Do not show the whole array each time one element changes!

What is the final output?

6. A very interesting, but simple data structure is the binary tree. Without getting into the details of how to implement them in a program there is still a lot that can be gained by looking at pictures and understanding how things work at that level. One thing that you want to do is to *traverse* the tree, which means you have to *visit* or go to every node in the tree.

Consider the tree shown in the figure below:



The arrows represent pointers to other nodes in the tree. When there is nothing to point to, then we say there is a *NULL* pointer as indicated by the electrical *ground* symbols in the figure.

One recursive algorithm to traverse the tree is shown below:

```
procedure traverse(root)  % root is a pointer to a tree/sub-tree
    if root = NULL then return

    traverse(left tree)   % follow the pointer to the left sub-tree
    print value of root   % print what is shown in the circle on the figure
    traverse(right tree)  % follow the pointer to the right sub-tree
end procedure
```

[5 marks] (a) What is the sequence of letters that would be output for the above tree when using this algorithm? Assume that the pointer to node A is the argument to the first call to traverse.

[5 marks] (b) Draw a tree that, when traversed with the above algorithm, will output the following string of characters:

```
8 * 4 - 6 * 3 + 5
```


Please:

- log on to the system
- start up oot, if you are going to use it
- do not turn this over until you are told to do so.