**ECE 4318 Final**

Name: **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Date:** 12/13/18, Thursday, 1 p.m. – 2:50 p.m., at 9-401

**Total Points**: 153 points

**Total Grade Points:** 20 grade points. 100 points = 20 grade points; points above 100 points are extra credits.

**Note:** For the final in class, you need to write code or pseudo code that gives idea on how it works. You do **not** have to run your source code and turn in output. However, for redo questions, those questions with coding need to be compiled and run (when drawing graphs for example, in class exam can be done by hand, but redo questions need to be done by software).

**Note:** you may redo or do up to 5 **(five)** questions of this final exam and submit by Friday 12/14/18, 6 p.m. Clearly specify which 5 questions you want to be reconsidered. Resubmission without specifying, without showing clearly your intention or with more than 5 questions resubmitted will not be considered.

**Matrix processing example by Nyhoff (22%)**

1. (12%) Code Analysis and reengineering. Please do this one in C programming language. When doing in class, you are **not** required to compile and run code with output, but you are required to turn in your reengineered source code

We have this piece of *spaghetti* code as an exercise of Nyhoff’s data structure book (ECE 304) (chapter 1).

**int row = 0, col;**

**A: col = 0;**

If (col < n) goto B;

goto A;

B: if (row < n) goto C;

goto E;

C: if (mat[row][col] == item) goto D;

col ++;

If (col < n) goto B;

row ++;

goto A;

D: cout << “item found\n”;

goto F;

E: cout << “item not found\n”;

F: ;

1. **(3%) Follow** the logic of this program and explain what it intends to do (does this compute a row sum, do a linear search, a binary search, a sort etc.?). Call this code C1.
2. **(3%) Explain** why this is called *spaghetti* code.
3. **(3%) Reengineer** or convert this to *structured* code. Call this code C2
4. **(3%)** Use a matrix mat =  to explain how code C1 and C2 work (in this case, n = 3).
5. (10%) Now consider the enhancement as follows:
6. **Modify** the structured code so that it computes and prints out the row sum for all rows.
7. **Enhance / modify** the code so that it computes the prints out the column sum for all columns as well.

**Linux driver spaghetti code from StackOverflow.com web site (30%)**

Below is part of a Linux SCSI driver code (called C3) with line numbers (incomplete). Full code is LinuxDriverExample.docx file in the distribution folder.

1: wait\_nomsg:

2: if ((inb(tmport) & 0x04) != 0) {

3: goto wait\_nomsg;

4: }

5: outb(1, 0x80);

6: udelay(100);

7: for (n = 0; n < 0x30000; n++) {

8: if ((inb(tmport) & 0x80) != 0) { /\* bsy ? \*/

9: goto wait\_io;

10: }

11: }

12: goto TCM\_SYNC;

13: wait\_io:

14: for (n = 0; n < 0x30000; n++) {

15: if ((inb(tmport) & 0x81) == 0x0081) {

16: goto wait\_io1;

17: }

18: }

19: goto TCM\_SYNC;

20: wait\_io1:

21: inb(0x80);

22: val |= 0x8003; /\* io,cd,db7 \*/

23: outw(val, tmport);

24: inb(0x80);

25: val &= 0x00bf; /\* no sel \*/

26: outw(val, tmport);

27: outb(2, 0x80);

28: TCM\_SYNC:

1. (12%) **Convert** this Linux driver partial code to structured code.
2. (4%) Lines 1 through 4 is a loop using goto. Convert this to a normal while or for loop. Show the code.
3. (8%) If we comment out line 12, then lines 7 through 13 will look like a for loop with a break. Convert this modified code snippet of lines to structured code.
4. (18%) **Try to convert** the whole code of lines 1 through 28 (line 12 unmodified) to structured code. Do as much as you can. How many loops and conditional constructs (if else) do you find?

**Project Management and Task Scheduling** (20%)

1. (20%) Scheduling and duration of a project. Suppose a project consists of 3 tasks each of 3 weeks long as follows.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Week | 1 | 2 | 3 | 4 | 5 |
| Task 1 |  |  |  |  |  |
| Task 2 |  |  |  |  |  |
| Task 3 |  |  |  |  |  |

Constraints:

* C1. Every task is done exclusively by one engineer; it can *not* be done by two engineers (such as engineer 1 doing the first 2 weeks, and engineer 2 finishing the 3rd week).
* C2. An engineer can *not* work on more than one task in the same week. The same engineer can work on different tasks in different week (so engineer 1 can work on task 1 in week 2 and then work on task 2 in week 3). We assume an engineer must be dedicated to one task in one week.
* C3. Task 2 can *not* start until the first week of task 1 is complete.
* C4. Task 3 can not start until the first week of task 2 is complete.
* C5. Common sense constraint: it is assumed that the third week of a task can not be done concurrently with week 2 (or week 1) of the task or before week 2 (or week 1).

The table above is a scenario with 3 engineers doing 3 tasks that take a total of 5 weeks duration. The man-weeks cost of this project is 3 \* 3 = 9 man-weeks.

1. Suppose there is only one engineer working on this project. What is the minimum duration of the project now?. Explain!
2. What would be the duration with 2 engineers working on this? Explain!
3. Can this project be finished in less than 5 weeks with 4 or more engineers? Why? Suppose the constraints stated above are still true.
4. If we remove constraint C3 and C4 but keep C1, C2, and C5, can the project be finished in 5 weeks with 2 engineers? Why?
5. If we remove constraint C1, C3 and C4 but keep C2, and C5, can the project be finished in 5 weeks with 2 engineers? Why?
6. An ambitious project manager wants to speed up the project and he / she has the hiring power of hiring 100 engineers on this project. With C1, C3, C4 removed but C2 and C5 stay, and with as many engineers as possible on this project, what is the minimum duration of this project? Why? Can this project be finished in 1 week?

**Dijkstra’s shortest path algorithm and depth first search algorithm in Data Structures (ECE 304 / ECE 3310) (30%)**

You have learned these two algorithms when you took the prerequisite ECE 304.

1. (18%) Dijkstra shortest path algorithm (concept and design, no coding necessary)
2. (3%) Concept: Explain briefly what Dijkstra’s shorted path algorithm is.



1. (15%) **Design**: Describe your top level / high level design on how you will implement this algorithm. The idea is on the design only, not coding yet. So block diagram (or flowchart) with the functions you may need is helpful. You can write your design in pseudocode if you want. You are NOT required to write code in this short period, but please show general ideas of how you define your data (what data are in that: distance between nodes?) and how do you update the distance data between nodes? How do you define the tree that starts from the start node say A until the final node C for example in Figure 1 on the left?

Figure 1

Furthermore, how do you plan to implement? Are you going to use structured approach with data, and functions etc., or are you going to use object oriented with classes and objects? In the answer of part (b), clearly indicate:

* Data: What are they and how you represent (for example, do you use arrays?). Show the names of the variables you are going to use.
* Functions: Give the name(s) of the function(s) you are going to use. Do you have functions that initialize, build and update the tree containing the tree T, the function to update the distance between nodes etc.?

1. (12%) Depth first algorithm (DFS). You have learned DFS in ECE 304
2. (2%) Concept: explain briefly DFS
3. (12%) Design: follow the same kind of argument as in Q6, think and elaborate the design of DFS. You’ll define the data and functions, but do NOT have to write code and test.

**Test Cases (16%)**

You have learned concepts of test cases before midterm. You can talk about test cases independent of some working source code.

1. (16+%) Think about 2-3 test cases for each of the following (2-3 cases for each part, with normal case, boundary case, abnormal case etc.) If 2-3 cases are not enough, and you feel that you may need 4, 5, or 6 cases, then do so. Of course, to test n! for n = 1, 2, 3, .., 50, you do NOT need 50 test cases however.
2. n! (n factorial) function. (test up to 50!)
3. Fibonacci numbers (test up to say the 50th Fibonacci numbers)
4. Dijkstra shortest path (may be one case with 4 nodes like Figure 1, and another with probably 6 nodes?) You may have to draw the graph by hand during in class exam or draw the graph by software in the redo.
5. DFS (Depth first search): one case with say 3 nodes below the root R, and another case with say 5 nodes below the root R.

**Difference of Computer Languages (24%)**

You have learned structured language C in ECE 114/1310, object oriented language C++/C# etc., in ECE 256/2310, assembly language (say PIC 18) in ECE 341. When you use compiler, programs written in C or C++/C# are compiled into machine code; when you use assembler, assembly programs are assembled into machine code.

1. (12%) Computer languages. (here we do not talk about interpretative language like Python).
2. Explain what machine language is. Give example of a couple lines of machine code
3. Explain what assembly language is. Give example of a few lines of assembly code and explain what that does.
4. Repeat (b) for structured language like C (or Fortran etc.)
5. Repeat (b) for an object oriented programming language like C++ / C# / Java etc.
6. (12%) For parts (b), (c), and (d) of Q9, find how the assembled machine code or the compiled machine code look like and relate them to the original source code.

Software Metrics (9%)

Chapter 24 Quality Management covers software metrics.

1. (9%) Fan-in, fan-out. Slide 59 covers this concept and describe it as Fan-in is a measure of the number of functions or methods that call another function or method (say X). Fan-out is the number of functions that are called by function X. A high value for fan-in means that X is tightly coupled to the rest of the design and changes to X will have extensive knock-on effects. A high value for fan-out suggests that the overall complexity of X may be high because of the complexity of the control logic needed to coordinate the called components.
2. Explain if you concur with the definition there of fan-in and fan-out. Does this mean high fan-in or high fan-out means a function is significant? Can you relate this to web site (the fan-in of web site and the fan-out of web site)?
3. Can you find example(s) of functions of web sites with high fan-in numbers such as 5?
4. Can you find example(s) of functions of web sites with high fan-out numbers such as 5?