Final Exam - APS105T - Spring 2016

B.Korst, P.Eng.

For this Final Exam, you are limited to using only the programming techniques presented in the lectures and covered in chapters 1 through – and including – 11 of the ZyBook. You can use your ZyBook, other books and notes or the Internet as a reference, but you cannot exchange messages with anyone *in any way*.

In addition, *programs copied from another student or straight, word for word, from the internet will result in a zero for the exam and an academic offence process.*

You will use NetBeans as your programming environment.

After every question for which you write code to answer, follow the submission procedure. You are free to run the submission procedure more than once. When you are done compiling and testing your programs, follow the instructions found at the end of this document.

Question 1a - 5 Marks - 5 minutes

Run the code found in project `FinalQ01a`. The code was intended to create a recursive function to calculate a factorial, but it presents a stack overflow. Resolve the problem and submit a working code.

Question 1b - 5 marks - 10 minutes

Write two functions: one that allocates memory space for a type double and returns a pointer to the type doublePointer (defined for you), then a second function to ensure no memory is wasted when the space is no longer needed. Your main function will

1. declare two pointers, `pi` and `e` of type `doublePointer`;
2. call one of your functions to allocate space and return a pointer to the allocated space;
3. assign the value 3.1415 to the location pointed at by `pi` and 2.718 to the location pointed at by `e`; and
4. will release the memory space allocated in item 2.

The skeleton code is found under `FinalQ01b`.

Question 2 - 5 Marks - 5 minutes

Write a full program in C that takes two inputs by the user and calls a function to check if the first number input is exactly divisible by the second number. The function should return a `bool` type, so your program should include the appropriate library in C. Your code should be inserted in the skeleton code given in project `FinalQ02`.

Question 3 - 10 Marks - 20 minutes

During election seasons, many public polls are taken to determine the people's changing support for a given party or candidate. Over the short term, the poll numbers may fluctuate widely, and so it is common practice to report the average of the last 'X' number of polls (e.g. the last 3 polls, or last 5 polls, etc.).

Write a program that repeatedly prompts the user for an integer input between 0 and 100 (inclusive), and reports the average of the last 5 inputs after every input of the user. Note that if the program has not yet received 5 inputs, it should only calculate the average of the numbers it has received thus far. In addition, as the user enters more values, only the *last five* are used to calculate the average.Your program should discard any numbers that are over 100, and not factor that into the average of subsequent results. Upon receiving a negative number as input, the program should exit.

An example output of the program is shown below, where the user input is in bold.

```
Input latest number:  1
Latest average is:  1.00
Input latest number:  3
Latest average is:  2.00
Input latest number:  5
Latest average is:  3.00
Input latest number:  202
Invalid input
Input latest number:  -3
Exiting program...
```

Insert your code in the skeleton code found in project `FinalQ03`.

Question 4 - 10 Marks - 20 minutes

This question makes use of command line arguments. Write a full C program to copy an existing file into another file. If the destination file exists, re-write it; if it does not, create a new file. Both files should be kept in the same directory as your C program is (this is the default directory). The ”existing” file is given to you, it is called `file1.txt`, and it contains a few lines of strings. You must include in your program a check for the number of arguments entered by the user. If the number of arguments is not exactly the number needed, your program must exit and print an error.

Two libraries `stdio.h` and `string.h` are included in the skeleton code and you must use *only* these two libraries. Your program should make use of functions such as `fgetc` and `fputc`.

In the comments of your solution code you must indicate how your program is supposed to be run by a user, if the user decides to run it via the command window. The skeleton code is found under project `FinalQ04`.

Question 5 - 15 Marks - 20 minutes

The value of the mathematical constant e can be approximated by the formula:

$$e \approx \sum_{n=0}^{m} \frac{1}{n!} = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \ldots$$

Write a complete C program that calculates the value of e to a precision of 10^{-6}. You must print the value of e calculated by your program, as well as the number of terms (that would be the m in the equation) in the summation when you reached the required precision. The skeleton code is found in project `FinalQ05`.

Question 6 - 15 Marks - 20 minutes

The program given in project `FinalQ06` calls a recursive function `quickSort` to sort an array of integers given. The program runs into a segmentation fault, right on the first try, so we know that there are problems in it. Your task is to resolve the problems and sort the array of integers given. Some problems are in the way the Quick Sort algorithm is implemented, but other problems appear at other parts of the program.

There is a total of **five** problems. These are NOT syntax problems; they may be logic, function calls misplaced, wrong parameters, etc.

Solve the problems and submit your code. When you find a problem, write a comment in the program stating that you found the problem and solved it.

Question 7 - 15 Marks - 30 minutes

Your task is to write a function that calculates the value of a resulting ”equivalent” load if two complex loads are connected in parallel. The skeleton code is provided in project `FinalQ07`, and you may use only the libraries given in the skeleton code. The result of a parallel equivalent is expressed by the equation below, where `Za` and `Zb` are the loads in parallel.

$$Z_{parallel} = \frac{Za * Zb}{Za + Zb} = \frac{Product}{Sum}$$

Write a function called `parallel` that will calculate the resulting load by means of a complex division between the product and the sum as shown above.

The steps to effect the complex division can be illustrated as follows for a generic complex number Z. An itemized explanation is also shown below.

$$\frac{Z_{num}}{Z_{den}} = \frac{(Z_{num} * Z_{new})}{Z_{real}} = \frac{R_{real} + jR_{imaginary}}{Z_{real}} = \frac{R_{real}}{Z_{real}} + j\frac{R_{imaginary}}{Z_{real}}$$

1. Create a new complex number which is a copy of the denominator, but with the changed sign on the imaginary value. For example:
$$Z_{den} = C + jD \qquad \text{then} \qquad Z_{dnew} = C - jD$$
2. Multiply the numerator by Z_{dnew}. The product is also complex, called R, which has a real and an imaginary component;
3. The product between the denominator and Z_{dnew} above will result in a purely real number, which is calculated as follows:
$$\text{If} \qquad Z_{den} = C + jD \qquad \text{then} \qquad Z_{real} = C^2 + D^2$$
4. Divide the real and imaginary parts of R (both are type `double`) by Z_{real} (also a type `double`) calculated in 3).
5. Return the complex value.

The two inputs to function `parallel` will be the two loads, and the prototype is:

```
IMPEDANCE parallel (IMPEDANCE t, IMPEDANCE v);
```

The printf calls are given, so make sure you pass the appropriate parameters to the printf call. You can check your result at:
`http://hyperphysics.phy-astr.gsu.edu/hbase/electric/serpar.html`

Question 8 - 20 Marks - 30 minutes

The skeleton code found in `FinalQ08` is part of a code to take a list of planet names from a file and create a linked list with them. The input file is called `planets.dat`. The resulting linked list is meant to be a sorted list by the planets' distances from the sun, and the list is to contain only unique names.

Take a look at `planets.dat`, and look at the the code that is given to you. For this question, you may **not** alter the `main` function. The overall flow of the main function is as follows: store all the planet data from planets.dat into a linked list (sorted by their distance from the sun), print the entire list (which may contain more than one entry per planet), filter the list such that only the unique planets remain, and finally print the list again after filtering. Your task is to complete the two functions: `insertPlanet` (which inserts *in order*) and `filterUniques`.

Submitting the solutions

You will submit your solutions, question by question, in a plain text .c file. You will save your program from NetBeans, and the questions should already be in files named per individual question, i.e. **FinalQ01a.c**, **FinalQ01b.c**, **FinalQ2.c**, etc. You will start by opening a terminal window and typing:

- When you are ready to submit, go to a terminal window (it could be the same one from which you opened the test.)
- Enter `aps105final submit` into the command line and press enter.

If you submit more than once, the latest files will REWRITE the previous submission. Only the latest submission will be considered.

You must submit your answers, or no record of them will be kept.

Note: To list your submission history you can use the following command: `aps105final list`.
Use this command regularly to check that you have submitted the questions you have answered.