CS699 Software Lab End Sem
23-Nov-2011, starting 2:30pm, Max. Marks: 30

1 General instructions

- Like in quiz2, you cannot use local login: you have to use the USB boot image or your own laptop.
- As in earlier exams, you are allowed to carry any amount of reference material.
- References are also provided in the submission server. Note that the following new material has been added recently, which you may find useful: 100-vim-commands.pdf, emacs-commands1.html, emacs-commands2.html, sed-tutorial-bruce-barnett.html, awk-tutorial-bruce-barnett.html.
- You **cannot** carry any of other students' material such as their code or their handwritten notes.
- Mobiles are not allowed; if you carry your mobile to the lab, it must be **switched off** and kept on the table next to you for the invigilators to see.
- Laptops' WiFi and other wireless interfaces (Bluetooth, 2G, 3G, etc.) must be disabled, preferably using a hardware switch which most laptops have.
- Any **attempt to cheat**, successful or not, will be punished severely.
- The exam is open-ended in theory, but in practice, after about 6-7 hours, I will start pestering people to get done soon.

2 Submission Instructions

1. Use the script prepare-upload.sh to create the file forupload.tgz. As you can see in the prepare-upload.sh script, ONLY the relevant files are copied over for submission:
2. In the unlikely event that you need to submit any file(s) other than the above files, talk to the instructor.
3. Upload the forupload.tgz file, onto the given URL.

3 Questions

---

1. (a) Read the general instructions
   (b) Read the general instructions again
   (c) Read the general instructions once again

---

2. **Emacs of Vim [1 mark, DEMO during exam]** During the exam, you have to demonstrate to the instructor or a TA, any significant feature of emacs or vim. This has to be a feature absent in a regular editor such as gedit.

   You have to be prepared to show the demo *any time* after 1 hour of the start of the exam, and before you submit and leave.

   The instructor/TA may come and ask you for the demo *any time* after the first hour. The demos will be taken in random order.

   Note that there is no submission for this part of the exam.

---

3. **SVN usage [1 mark, DEMO during exam]** Just like the emacs/vim demo, you have to demonstrate this during the exam. The rules are the same: be ready *any time* after the first hour. You have to show that you have the end-sem directory and the relevant files within it, as a working copy somewhere in some svn repository: this can be a new repository, or the same as what you have been using earlier.

   You will *not* be submitting the svn repository; you will only be submitting some specific files from the working copy.

---

4. **Script to tabulate Makefile marks in quiz-2 [2 marks]**

   File to be used: tabulate-makefile-marks.sh

   During manual correction of the Makefile submissions in quiz-2, I created a file called makefile-correction.txt for each student. The format of this file is: first line has the marks preceded by "Makefile marks="; other lines are for explanation.

   There is one directory per student, with the directory name being the student's roll number. See the testruns/makefile-corr directory for an example. There are no other files or directories in testruns/makefile-corr. You can assume that there is a makefile-correction.txt file in each student's directory, even if the student has not submitted the Makefile.

   When run from within the testruns/makefile-corr directory, your script should print output as per what is given in the file "testruns/testruns.txt".

   Marking: 1 mark for correctness, 1 mark for a solution which involves a single loop, with a single line body. *Hint for single line solution:* I was able to do it using sed.

---

5. **SED-based section print in quiz-2 code [3 + 1 = 4 marks]**

   (a) **File to be used: quiz2-sec04.sed**

   In quiz-2, you had to write a python script within a template file provided. Your code had to be within the lines marked BEGIN to END for each section: 01, 02, ... 10. In this context, write a sed script in the file quiz2-sec04.sed, so that when you run
   `sed -f quiz2-sec04.sed some-input-python-code.py` the output matches the following specifications:

   - **[1 mark]** It should print the lines of code appearing between the BEGIN and END lines of section 04 (marked by `DONT_ERASE_04`).
     Hint: In sed, you can specify a range of lines like this:
     `sed '3,6 d'`
     will delete lines 3 to 6 in the input. The range specification can be in terms of a pattern too, like `/regexp-pattern/`: both the start as well as the end. Also, find out how to specify end-of-file as marker.
   - **[0.5 marks]** It should ignore (i.e. not print) all lines involving print commands.
   - **[0.5 marks]** It should ignore all "None" statements. Note that the word "None" may appear as part of regular code, e.g. as part of variable assignment or as an argument to a function call, in which case the statement is not a "None" statement.
   - **[0.5 marks]** It should ignore all comment lines; i.e. lines starting with # as the first non-space character.
   - **[0.5 marks]** It should ignore all empty lines; i.e. lines with no non-space characters.

   Note-1: you cannot assume anything about the input file, such as number of lines or any such thing.

   Note-2: Be sure to see the sample test cases.

   (b) **File to be used: gen-quiz2-secX.sh**

   Write a single line sed command in file gen-quiz2-secX.sh which will a single command-line argument, the section number (e.g. 08), read quiz2-sec04.sed in STDIN, and write out the sed script for that section in STDOUT. (You need not handle the case where the argument given does not have a leading 0). See the sample test cases for details on how this is supposed to work.

---

6. **Postfix mail log parsing [5 marks]**

   In this exercise, we are going to extract some statistics from the mail log of a popular SMTP daemon, postfix. Example mail log files are given in the testruns directory (these are slightly modified version of the original mail log file, to make your awk scripting easier).

   Each mail sent by the daemon has an id. For instance, the very first entry in the "mail.log.example1" log has an id "873FD248DF9". This id repeats in several lines, as the mail goes through several stages of processing. You can assume for this exercise that the log entries of different emails are not interspersed with one another.

   You *have to* use awk scripts for the following exercises, although it may be possible to solve it alternatively. In each case, look at the test run outputs for an idea of what/how you should print.

   (a) **Mail log in a nice format: 1 mark**

   File to be used: maillog.awk

   First write an awk script to print for each email sent, the id, the from address, the to address, and the mail size.

   (b) **Counting number of bytes processed: 1 mark**

   File to be used: count-bytes.awk

   Now write an awk script to count the total number of bytes sent by the mail server.

   (c) **Counting number of bytes per recipient: 2 marks**

   File to be used: count-bytes-per-recipient.awk

   Now write an awk script to count the number of bytes per recipient.

   (d) **Counting number of emails per recipient: 1 mark**

   File to be used: count-emails-recd.awk

   Write an awk script to count the number of emails received, for each recipient.

---

7. **Eratosthenes sieve using Python lists [3 marks]**

   File to be used: eratosthenes.py

   The Eratosthenes sieve method is a popular method to list all the prime numbers $\leq N$. It starts with an array of numbers, which is initially 2 to N. In each step, some numbers are crossed out (they fall through the sieve). The numbers to be crossed out in a step are the numbers $\leq N$ which are multiples of x, where x is the next largest number not to be crossed out so far (in subsequent steps, the x increases monotonically). The final list of numbers not crossed out represents the list of prime numbers $\leq N$.

   Given that the algorithm deals with lists at each stage, this is a prime candidate for elegant implementation in python. Complete the code given in the final "eratosthenes.py". Beware of where you can add code in the given template!

   Marking: 1 mark for correctness, 1 additional mark for solution based on filter, 1 additional mark and 3HP for solution involving a single line for-loop-body.

---

8. **Python script to help with house-point (HP) additions [6 marks]**

   File to be used: hp-for-the-day.py

   As in quiz-2, most of the information is given in the code template itself. Beware that you should add code only within the BEGIN-END sections. Don't forget to look at the testrun outputs at each stage.

---

9. **Simplified latex2html using lex/yacc [6 marks]**

   Files to be used: mylatex2html.l, mylatex2html.y, mylatex2html.h, Makefile

   (The file mylatex2html.h may or may not be necessary to modify.)

   The program latex2html does a good job of converting a latex document to HTML. In this exercise, we are going to write a very simplified version of latex2html: handling only a single latex table as input, and outputting a HTML version of the same.

   The input format we will consider is shown through examples table1.tex and table2.tex, and is described below.

   - The latex table begins with a begin-tabular as shown. The contents within the second curly braces in this line specifies the indentation and bordering of various columns; this information can be ignored for this exercise.
   - The latex table ends with an end-tabular as shown.
   - In-between the begin and end, each row of table ends with a double-backslash.
   - Within each row, different columns are separated by the & character.
   - The contents of each cell can consist of any number of words separated by space characters.
   - Each word can be any combination of characters [a-z] [A-Z] [0-9], comma, dash, and full-stop.
   - We'll assume that spaces and newlines can be ignored, except as separators between other significant tokens.
   - For simplicity, we'll also assume that you don't have to handle any table borders, or other formatting like bold or italics.
   - You can assume that there are an equal number of columns per row, and there is no need to check this.
   - For ease of parsing, you can assume that there is at least one row in the table, and that there is at least one column in each row.

   (a) **Makefile: 1 mark** Write an appropriate Makefile for your compilations.

   (b) **Row count: 1 mark** Use the lex code only, or the lex+yacc code to count the number of rows and print it onto STDERR.

   (c) **Column count: 1 mark** Use the lex code only, or the lex+yacc code to count the number of columns and print it onto STDERR.

   (d) **Latex table to HTML: 3 marks, 5HP** Write the grammar for parsing, and print the final HTML table onto STDOUT. *Hint:* I found it useful to use appropriate #define statements to limit the number of columns as well as the number of bytes in each cell entry. I found it easy to statically store an entire row of contents before printing it. This may not be necessary however.

Note-1: This question will be semi-manually corrected, so you need not be exact with respect to output formats.

Note-2: I have given my executable parser mylatex2html.br.exe in the testruns directory so that you can test against any test output.

---

10. **PHP to upload a latex file and run mylatex2html [2 marks]**

    File to be used: mylatex2html.php

    Add to the template file provided so that the PHP file will run your mylatex2html and display the output as a table as part of its output HTML. You can use the file upload-latex.html to upload the file to "mylatex2html.php".

    Note-1: In case your 'mylatex2html' does not work, you can use the 'mylatex2html.br.exe' provided in the testruns directory to test your PHP code.

    Note-2: you may have to copy the PHP code to some location in /var/www to test it. If you modify the PHP code in the /var/www location, **do not forget** to copy to the submission location before submitting! The best is to use soft-links, if your apache2 server supports it (if you don't know what this means, don't try to learn during the exam).

    Reference hints: Look at the files features.file-upload.html and book.exec.html in the PHP reference provided.
