INDIAN STATISTICAL INSTITUTE

First Semestral Examination : 2014–15

Course : Master of Technology in Computer Science (First Year)

Subject : Introduction to Programming (A1) for M.Tech. (CS) I

Date : 19 November 2014 Maximum Marks : 100 Duration : 3 Hours

Note: You are expected to code in C to solve the following problem(s), and you are allowed to use any academic resource of your choice (except the Internet) during the course of this examination. Write the theoretical part(s) of the solution(s) in the answer-sheet provided, and submit the relevant C source file(s) separately. In the theoretical part(s) of the solution(s), you must provide basic sketch of the algorithm(s) that you use, along with basic complexity analysis of your chosen algorithm(s), wherever appropriate. You must justify the C data-type(s) that you use while coding, and provide basic pictorial illustration(s) of the memory stack-frame(s) for clarification, wherever appropriate.

Problem 1 : Short Questions

Maximum Marks: 50 = 5 + 5 + 5 + 10 + 10 + 15

1A. Is it possible to determine the byte-size of a structure in C without using the inbuilt `sizeof()` function? If so, write a sample C code snippet to demonstrate your strategy.

1B. Devise a one-pass algorithm to find/print the data located exactly at the middle of a singly-linked list containing odd number of nodes. Write the pseudocode for your algorithm.

1C. Hamming weight of a binary string refers to the number of 1's in the string. Write a C program that takes a binary string as input from the user, and outputs the Hamming weight of the same.

1D. Palindromes are words or phrases (or any sequence of symbols) that read the same forward or reversed, ignoring the punctuations and blank spaces. Write a C program that takes an English string (may be a few words long) as input from the user, and checks if it is a palindrome.

1E. Write a C program that takes an English word (size within 10 characters) as input from the user, and writes all possible permutations of the same in a file. Make sure that the permutations are generated and written in a lexicographic (alphabetic) order.

1F. Write a C program that imitates the standard UNIX function `cat`. The program should take as command-line input one or more filenames, and print on the standard output all the files in a concatenated format. Make sure that the program exhibits plausible errors in a proper format.

Problem 2 : Integer Polynomials

Maximum Marks: 25

Polynomial arithmetic is analogous to Integer arithmetic, especially in terms of addition and multiplication. If we define the notion of size for polynomials as their degree, with constant polynomials defined to have degree zero, we get a natural notion for the division algorithm with polynomials $a(x)$ and $b(x)$, where $\deg(a) \geq \deg(b)$, as follows.

$$a(x) = q(x) \cdot b(x) + r(x) \qquad \text{with quotient } q(x) \text{ and remainder } r(x) \text{ with } 0 \leq \deg(r) < \deg(b).$$

This automatically lets us define divisibility of polynomials as '$b(x)$ divides $a(x)$' if and only if $r(x) = 0$, and therefore, extends the notion of GCD naturally to the polynomials. The GCD of two polynomials $a(x)$ and $b(x)$ is a polynomial $d(x)$, of the highest possible degree, that divides both $a(x)$ and $b(x)$.

Write a C program to represent polynomials with integer coefficients in a format suitable for basic arithmetic operations like addition and multiplication. Use the basic operations to implement a quotient-remainder division routine for integer polynomials, and then use this division to implement a complete GCD routine for integer polynomials. At the end of the day, your program should take two polynomials as input from the user, and output their GCD.

Example 1 : Input $x^4 + 2x^3 + 2x^2 + 2x + 1$ and $x^3 + 2x^2 + 2x + 1$. Output $x + 1$.
Example 2 : Input $x^4 + x^2 + x + 1$ and $x^3 + 2x^2 + 2x + 1$. Output 1.
Example 3 : Input $x^2 + 2x + 1$ and $x + 1$. Output $x + 1$.

Problem 3 : Regular Expressions

Maximum Marks: 25

Regular expressions are sequences of characters forming a search pattern, mainly for use in pattern matching with strings. We will consider a small class of regular expressions, consisting of the following search patterns and identifiers.

`x` : Matches the exact character `x` in input string

`.` : Matches any single character in input string

`*` : Matches zero or more occurrences of previous character

`^` : Matches only at the start of input string

`$` : Matches only at the end of input string

Write a C program that matches regular expressions covering the above-mentioned classes of search patterns. Your program should accept as input a regular expression conforming to the classes defined above, and an input string. It should output 1 if the regular expression matches the input string (all lower case letters), and 0 otherwise.

Example 1 : Input `^so` and `sourav`. Output 1.
Example 2 : Input `^ra` and `sourav`. Output 0.
Example 3 : Input `^.ra` and `sourav`. Output 0.
Example 4 : Input `^.*ra` and `sourav`. Output 1.
Example 5 : Input `ra$` and `sourav`. Output 0.
Example 6 : Input `rav$` and `sourav`. Output 1.
Example 7 : Input `ra.$` and `sourav`. Output 1.

Solving a problem correctly is necessary, but not sufficient, as it does not guarantee the maximum marks alloted for the problem. Sufficient credit is reserved in each case for *smart* algorithm and *good* coding practice. Good luck! ☺