CS3101: Class Test 3

Duration: 1 hour 15 mins **Total marks: 25**

Instructions

- All programs should be compatible with GNU C compiler.
- There will be a plagiarism check. The suspected copies will attract a deduction of marks for the problem irrespective of who is the original author of the code. Copying from internet will also be considered as plagiarism.
- Late submissions will attract marks deduction.
- You are advised to compile the code before submitting them to WeLearn. If a code does not compile at our end, marks will be deducted for the problem.
- Each program should follow a strict naming convention: **QNo.c** (e.g. Q1.c, Q2a.c etc.). Programs not adhering to the convention will not be corrected.
- All codes (.c files) should be submitted in a single zipped folder (folder name containing your WeLearn ID e.g. 17MS001) to WeLearn.
- You should put appropriate comments in the code.
- You may reuse the codes shared by the instructor.

1. **(Marks: 5)** Write a C program to generate the pattern in Figure 1 for any integer n (> 0), where Figure 1 shows the output for $n = 10$.

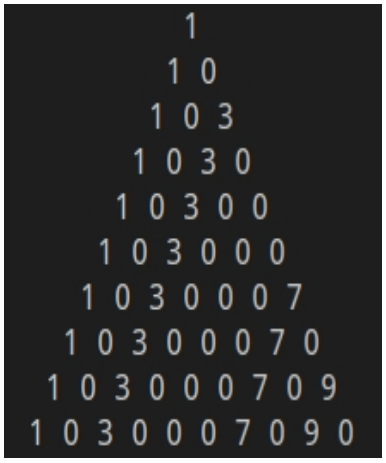

Figure 1: Example output (n=10) for Q1

2. **(Marks: 6)** A 2-D square matrix (number of rows same as the number of columns) M is idempotent if $M * M = M$ (that is, the product of the matrix with itself is same as the matrix itself). Write a C program to (i) read a 2-D matrix M from the user using **only** pointer notation and (ii) check if it is idempotent using **only** pointer notation and print an appropriate message. That is, you can only declare the 2-D matrix in a non-pointer notation (e.g. int M[3][3]) or any other temporary matrix, if required.

Thereafter, you have to use only pointer notation (used for accessing 2-D array elements) to perform the operations needed for the tasks in (i) and (ii). You may assume the number of rows of the matrix according to your choice.

3. **(Marks: 6)** In C programming language write a **recursive** function `calculateSum()` to find the sum of the series x - $\frac{x^3}{3!}$ + $\frac{x^5}{5!}$ - $\frac{x^7}{7!}$ +...$(-1)^m \frac{x^{(2n-1)}}{(2n-1)!}$ such that n = 1, 2, 3, ... and m is 2 if n is odd, 1 otherwise. $a!$ denotes the factorial of a non-negative integer a. Here x (*float* data type) and n (*int* data type) need to be passed as arguments to `calculateSum()`.

4. **(Marks: 8)** Consider a login interface of the Research and Analysis Wing for secret agents asking for inputs (all in CAPS): the first name (*Fname*, e.g. RAVINDRA), last name (*Lname*, e.g. KAUSHIK), code name (*Cname*, e.g. TIGER) and the highly confidential encrypted code, expected to be available exclusively to the agent (*EncryptedCode*, e.g. WNA). For double security check, given *Fname*, *Lname* and *Cname*, the system automatically generates the encrypted code (to check with *EncryptedCode*) as follows:

   1. The string *Fname* is added **character-wise** with *Lname* to generate an intermediate string *intermediate*. Note that, in C, a character, say 'A', is represented by its ASCII value which enables the addition of two characters like integers to yield another integer value. However, in this encryption scheme, we consider only capital letters (ASCII range: 65 – 90). So, if the resulting character ASCII value (say v) exceeds 90, it is rescaled to v_{new} = (v-90)%26 + 65. Consider the first letters of RAVINDRA and KAUSHIK, viz. R (ASCII value: 82) and K (ASCII value: 75) respectively, adding which gives a value v = 82 + 75 = 157 for which v_{new} = (v-90)%26 + 65 = (157-90)%26 + 65 = 80 (the ASCII value of P). Therefore, for *Fname* as RAVINDRA and *Lname* as KAUSHIK, the we get string *intermediate* as PODOIZPA. Note that, the extra characters on the right of the longer string are carried forward. See Table 1 for details.
   2. The string *intermediate* is added **character-wise** with *Cname* to generate the final string *final* (details same as in step 1). From *intermediate* as PODOIZPA and *Cname* as TIGER, we get string *final* as WKXGNZPA.
   3. Finally, the system generates encrypted code *encrypted* by selecting the first, middle (first character of the right half in case of even string length) and last characters of *final*. That is, from *final* as WKXGNZPA, we get *encrypted* as WNA.

   Write a C program to implement the above system. Take *Fname*, *Lname* and *Cname* and *EncryptedCode* as input and generate *encrypted*. If *EncryptedCode* matches with *encrypted*, print a welcome message like "Welcome Agent TIGER", else print a message indicating attempted security breach.

First name	R	A	V	I	N	D	R	A
Last name	K	A	U	S	H	I	K	
Intermediate string	P	O	D	O	I	Z	P	A
Code name	T	I	G	E	R			
Final string	W	K	X	G	N	Z	P	A
Encrypted code	W				N			A

Table 1: Example encryption scheme for Q4.