TRINITY COLLEGE DUBLIN

THE UNIVERSITY OF DUBLIN

Faculty of Engineering, Mathematics & Science
School of Computer Science & Statistics

Integrated Computer Science Programme **Trinity Term 2015**
B.A. (Mod.) Business and Computing
Year 2 Annual Examinations

Systems Programming I (CS2014)

Thursday 14th May, 2015 **Luce Upper** **14:00 – 16:00**

Dr David Gregg

Instructions to Candidates:

- Answer 2 out of the 3 questions
- All questions are marked out of 50
- All program code should be commented, indented and use good programming style

Materials permitted for this examination:

- Calculator.

1. One of the most common programming problems is to find the sum of the items in an array. The following code is perhaps the simplest solution:

```
float compute_sum(float * array, int size)
      float sum = 0.0;
      for ( int i = 0; i < size; i++ ) {
            sum = sum + array[i];
      }
      return sum;
}
```

A problem can arise with this code. Floating point numbers consist of a mantissa and an exponent. When we add floating point numbers, the computer scales the mantissa of the smaller number downwards according to the difference in the exponents. Given that floating point numbers have only a limited number of digits, the smaller number may disappear partially or completely after scaling. In the above algorithm, the *sum* variable contains the sum of all the array elements seen so far. In a large array the value of the *sum* variable can become very large, and so the numbers towards the end of the array can be lost, because they are too much smaller than the sum to count.

A good solution to this problem is to add the numbers of the array together in pairs. For example, we might add `array[0]` to `array[1]` in one pair, and `array[2]` to `array[3]` in another pair. This continues along the array until we have `size/2` sums of pairs. We then apply the same algorithm to each of the sums of pairs, to get `size/4` sums of sums of pairs. By applying this approach repeatedly, we can get a sum of the entire array, but ensure that each addition is applied to numbers that are sums of a similar number of original numbers.

(Question 1 continues on next page)...

... (Question 1 continued from previous page)

Write a C routine that computes the sum of an array of floating point numbers by repeatedly applying this pairwise sum algorithm until a single sum is left. Please comment on the time and space complexity of your algorithm.

[50 marks]

2. The Nucleic Acid Notation (NAN) uses the letters G, C, A and T to represent the four molecules that form the subunits of deoxyribonucleic acids (DNA). The molecules combine to form a long string of molecules that can be represented as a string of characters, where the characters are selected from the symbols G, C, A and T.

   The strings are sometimes very long, and thus require a lot of memory. One solution to this problem is to store the strings in a compressed format. There are only four possible symbols, so a single symbol could be stored in two bits, rather than the eight bits needed to store the corresponding character.

   Write a C routine that takes a string of symbols in standard character format and converts it to compressed format. Your function should have the following prototype:

   ```
   unsigned char * compress_nan(char * input, int * result_length);
   ```

   Your function should accept the string input and return a compressed version of the string where each symbol is packed into two bits. The compressed string should be therefore around four times shorter than that original. The length (in symbols) of the compressed string should be returned via the `result_length` pointer.

   Secondly, write a function that takes in a compressed version of a string and returns it in standard string format. Your function should have the following prototype:

   ```
   char * uncompress_nan(unsigned char * input, int input_length);
   ```

   Your function should accept input, a compressed NAN string, along with the number of symbols in that compressed string. It should return a standard NULL-terminated uncompressed string of the symbols C, G, A and T.

   [50 marks]

3. One way to represent sets of items is to use a linked list. Write a C abstract data type (ADT) to represent sets of strings. Your ADT should provide functions to:
    - create a new empty set
    - add a string to the set
    - remove a string from the set
    - check whether a given string is a member of the set
    - compute the union of two sets of strings
    - compute the intersection of two strings
    - free the memory used by the set

[50 marks]