ENGR 244 – ENGINEERING COMPUTING – FALL 2017 – TERM EXAM I

Name: _____

(please repeat **name** or **ID** on *every page*)

Student ID: _____

- You have **100 minutes** from the moment that the beginning of the exam is announced.
- This term exam consists of four problems. **Two of them will count.** Each problem carries the same weight, i.e., 7.5 credits (out of 100 credits for the whole course). At most 15 credits can be gained from the present term exam.
- If you choose to **work on two problems**, the outcome of these two problems will count. If you choose to work on three or four problems, the two best outcomes will count, and the problems with the least amount of credits obtained will be disregarded.
- Make sure that every sheet of paper you submit contains your **name** and/or **student ID**.
- Any access to means of communication, e.g., **cell phones**, constitutes a case of **cheating** irrespective of what is communicated. You may use no material except that which is necessary for writing, as well as a calculator and one A4 paper with handwritten notes.
- If you use **handwritten notes**, they need to be turned in as part of your exam material.

  (They will not be returned to you.)

In your own interest please avoid unnecessarily long answers.

TERM EXAM I – PROBLEM 1

Consider the following program:

```
#include <iostream>
using namespace std;

double sum_of_cubes(double* a, double* b, double* c)
{
  return (*a)*(*a)*(*a) + (*b)*(*b)*(*b) + (*c)*(*c)*(*c);
}

int main()
{
  double x = -2.0;
  double y = 0.5;
  double z = 3.0;

  double sum = sum_of_cubes  // this line is incomplete, part of the statement is missing

  cout << sum << "\n";
  return 0;
}
```

a) Complete the fourth statement of the main function such that the program, when executed, will write "19.125" to standard output, i.e., the sum of the cubes of -2, 0.5, and 3.

b) In the code shown above, the arguments are *passed by address* (i.e., "passed by reference"). Rewrite "sum_of_cubes" such that they are *passed by value*. What is the difference between the outcome of the original version and the outcome of your new version of the function?

How does the fourth statement of the main function have to be modified in order to pass the arguments of the "sum_of_cubes" function *by value*, rather than *by address*?

TERM EXAM I – PROBLEM 2

Consider the following program:

```
#include <iostream>
using namespace std;

int main()
{
  int x_input;
  cout << "Please give an integer x to be separated into prime factors: ";
  cin >> x_input;
  if(0 >= x_input)
  {
    cout << "x input is zero, negative, or invalid.\n";  return 0;
  }
  int x = x_input;
  int y = 2;
  int z = 0;
  cout << "x = " << x << " is the product of ";

  while(x >= y*y)  // this is the loop to be analysed here
  {
    while((x % y) == 0)  // x is divisible by y
    {
      z = z + y;
      x = x / y;
      cout << y << ", ";
    }
    if(y == 2) y = 3;
    else y = y + 2;
  }

  if(z == 0)
  {
    cout << x << " only (it is a prime number).\n";
    z = x;
  }
  else if(x == 1) cout << "and no other number.\n";
  else
  {
    cout << "and " << x << ".\n";
    z = z + x;
  }
  cout << "The sum of all prime factors is " << z << ".\n";
  return 0;
}
```

a) Assume that *x_input*, i.e., the number given by the user, is 99. What is the program output?

b) For the outer **while** loop ("loop to be analysed here"), determine the loop precondition, the loop condition, the loop invariant, and the loop postcondition, as discussed in the lecture.

TERM EXAM I – PROBLEM 3

Consider the following program:

```
#include <iostream>
using namespace std;

int* klmn_function(int k, int l, int m, int n)
{
  if(k > l) return klmn_function(l, k, m, n);
  if(m > n) return klmn_function(k, l, n, m);

  // function here is in state A(k, l, m, n), depending on the argument values

  if(k > m) return klmn_function(m, l, k, n);
  if(l > n) return klmn_function(k, n, m, l);

  // function here is in state B(k, l, m, n), depending on the argument values

  int* x = new int[4];  // create integer array with four elements, x[0] to x[3]

  x[0] = k;
  if(l < m) { x[1] = l;  x[2] = m;  }
  else { x[1] = m;  x[2] = l; }
  x[3] = n;

  return x;
}

int main()
{
  int* y = klmn_function(42, 426, 7, 89);
  cout  << y[1] << "\n";
  return 0;
}
```

a) Describe briefly what the function "klmn_function" does: What does "klmn_function" accomplish in general, i.e., for arbitrary combinations of values for the arguments *k*, *l*, *m*, and *n*?

b) What is the return type of "klmn_function"?

c) What is the output (to standard output, i.e., "cout") of the program given above?

d) Characterize, as precisely and generally as possible, the condition of the function in "state A", i.e., after the first two **if** statements, and in state B, i.e., after the first four **if** statements. Think of preconditions and postconditions as discussed in the lecture. Your answer should be valid for the function in general, i.e., for any combination of argument values *k*, *l*, *m*, and *n*.

e) Why is it wrong to write "**int** *x*[4];" instead of "**int*** *x* = **new int**[4];"? What would happen?

TERM EXAM I – PROBLEM 4

a) Write a function called "count_common_years" with a single argument, **int** *n*, and a return value which is specified as follows:

- If its argument is zero, the function count_common_years returns 0.
- For all positive values of *n*, count_common_years(*n*) is the n^{th} natural number which fulfills any of the following two conditions:
    - *n* is not divisible by 4 (i.e., its remainder after division by 4 is not 0),
    - *n* is divisible by 100 (and hence also by 4), but not by 400,

  beginning with count_common_years(1), which should be 1.

  (If you are not sure how to implement this, continue by using the first condition only.)
- For negative *n*, count_common_years(*n*) is given by 1 – count_common_years(–*n*).

  Hence, count_common_years(–1) should be 0, count_common_years(–2) should be –1.

(If you need many lines of code, this is a sign that you are not doing it in the best possible way.)

b) Discussion of *time requirements*: How many elementary operations does a call to your function require if the value of the argument *n* is given as 5? Count each assignment, arithmetic operation, and logical operation as one elementary operation.

Explain or visualize compactly how you obtain this result.

c) How does the number of elementary operations depend on *n*: How does it scale asymptotically for great values of *n*? (Here you do not need to give the exact number of operations, only the *order of* its dependence on *n*, as usual for discussions of asymptotic scaling.)

Here it is sufficient to state the correct answer; no proof or justification needs to be given.

d) How would the time requirements scale asymptotically for the *fastest possible algorithm* that solves the given problem, as a function of *n*? (You do not need to prove this, but do explain *why* you believe it. Avoid excessively long answers, two or three sentences will be enough.)