AUIS ENGR 244 (Engineering Computing), Course Assignment 1

Deadline:[1] October 9, 2017; Tutorial Discussion: October 16, 2017

Avoiding the "goto" statement

1) Rewrite the following code without "goto". Assume that *n* is an integer which has been assigned a value before (e.g., from user input). What is the meaning of the output value *y*?

```
L1:  int k = 0;
L2:  int y = 1;
L3:  if(k > n) goto L9;
L4:  std::cout << n << "\t" << k << "\t" << y << "\n";
L5:  y = y * (n - k);
L6:  k++;
L7:  y = y / k;
L8:  goto L3;
L9:  std::cout << "# Done. \n";
```

Working with functions

2) Write a function that, for $x \in \mathbb{R}$, returns $f(x) = x \sin x^{-1}$ and does not fail for the case $x = 0$. You can use the function **double** sin(**double** *x*) from the C standard library (#include <cmath>).

Working with iterations and arrays

3) Write a program that accomplishes the following tasks, in the given order:

- An array with 800 elements of type **unsigned long** is defined; all are initialized to 0.
- The series of cubic numbers, 0, 1, 8, 27, ..., 799^3, is written into the array.
- For all combinations of integer values of *i* and *j* with $0 \le i < j \le 799$, check if $j^3 - i^3$ is divisible by 8009. The cubic numbers are read from the array, not recomputed.
- An output is made, using the "cout" stream, stating how many such pairs (*i*, *j*) exist.

A construction of the following type can be used to consider all combinations of *i* and *j* values:

```
for(unsigned i = 0; i < 799; i++)
{
  for(unsigned j = i + 1; j < 800; j++)
  {
    // block of code to be executed for each pair of i and j
  }
}
```

1 *Submissions by groups of up to three people are allowed. All groups are required to work independently (no copying, etc.). Unless otherwise stated, each problem contributes 0.5% to the overall grade. Please submit the results on paper; in case of code, print the program or the part of it that is relevant to the problem. Submissions can be handed in on Monday, Oct. 9, in class, or deposited in the mailbox in room B-F2-01 by Sunday, Oct. 8.*

It is best to use a condition like "((*x* % 8009) == 0)", or similar, with the modulo operator "%".

Recursive function calls

4) A function that contains a call to itself is called a recursive function. Consider this example:

```
int recur(int a, int k)
{
  if(k == 0) return 0;
  else return recur(a, k - 1) + a*a*(2*k - 1);
}
```

Which range of values for the arguments *a* and *k* corresponds to a successful behavior of a program that includes a call to "recur(*a*, *k*)"? What happens for values outside this range?

Modify the code such that, instead of causing the program to fail (or to run indefinitely), "recur" returns zero if its arguments are outside the acceptable range.

5) For *a* and *k* chosen within the valid range determined above, what does the function "recur" compute, i.e., what is the return value of "recur(*a*, *k*)"? Rewrite the function "recur" in a way that avoids recursive function calls.

Working with pointers

6) Explain how the outcome of calling "add13ptr" differs from the outcome of calling "add13":

```
void add13ptr(unsigned* x)
{
  *x += 13;
}
```

```
void add13(unsigned x)
{
  x += 13;
}
```

7) Why can the following code lead to a program failure ("segmentation fault") at run time?

```
unsigned sum_mod(unsigned* sum, unsigned numerator, unsigned denominator)
{
  if(sum != NULL)
  {
    sum += numerator;
    return *sum % denominator;
  }
  else return numerator % denominator;
}
```

What did the programmer probably want to do, and how can the code be fixed?