CS319: Scientific Computing (with C++)

CS319 Lab 1: Numbers and Programming

23 Feb 2021

Goal To gain familiarity with the conceptsof

- basic C++ program structure;
- input and output,
- Computer representation of number – particularly `int`s, `float`s and `double`s.
- Also: some *flow-of-control* (`if`) and *loops* (`for`)

You don't have to submit anything this week; your first homework assignment will be in Lab 2 (next week).
The exercises in Q1 and Q2 can be done using an online compiler, such as `http://cpp.sh/`. However, other would take too long to run online, so must be run on your own PC.

Q1. (`if/else if/else`-statements) Write a C++ program that prompts the user to enter two integers, x, y, and then reports which in quadrant the point (x, y) is found, or if (x, y) is on an axis (i.e., one or both are zero. (Tip: see https://en.wikipedia.org/wiki/Quadrant_(plane_geometry) for a definition of quadrants *I*, *II*, *III* and *IV*.

Q2. (`for`-loops) Recall that a **prime** number is one that has *exactly* two (distinct) divisors: 1 and itself. Write a program that computes all the prime numbers between 2 and 99 using the *Sieve of Eratosthenes*. It should declare an `int` array, P, of size 100, and initialise it so that $P[i] = i$. Then, using the algorithm described in https://brilliant.org/wiki/sieve-of-eratosthenes/ set $P[i] = 0$ where i is not prime.
When complete, your code should list all primes less than 100.

Q3. The following code snippet finds the largest `int` that is correctly representable by your computer. It also computes the time taken. (Full code at Lab1-Q3.cpp).

```
18    clock_t start;
      float diff, diff_seconds;
20    start=clock();

22    int i=1;
      int j=i+1;
24    while ( i<j )
      {
26       i++;
         j=i+1;
28    }
      diff = (float)(clock()-start);
30    diff_seconds = diff/CLOCKS_PER_SEC;
      std::cout << "Overflow at i="<< i << std::endl;
32    std::cout << "Computation took " << diff_seconds
           << " seconds."  << std::endl;
```

Q3(a) Read the code carefully, and make sure you understand it. Test it, making sure you compile without any optimisations. Do the results agree with the theory covered in class?

Q3(b) There are other types of integers available in C++, for example, `short int`, `unsigned int` and `long int`. Try this program using `short int`s and `unsigned int`. Do you get the expected results?

Q3(c) Suppose you wanted to use this program to test the largest `long int` your C++ programs can represent. Estimate how long your program would take to run.

Q4. Write a programme to tries to compute the smallest `float` greater than zero that your computer can represent. For example, you could initialise a `float`, x, as 1.0. Then, for as long as your computer thinks that $x/2 > 0$, divide x by 2. When you are done, x should be a good approximation of the smallest number representable.
Does the answer given by your code agree with theory? If not, can you give a reason why?

Q5. Next we want to compute the largest `float` representable. This is a little more tricky; where as small floats are eventually rounded to zero (which is a number), large ones tend to infinity (which is not). Try a similar approach as in Question 4, but include the header file `math.h` (and include the `h`); and use the function `isfinite` to test if x is finite or not. Depending on your compiler, you may have to compile against the `math` library.