

Physics 420/580: Exercise 1

1. Download the Exercise 1 instructions and source code from the class website. You can either do this from a web browser or from the terminal using the `curl` command.

```
$ WEBPATH=http://www.ualberta.ca/~kbeach/phys420_580
$ curl $WEBPATH/docs/Exercise1.pdf -o Exercise1.pdf
$ curl $WEBPATH/src/Exercise1.tar.gz -o Exercise1.tar.gz
$ acroread Exercise1.pdf &
$ gunzip Exercise1.tar.gz
$ tar -xf Exercise1.tar
$ cd Exercise1
```

The pdf instructions can be opened using either `acroread` or `xpdf`. The file ending in `.tar.gz` is an archived and compressed directory of files (sometimes called a *tarball*).

2. There are two programs in the `Exercise1` directory. Typing `make` will cause both of them to compile under `g++`.

```
$ make
g++ -o fp fp.cpp -O2
g++ -o properties properties.cpp -O2
```

The `-o` flags specify the names (`fp` and `properties`) of the resulting executables. You might want to type `more makefile` to see how the automated compilation is set up.

3. Explore the behaviour of single-precision floating point numbers. The `properties` program uses the C++ class `numeric_limits` to report the properties of the `float` data type provided by your computer.

4. The program `fp` illustrates some of the limitations of the floating point types. It outputs four data files: `growth.dat`, `decay.dat`, `log.dat`, and `linear.dat`.

(a) The first two files show how unit values of `float`, `double`, and `long double` evolve under repeated operation by `*=10` and `*=0.1`. What's different about the approach to ∞ and the approach to 0? How do the maximum and minimum values compare to those given by `numeric_limits<typename>::max()` and `min()`?

(b) The second two files contain a double-precision calculation of $\log(1-x)/x$ and $1-x$ over the ranges $[-1, 1]$ and $[-5 \times 10^{15}, 5 \times 10^{15}]$. Try making the following plots:

```
$ ./fp
$ gnuplot
> plot [] [-5:0] "log.dat" index 0 using 1:2 with lines
> plot [] [-3:1] "log.dat" index 1 using 1:2 with lines
> plot "linear.dat" index 0 with lines
> plot "linear.dat" i 1 w l
> quit
```

On the larger scale, the plots look smooth. On the zoomed-in scale, there is a great deal of structure. Where does that come from?

- (c) Write a program that produces the following floating point exceptions.

Exception	Cause	Result
overflow	too large to represent	$\pm\text{Inf}$
underflow	too small to represent	0
divide by zero	computing $x/0$, where x is finite and nonzero	$\pm\text{Inf}$
invalid	$\text{inf} - \text{inf}$, $0 \times \infty$, ∞/∞ , $0/0$, $\sqrt{-1}$	NaN (not a number)

(d) Show that +0 and -0 both exist and are two different floating point numbers. Why is that? What purpose does -0 serve?

5. Write a C++ program that performs the summation

$$S_N = \sum_{n=1}^N \frac{1}{n}$$

in both *increasing* and *decreasing* order. Use single-precision floating point arithmetic and report your results for $N = 100\,000\,000$. Explain the large discrepancy between the two values. Which is more accurate and why?

6. Archimedes's approximation to π involves circumscribing a circle with polygons of successively larger order. The length of the polygon perimeter tends to the circumference of the circle. This limit can be implemented as $\lim_{n \rightarrow \infty} 6 \times 2^n \times t_n = \pi$, where t_n is defined by the recurrence relation

$$t_0 = \frac{1}{\sqrt{3}}, \quad t_{n+1} = \frac{\sqrt{t_n^2 + 1} - 1}{t_n}$$

or, equivalently, by

$$t_0 = \frac{1}{\sqrt{3}}, \quad t_{n+1} = \frac{t_n}{\sqrt{t_n^2 + 1} + 1}.$$

Write a simple C++ program that outputs in two columns the order- n approximation to π given by these two relations. The first several lines of output should read

0	3.464101615137754	3.464101615137754
1	3.215390309173471	3.215390309173472
2	3.159659942097494	3.159659942097501
3	3.146086215131401	3.146086215131435
4	3.142714599645314	3.142714599645369

Compare these results to the exact value, available as `M_PI` in the `cmath` library. Discuss the accuracy of the two approaches. Explain why one fails and the other does not.

7. An N -bit binary register can hold 2^N unique bit patterns. To represent an unsigned integer, the bit patterns are interpreted as the base-2 numbers 0 to $2^N - 1$. There are several possible ways to represent signed integers. Nonetheless, on all modern computers this is accomplished using the *two's complement* scheme, in which the the numbers -2^{N-1} to $+2^{N-1} - 1$ are represented by the same bit patterns reinterpreted as 0 to $+2^{N-1} - 1$ and -2^{N-1} to -1 in lexical order.

The table below illustrates the unsigned (left) and signed (right) values for $N = 8$:

127	0	1	1	1	1	1	1	1	127
126	0	1	1	1	1	1	1	0	126
3	0	0	0	0	0	0	1	1	3
2	0	0	0	0	0	0	1	0	2
1	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	0	0
255	1	1	1	1	1	1	1	1	-1
254	1	1	1	1	1	1	1	0	-2
253	1	1	1	1	1	1	0	1	-3
129	1	0	0	0	0	0	0	1	-127
128	1	0	0	0	0	0	0	0	-128

The advantage of two's complement numbers is that they can be added in exactly the same way as conventional binary numbers:

.	.		1	1			1	
	0	1	0	1	1	0	0	1
+	1	0	0	1	1	1	0	1
	1	1	1	1	0	1	1	0

Here, red ones represent the carry bits. The computation above can be interpreted as either $89 + 157 = 246$ or $89 - 97 = -8$.

- (a) Devise a rule for two's complement negation that maps $x \rightarrow -x$.
- (b) There is one nonzero value that behaves as $x \rightarrow x$ under negation. What is it?
- (c) There are four possible values for the two leftmost carry bits (marked with dots): 00, 01, 10, 11. Which combinations denote an overflow for unsigned binary and which for signed (two's complement) binary?

8. Change the octal numbers 12_8 , 5655_8 , 2550276_8 , 76545336_8 , and 3726755_8 to hexadecimal notation using the digits $0, 1, \dots, F$. Hint: It may be easier to use the base-2 bit patterns as an intermediate step.
9. We can generalize a positional number system to negative bases by writing

$$(\dots a_3 a_2 a_1 a_0 . a_{-1} a_{-2} \dots)_b = \dots + a_2 b^2 + a_1 b + a_0 + a_{-1} b^{-1} + \dots$$

in the usual way but requiring that the digits be in the range $0 \leq a_k < |b|$. For example,

$$\begin{aligned} 1322_{-4} &= -22 \\ 133.21_{-5} &= 12.64 \\ 2050.23_{-6} &= -462.25 \\ 6231.46_{-8} &= -2967.40625 \end{aligned}$$

- (a) Negate each of the identities above.
- (b) Is there a unique negative base representation for each number?
- (c) Is it ever necessary to have a leading minus sign, e.g., -54_{-3} , when the base is negative?

10. Consider a 6-digit *factoradic* system

$$(a_6 \dots a_3 a_2 a_1) = a_6 \cdot 6! + \dots + a_3 \cdot 3! + a_2 \cdot 2! + a_1, \quad \text{where } 0 \leq a_k \leq k$$

- (a) What are the largest and smallest numbers representable?
 (b) Is every number in between representable?

11. Imagine a hypothetical computer that stores floating points numbers as a sign bit along with (offset) exponent and fraction parts comprising two and five *octal* digits, respectively:

\pm	e_2	e_1	f_5	f_4	f_3	f_2	f_1
-------	-------	-------	-------	-------	-------	-------	-------

If we denote this floating point scheme by $(\pm, e, f) = \pm(0.f \times 10^{e-37})_8$, then the largest and smallest (nonzero) positive numbers are

$$\begin{aligned} (+, 77, 77777) &= (.77777 \times 10^{40})_8 \\ &= 0.999969 \times 8^{32} \\ &= 7.92257 \times 10^{28} \end{aligned}$$

and

$$\begin{aligned} (+, 00, 00001) &= (.00001 \times 10^{-37})_8 \\ &= 0.0000305176 \times 8^{-31} \\ &= 3.08149 \times 10^{-33} \end{aligned}$$

The number $(+, 00, 10000) = (.10000 \times 10^{-37})_8$ is the smallest positive number that retains five significant digits. Everything smaller has a leading zero in the fractional part and is called *denormalized*.

Consider the numbers

$$\begin{aligned} a &= 454592 = (.15677 \times 10^7)_8 = (+, 44, 15677) \\ b &= 30150656 = (.16301 \times 10^9)_8 = (+, 46, 16301) \\ c &= -451712 = (.15622 \times 10^7)_8 = (-, 44, 15622) \end{aligned}$$

The addition of a and b is carried out as follows.

$$\begin{aligned} a + b &= (+, 44, 15677) + (+, 46, 16301) \\ &= (+, 48, 00157) + (+, 46, 16301) \\ &= (+, 48, 16460) \end{aligned}$$

- (a) Compute $a + c$ and $b + c$ in the same vein.
 (b) Work out the steps for the multiplication operation and compute $a \times b$, $a \times c$, and $b \times c$.
 (c) Comment on the loss of significant digits in these operations.

12. Let's invent a 20-bit floating-point representation that has 1 sign bit, seven exponent bits (offset 63), and twelve fraction bits. This representation is simplified in that it has no reserved values (for **inf**, **nan**, etc.) and only properly normalized numbers are used. As usual, the leading significand bit is hidden.

(a) In base 2,

$$\pi \doteq 11.001001000011111101101010100010\dots$$

How is this number stored in the 20-bit floating-point format described above? (Use rounding rather than truncation.)

(b) What are the largest and smallest positive values that can be represented?

(c) Is it possible to represent zero?