Exercise 2

Introduction to Computational Astrophysics, SoSe 2024

Seha Lee

Task 1. *Integer representation*

Solution. (a) $2^{10} - 1 = 1023$
(b) The largest positive integer has 0 for its most significant digit to indicate it's positive, and then the rest is 1 to be the largest, i.e., $n_{max} = 01111111111111111111111111111111_2$. If you add 1 to this value, the result is 33 digit number, the most significant digit being 1 and the rest being 0, i.e., $n_{max} + 1 = 10000000000000000000000000000000_2$. Utilizing two's complement we can avoid the redundancy of zeros and inefficiency of computations caused by the use of sign bit.

□

Task 2. *Floating point number representation*

Solution. (a)
The completed code:

```c
#include <stdio.h>

int main() {
   float y, x = 1.043E-13 ;
   x = x / 10. ;
   y = x*x - (1.043E-14)*(1.043E-14) ;

   printf("x= %.15f\n", x);
   printf("y= %.15f\n", y);

   return 0;
}
```

It was expected that x and y are 1.043E-14 and 0 respectively. However, computed results are x= 0.000000000000010 and y= -0.000000000000000. There is a clear discrepancy between expected and actual computed result for y, which is a very small negative number close to zero but not exactly zero. This is due to the limitation of computer memory using only finite number of bits to store real numbers, leading to inaccuracy in floating point arithmetic as in this case,

resulting in catastrophic cancellation.

(b)
The bit patterns for the numbers expressed by standard IEEE 754 is as following, the sign bit, exponent, and mantissa respectively:

- 7 is $0 \quad 10000001 \quad 11000000000000000000000$
- $1.E-7$ is $0 \quad 01110111 \quad 00001010001111010111000$.
- The sum of 7 and $1.E-7$ retains the bit pattern of 7. This is due to absorption.

□

Task 3. *GRS bits*

Solution. (a)
$32 - 2.125 = 29.875$

(b)

$$
\begin{array}{rr}
1.000\,000 \times 2^5 & \\
-\,0.000\,101 \times 2^5 & \text{GRS } 101 \\
\hline
0.111\,011 \times 2^5 & \\
1.110\,11 \times 2^4 & \text{shifted left} \\
1.111 \times 2^4 & \text{rounded}
\end{array}
$$

□

Task 4. *Radius calculation*

Solution. By using the relative value to the Sun, we do not have to introduce constants with extreme values such as σ_{SB}, avoiding potential problems of floating point arithmetic that can lead to inaccurate results.
The code is:

```cpp
#include <iostream>
#include <cmath>

int main() {
    const double tempSun = 5778;
    const double radiusSun = 695700; // km

    double logLsun;
    double Teff;

    std::cout << "Enter log(L*/Lsun): ";
    std::cin >> logLsun;
```

```
    std::cout << "Enter Teff (in Kelvin): ";
    std::cin >> Teff;

    double radius = sqrt(pow(10, logLsun) * pow(tempSun / Teff, 4));

    std::cout << "The radius is: " << radius << " R_sun" << std::endl;

    return 0;
}
```

□