Anmol Chaure
Prof. Michal Kuraz's Intro to Fortran
October 15, 2025

Homework 2

Task 1. Write a Fortran program that:

1) Declares two integers of different kinds, e.g. integer(kind=4) and integer(kind=8).
2) Computes the factorial n! in a loop from n = 1 upwards.
3) Stops when an overflow occurs (e.g. when the value becomes negative or smaller than the previous one).
4) Prints the last correct value and the n where overflow happened.

Observations

Every data type, in this case integer has a specified bit sized memory allocated for its value. An **Integer overflow** occurs when a calculation exceeds this specified allocation it causes the value to wrap around to a smaller number or result in undefined behavior (like **negative numbers in this case**). In this task integer overflow was demonstrated by taking factorials of two kinds of integer, namely integer **kind=4** and **kind=8**, each of them has **4 bytes (32 bits) and 8 bytes (64)** of memory respectively.

For **kind=4, program overflows at 14** by repeating the same result, for **kind=8, overflows at 21** by resulting in negative numbers. Therefore, precisely demonstrating how memory allocation affects overflow.

```fortran
program overflow_factorial
    implicit none

    integer(kind=4):: fact4, previous_fact4
    integer(kind=8):: fact8, previous_fact8
    integer :: n

    !initializing to avoid error
    fact4 = 1
    fact8 = 1
    previous_fact4 = 1
    previous_fact8 = 1

    do n =1,50
        if(n>1) then
            previous_fact4 = fact4
            fact4 = fact4*n
            if(fact4 < previous_fact4) then
                print*, "Oveflow Fact4 at n =", n
                exit
            end if
        end if
    end do

    do n=1,50
        if(n>1) then
            previous_fact8 = fact8
            fact8 = fact8*n
            if(fact8<previous_fact8) then
                print*, "Overflow Fact8 at n=", n
                exit
            end if
        end if
```

```
    end do
end program overflow_factorial
```

Listing 1. Code to demostrate Integer Overflow

```
           n       fact4
           2           2
           3           6
           4          24
           5         120
           6         720
           7        5040
           8       40320
           9      362880
          10     3628800
          11    39916800
          12   479001600
          13  1932053504
          14  1278945280
Oveflow Fact4 at n =          14
```

```
           n                 fact8
           2                     2
           3                     6
           4                    24
           5                   120
           6                   720
           7                  5040
           8                 40320
           9                362880
          10               3628800
          11              39916800
          12             479001600
          13            6227020800
          14           87178291200
          15         1307674368000
          16        20922789888000
          17       355687428096000
          18      6402373705728000
          19    121645100408832000
          20   2432902008176640000
          21  -4249290049419214848
Overflow Fact8 at n=          21
```

Figure 1. Demonstration of integer overflow

Task 2. Write a program similar to the in-class example:

1) Start with big = 1.0d0 and small = 1.0d0.
2) In each loop iteration:
   - compute sum = big + small,
   - print small, big, and sum,
   - divide small by 2.
3) Stop when adding small no longer increases big

Observations

Loss of significance is an undesirable effect in calculations using floating-point arithmetic. Computer systems use floating point arithmetic to express fractional numbers. In this code we try to demonstrate the loss of significance or precision. To be exact, the program shows that when adding a very small number(small) to a much larger one(big), the result of their addition stops changing as the system cannot represent the difference in numbers after a certain limit called **machine epsilon**. This happens due to the finite number of bits available to store floating point numbers which limits precision of real numbers.

In this case, the limit reaches at , therefore, we can say that it is our machine epsilon.

```fortran
program significance
    implicit none


    real(kind=8) :: small,big, summ
    integer :: i,d

    big = 1.0d0
    small = 1.0d0

    do i=1,1000
        summ = big+small
        print*, small, big, summ
        if(summ<= big) then
            print *, 'Adding small has no effect on big at iteration', i
            exit
        end if
        small = small/2.0d0
    end do
end program significance
```

Listing 2. Code to demostrate Integer Overflow

Task 3. Reflect

1) **What numerical property limits integer types?**

   Integer types are limited by their **range**, which is determined by the number of bits used to store them. For example, a signed 32-bit integer can represent values from -2,147,483,648 to 2,147,483,647.

2) **What numerical property limits real types?**

   The numerical properties that limit real types include their **maximum values and precision**, which are determined by the specific implementation and the number of bits allocated for storage. Typically, single precision real numbers can represent about seven decimal digits, while double precision can represent around fifteen decimal digits.

3) **Which error (overflow or precision loss) is more dangerous for scientific simulations,and why?**

   Precision loss is generally more dangerous for scientific simulations because it can lead to significant inaccuracies in calculations, especially when small changes in input can result in large discrepancies in output. Overflow can often be detected and handled, while precision loss may go unnoticed, compromising the reliability of the results.

Task 4. Bonus

Implement a single program that performs both experiments sequentially: first integer factorial overflow, then real precision loss. Print the number of iterations before overflow and before significance loss. Comment how these limits correspond to the bit length of the data types used.

```
integer overflow
 n                    factorial(kind=4)
           2           2
           3           6
           4          24
           5         120
           6         720
           7        5040
           8       40320
           9      362880
          10     3628800
          11    39916800
          12   479001600
          13  1932053504
          14  1278945280
 overflow at n =          14
```

```
 loss of signicance (kind =4)
small                 big                 sum
1.00000000        1.00000000        2.00000000
0.500000000       1.00000000        1.50000000
0.250000000       1.00000000        1.25000000
0.125000000       1.00000000        1.12500000
6.25000000E-02    1.00000000        1.06250000
3.12500000E-02    1.00000000        1.03125000
1.56250000E-02    1.00000000        1.01562500
7.81250000E-03    1.00000000        1.00781250
3.90625000E-03    1.00000000        1.00390625
1.95312500E-03    1.00000000        1.00195312
9.76562500E-04    1.00000000        1.00097656
4.88281250E-04    1.00000000        1.00048828
2.44140625E-04    1.00000000        1.00024414
1.22070312E-04    1.00000000        1.00012207
6.10351562E-05    1.00000000        1.00006104
3.05175781E-05    1.00000000        1.00003052
1.52587891E-05    1.00000000        1.00001526
7.62939453E-06    1.00000000        1.00000763
3.81469727E-06    1.00000000        1.00000381
1.90734863E-06    1.00000000        1.00000191
9.53674316E-07    1.00000000        1.00000095
4.76837158E-07    1.00000000        1.00000048
2.38418579E-07    1.00000000        1.00000024
1.19209290E-07    1.00000000        1.00000012
5.96046448E-08    1.00000000        1.00000000
 Addition has no effect          25
```

Figure 2. Demonstration of integer overflow and loss of significance