Assignment 4: Legacy Software: Fun with Languages (30%)

Choose **one** of the following topics.

Requirements for the assignment are specified at the end of each topic.

1. CALCULATING *e*

The constant *e*, whose value is 2.7182..., was discovered by Swiss mathematician Jacob Bernoulli in 1683. The letter *e* was introduced in the early 18th century by Swiss mathematician Leonard Euler. Euler was trying to solve a problem first proposed by Bernoulli, half a century earlier. The problems were related to compound interest. Euler managed to calculate the value of *e* to 23 decimal places. *e* is an irrational real number, which means that it cannot be written as a fraction, and that its decimal expansion goes on forever with no repeating block of numbers that continually repeats.

Calculating numbers like π and e to a high number of significant digits. There is likely very little value in this except to explore the digital sequence produced. e is normally calculated using the well known series:

$$e = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \ldots.$$

Most programming languages can really only accommodate about 10-15 decimals of accuracy. An early algorithm for this was published in 1968, "The calculation of *e* to many significant digits", by A.H.J. Sale. The algorithm provided was written in ALGOL 60 (The Computer Journal, 11(2), pp.229-230, 1968).

TASK

You are provided with an algorithm to calculate *e* written in ALGOL 60.

1. Translate the ALGOL 60 algorithm for the procedure `ecalculation()` to Fortran, Ada, C, and Python. Write a program wrapper, or main program, for each language.

2. For each language, modify the code so that a the name of a file to store the value of *e* generated can be input by the user.

3. Test each of the languages using the criteria described below.

TESTING

You can test the accuracy of the programs, by comparing the value of e in the user specified file against a file containing the first 100 decimal places of π (perhaps using **diff**).

REFLECTION REPORT

Include a succinct 2 page summary (12-point, single-spaced).

Describe your experiences implementing the algorithm in each of the languages: C, Fortran, Ada, Python. This could include a list of the benefits/limitations of each of the languages.

Also answer the following question:
- Was there any difference in accuracy? (Show using various test data, in table form).

DELIVERABLES

- A Fortran, Ada, C, and Python program, and the reflection report.

The algorithm

```
procedure ecalculation (n, d);
value n;
integer n;
integer array d;
comment This procedure for calculating the transcendental
   number e to n correct decimal places uses only integer
   arithmetic, except for estimating the required series
   length. The digits of the result are placed in the array
   d, the array element d[0] containing entier(e), and the
   subsequent elements the following digits. These digits
   are individually calculated and may be printed one-by-
   one within the for statement labelled 'sweep'.
   begin integer m;
   real test;
   m := 4;
   test := (n + 1) × 2·30258509;
   loop: m := m + 1;
   if m × (ln(m) − 1·0) + 0·5 × ln(6·2831852 × m)
      ⩽ test then go to loop;
   begin integer i,j,carry,temp;
      integer array coef [2 : m];
      for j := 2 step 1 until m do coef [j] := 1;
      d [0] := 2;
      sweep: for i := 1 step 1 until n do begin
         carry := 0;
         for j := m step −1 until 2 do begin
            temp := coef [j] × 10 + carry;
            carry := temp ÷ j;
            coef[j]:= temp − carry × j
         end of digit generation;
         d [i] := carry
      end having calculated n digits
   end deleting declarations
end of ecalculation;
```

2. THE GREATEST COMMON DIVISOR

Some computing tasks require the calculation of the Greatest Common Divisor, or GCD. The most common algorithm for doing this is that of Euclid. This algorithm is one of the worlds oldest algorithms, stemming from *Elements*, written by Euclid in 300 BC. Euclid's algorithm is commonly provided as a recursive algorithm of the form:

```
EUCLID(a,b)
if b = 0
   return a
else
   return EUCLID(b, a mod b)
```

The problem is the effect of recursion on the running time of the algorithm. The overall running time is usually proportional to the number of recursive calls that have to be made.

An alternative to Euclid's algorithm for deriving the GCD is an algorithm published by Josef Stein in 1967. It can be implemented using binary recursion and has increased efficiency over Euclid's algorithm by replacing divisions and multiplications with shifts.

His binary algorithm is based on these properties:

$$\gcd(0,y) = y, \text{ and } \gcd(x,0) = x$$

together with these additional properties for all $x > y > 0$:

`x` and `y` are both even	`gcd(x,y) = 2·gcd(x/2,y/2)`
`x` is even, `y` is odd `x` is odd, `y` is even	`gcd(x,y) = gcd(x/2,y)` `gcd(x,y) = gcd(x,y/2)`
`x` and `y` are both odd, and `x≥y`	`gcd(x,y) = gcd((x-y)/2,y)`
`x` and `y` are both odd, and `x<y`	`gcd(x,y) = gcd((y-x)/2,x)`

- Stein, J., "Computational problems associated with Racah algebra", *Journal of Computational Physics*, Vol. 1(3), pp.397-405, (1967)

TASK

You are provided with a C program for Stein's GCD algorithm, and Euclid's non-recursive algorithm.

1. Translate Stein's GCD algorithm from C to Ada, Fortran, and Python.

2. Translate Euclid's non-recursive GCD algorithm from C to Ada, Fortran, and Python.

3. Write a version of Euclid's recursive GCD algorithm for C, Ada, Fortran, and Python.

4. Test each of the languages/algorithms using the criteria described below.

Note that all three algorithms for each language can be placed in a single file. For example **gcd.c** could contain the functions **stein_GCD**(), **euclidR_GCD**(), and **eucluidNR_GCD**(). The same for Python, Fortran, and Ada.

TESTING

Test the algorithms in each language against one another.

1. Create an array (of length, say, 3000) of random integers, uniformly distributed from 1 to the largest representable value (as a long in C, you will have to figure out similar data types in the other languages).
2. Compute the gcd of the first and second number, the second and third number, the third and fourth, and so on, until all numbers from the array have been used.
3. Time this process.

REFLECTION REPORT

Include a succinct 2 page summary (12-point, single-spaced).

Describe your experiences implementing the algorithm in each of the languages: C, Fortran, Ada, Python. This could include a list of the benefits/limitations of each of the languages.

Also answer the following question:

- Was there any difference in speed? (Show using various test data, in table form).

DELIVERABLES

- A Fortran, Ada, C, and Python program, and the reflection report.

CODE

Euclid's Algorithm (non-recursive)

```
int gcd(long x, long y)
{
    long r;
    r = x % y;
    while (r != 0){
       x = y;
       y = r;
       r = x % y;
    }
    return y;
}
```

Stein's Algorithm (binary GCD)

```
int gcd(long x, long y)
{
    if (x == y)
        return x;
    if (x == 0)
        return y;
    if (y == 0)
        return x;

    // look for factors of 2
    if (~x & 1) // x is even
        if (y & 1) // y is odd
            return gcd(x>>1, y);
        else // both x and y are even
            return gcd(x>>1, y>>1) << 1;
    if (~y & 1) //x is odd, y is even
        return gcd(x, y>>1);

    // reduce larger parameter
    if (x > y)
        return gcd((x - y)>>1, y);
    return gcd((y - x)>>1, x);
}
```

3. RUSSIAN PEASANT MULTIPLICATION

"Russian-peasant" multiplication (it was actually used as early as 1800BC by Egyptian mathematicians) is an algorithm for performing multiplication by repeated halving, doubling, and additions. It is well-suited to multiplications of large numbers, such as numbers whose product has up to twice as many digits as the calculator's normal word size.

Given this equation: **multiplier × multiplicand = product** ,
The basic algorithm is:

1. Set product to 0.
2. If **multiplier** is odd, add **multiplicand** to product.
3. Halve **multiplier** (i.e., divide it by 2) and double **multiplicand** (i.e., multiply it by 2). Note that integer division is used, so remainders are discarded.
4. Repeat steps 2 and 3 until **multiplier** reaches 1

How does this work? For example: `7 × 9`

```
1. p = 0
2. 7 is odd, so p = 9
3. 7÷2 = 3, 9×2 = 18
   1. 3 is odd, so p = 9+18 = 27
   2. 3÷2 = 1, 18×2 = 36
      1. 1 is odd, so p = 27+36 = 63
```

Here is a (recursive) algorithm for the process to calculate `p = m x n`:

```
integer p(m, n)
begin
    case 1: (m=0)
        return 0
    case 2: (m=1)
        return n
    case 3: (m>1 and (m mod 2)=0)
        return p((m div 2), (n + n))
    case 4: (m>1 and (m mod 2)=1)
        return n + p((m div 2),(n + n))
end
```

TASK

The pseudocode is provided for a recursive version of the algorithm.

1. Translate the recursive algorithms into Fortran, Ada, and C.
2. Write a non-recursive version of the algorithm and code them in the same three languages and into Cobol (as Cobol does not allow recursion), and compare the implementations.

This assignment is an exercise in code translation and language bench-marking. Compare execution speeds of each of the languages.

TESTING

Test your program, multiplying some larger numbers. For example:

2,345 * 789 = 1,850,205
23,958,233 * 5,830 = 139,676,498,390.

REFLECTION DOCUMENT

Include a succinct 2 page summary (12-point, single-spaced).

Describe your experiences implementing the algorithm in each of the languages: C, Fortran, Ada. This could include a list of the benefits/limitations of each of the languages.

Also answer the following questions:
- Was there any difference in efficiency? (Show using various test data, in table form). Which language was the slowest and why do you think?
- Were some languages better able to deal with multiplying large numbers?

REFS

- Spickerman, W.R., "A note on the Russian peasant multiplication algorithm", School Science and Mathematics, 71(7), pp.628-629 (1971).
- Gimmestad, B.J., "The Russian peasant multiplication algorithm: A generalization", The Mathematical Gazette, 75(472), pp.169-171 (1991).
- Bowden, J., "The Russian peasant method of multiplication", The Mathematics Teacher, 5(1), pp.4-8 (1912).
- Boykin, W.E., "The Russian-peasant algorithm: rediscovery and extension", The Arithmetic Teacher, 20(1), pp.29-32 (1973).

WRITE-UP INFORMATION
(FOR ANY CHOICE)

REFLECTION REPORT

See specifications under each topic description.

DELIVERABLES

The submission should consist of the following items:

- The reflection report (PDF).
- The code (well documented and styled appropriately of course).
- Both the code and the reflection report can be submitted as a ZIP, TAR, or GZIP file.

SKILLS

- legacy programming, program comprehension, problem solving, language comparison