École normale supérieure
L3 geosciences
Algorithmics and Fortran exercises

Lionel GUEZ

April 12, 2022

Contents

Create a directory where you will place all your Fortran files. Copy the file GNUmakefile to this directory.

You can refer to the Fortran 2003 standard: `https://wg5-fortran.org/N1601-N1650/N1601.pdf`. In particular, the description of standard intrinsic procedures will be useful: § 13.7, page 316.

1 Identifiers

Below is a list of words. Which ones cannot be used as Fortran identifiers? Why? Some of the words are valid identifiers but it is still advised not to use them. Can you spot them and tell why they are not recommended?

- température
- pression1
- `gcm_model_hybrid_sigma_pressure_coordinate`
- `0_vector`
- count
- `_class`
- i
- character

2 Integer versus real

1. What would you see on screen from the following statements?

   ```
   print *, 11 / 5
   print *, 11. / 5
   print *, 11 / 5.
   print *, 11. / 5.
   ```

2. The execution of an assignment statement is actually a two-step process: evaluation of the expression on the right-hand side; assignment to the variable on the left-hand side. Keeping this in mind, tell what you would see on screen from the following statements.

   ```
   integer i
   real x
   i = 11 / 5
   print *, i
   i = 11. / 5
   print *, i
   x = 11 / 5
   print *, x
   x = 11. / 5
   print *, x
   ```

3. Let us stress the difference between mathematical notation and Fortran statements. With the following Fortran declarations :

   ```
   integer i, j
   real x
   ```

   what expression would you write in Fortran to produce the following values, written with usual mathematical notations ($\lfloor \ \rfloor$ for integer floor), assuming i and j are positive: $\lfloor x \rfloor$, $\left\lfloor \frac{i+j}{2} \right\rfloor$, $\frac{i+j}{2}$, j/i?

4. Finally, let us stress the static typing in Fortran. What would you see on screen from the following statements (compare with Python analoguous statements)?

   ```
   integer i, j, k
   i = 11
   j = 5
   i = real(i)
   print *, i
   k = real(i) / real(j)
   print *, k
   ```

3 Literal real values, printing to standard output

Write a Fortran program which does the following assignments and computations:

$$\mathtt{BVSEC} = 10^{-5}$$

$$\mathtt{TTLL} = 260$$

$$\mathtt{TTLLM1} = 258$$

$$\mathtt{RD} = 287$$

$$\mathtt{RCPD} = \frac{7}{2}\mathtt{RD}$$

$$\mathtt{HO} = 7 \cdot 10^3$$

$$\mathtt{ZHLL} = 3 \cdot 10^4$$

$$\mathtt{ZHLLM1} = 3{,}2 \cdot 10^4$$

$$\mathtt{BVLL} = \sqrt{\max\left[\mathtt{BVSEC}, \frac{(\mathtt{TTLL} + \mathtt{TTLLM1})\mathtt{RD}^2}{2 \times \mathtt{RCPD} \times \mathtt{HO}^2} - \frac{\mathtt{TTLL} - \mathtt{TTLLM1}}{\mathtt{ZHLL} - \mathtt{ZHLLM1}}\frac{\mathtt{RD}}{\mathtt{HO}}\right]}$$

and finally prints the value of BVLL. All the data objects above should be declared of real type. Compute bvll with a single statement, but spread the statement onto several lines so as to keep the number of characters per line below 80. In all your programs, you should always take care of writing lines with less than 80 characters or so, as this improves the readability of your code.

You should obtain:

$$\mathtt{bvll} \approx 2{,}178\,14 \cdot 10^{-2}$$

Be careful how you write literal real values and how you write the square of a value.

4 Using the character type

1. What would you see on screen from the following statements?

   ```
   character(len = 8) mot
   print *, mot
   ```

2. How would you fill the variable mot with whitespace?
3. What would you see on screen from the following statements (assuming the previous declaration of mot)?

   ```
   mot = "a"
   print *, "a", "b"
   print *, mot, "b"
   ```

4. What would you see on screen from the following statements (continuing from the statements of the previous questions)?

   ```
   mot = mot // "version"
   print *, mot, "!"
   print *, mot // "version", "!"
   ```

5. How would you append the string `"version"` to the letter `"a"` in variable mot so that mot contains `"aversion"`?

5 Reading from standard input

Write a program to convert from francs to euros. The user of the program should input the amount in francs and the program should ouput the amount in euros. Do not use a new variable for the amount in euros. Exchange rate: 1 € = 6,559 57 F.

6 Selection, character type, named constant, stop statement

Modify the program you wrote in the previous exercise so it can also convert from euros to francs. The user should enter the amount followed by the currency code. The currency code is made of three upper case letters: EUR for euros and FRA for francs. The program should recognize the input currency code and convert to the other currency. The program should print an error message if the input currency is not one of the two expected currencies.

In your program, declare the exchange rate as a named constant.

7 Complicated logical expression

Write a Fortran program which tells whether a given year is a leap-year. The user should enter a year and the program should print the answer: leap-year or not.

Remember that a year can only be a leap-year if it is a multiple of 4. However, a year multiple of 100 is only a leap-year if it is also a multiple of 400.

In your program, reduce the decision process above into a single logical expression, not using any intermediary variable. Use this logical expression in a single selection structure.

Use the `modulo` intrinsic function.

8 Iteration

The Fibonacci sequence is defined by the recurrence relation :

$$\begin{aligned} u_0 &= 1 \\ u_1 &= 1 \\ u_n &= u_{n-1} + u_{n-2} \quad \text{for } n \geq 2 \end{aligned}$$

1. Write a program that prompts the user for an index value n and writes u_n. The program will then prompt the user for a new index value n, and perform a new computation. The program will stop if the user enters a strictly negative value of n.
2. Does the computation remain valid when n is large? How can the program ensure that the computation is valid? If the user enters a value of n which is too big then the program should carry the computation to the largest possible index i, print a message signaling that n is too big, and print the last computed term u_i. The program will then ask for a new value of n.

Use the `huge` intrinsic function.

9 Finite precision, order of operations on real numbers

Consider the series defined by the partial sum:

$$S_n = \sum_{i=1}^{n} \frac{1}{i^2}$$

The sum of this series is $\pi^2/6$.

In order to compute S_n, in which order should we add the terms?

Write a program which computes and prints $\pi^2/6$ and also prints two values of S_n, for a given n, by adding terms in both orders. Do not limit the value of n: just let the program add all the terms for i between 1 and n, whatever the input value of n. How can we ensure that we do not overflow the maximum integer value when we compute i^2? (Is it useful to carry out the computation of i^2 *exactly*, that is as the square of an integer number?) If we add the terms in ascending order of i, what is the maximum useful value of n that improves the approximation? In the program, compute this maximum value of n and print it. But just print it as an information to the user, do not use it as a limit to the value of n. Try the computation (in both orders) for $n = 10^4$ and $n = 10^5$, compare the results. Use the following intrinsic functions: `sqrt`, `epsilon`, `real`, `acos`.

10 Arrays

1. Write a program which computes the day number in the year. The user should input the year, month and day in the month. Take leap-years into account. Use the intrinsic function sum instead of writing a loop.

2. Write a program which does the reverse operation. The user should input the year and day number in the year and the program should output the corresponding month and day in month.

11 Array element order

Suppose we need to input a size-3 square matrix in our program. We want the user of the program to input the values of the matrix in the terminal in a natural way: one line of the matrix after another, separated by newline. How should we program the input of the matrix? Can we just use a read statement with the whole matrix?

Write the program that reads the matrix and, as a test, prints back:

- the whole matrix, using just the whole array in the print statement;
- the whole matrix, line by line;
- the second column of the matrix.

12 Allocatable arrays

Write a program which asks the user for an integer value $n \geq 2$ and prints all the prime numbers between 2 and n. The program should compute the prime numbers using the sieve of Eratosthenes.

The idea of the sieve of Eratosthenes is to cross non-prime numbers below n in the following way: starting at 2, and ascending, if you find a number which has not yet been crossed, then it is a prime number and you cross all its multiples starting at the square of this number. At the end of this process, the non-crossed numbers below n are all the prime numbers below n. You should take into account the fact that a number may already have been crossed as a multiple of a previous prime number. Also, note that you can stop scanning for prime numbers when you reach $\sqrt{n}$ (if a non-prime number is strictly greater than $\sqrt{n}$ and lower than or equal to n then it has at least one divisor (> 1) lower than or equal to $\sqrt{n}$, so it has already been crossed).

13 Memory consumption and number of arithmetic operations

Pascal's triangle is a method to compute the coefficients in the expansion of $(a + b)^n$. For example, for $n = 4$, the expansion is:

$$(a + b)^4 = a^4 + 4a^3b + 6a^2b^2 + 4ab^3 + b^4$$

thus the coefficients of the monomials are:

```
1 4 6 4 1
```

One can compute the coefficients for the i-th power from a recurrence relation. Let c_{ij} represent the j-th coefficient in the expansion of the i-th power, we have then:

$$c_{ij} = c_{i-1,j-1} + c_{i-1,j}$$

Cf. figure (1) and table (1). In this exercise, we want to write a program

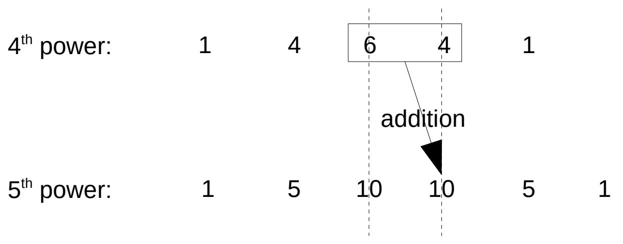

Figure 1: Computing a row in Pascal'triangle, from the previous row.

n	coefficients du développement de $(a+b)^n$				
0	1				
1	1	1			
2	1	2	1		
3	1	3	3	1	
4	1	4	6	4	1

Table 1: Pascal's triangle, down to $n = 4$.

computing Pascal's triangle using as little memory and doing as few arithmetic operations as possible. We will proceed in three steps, diminishing memory and operations at each step.

1. Write a program which asks the user for a value $n \geq 2$ and computes Pascal's triangle down to the n-th power. The program must print only the last row of the triangle, corresponding to the n-th power. In this first version of the program, use two vectors with sizes n and $n+1$, to store two successive rows of the triangle. You will only define part of these vectors when you compute the first rows of the triangle. The program should only perform $i-1$ additions to compute the coefficients of the i-th power (not n additions: do not add zeroes).

2. Write a second version of the program, using a single vector with size $n+1$. Going from one row of the triangle to the next, modify this vector, choosing the right order for computing the coefficients of the row.

3. Write a third version of the program where you half the number of operations and the size of the vector. Each row of the triangle is symmetric with respect to its central element or central two elements. So you only

need to compute half the coefficients. Try to avoid as much as possible tests on the parity of the power i for the current row. Also, we know that:

$$c_{ii} = c_{i,2} = i$$

so use this to spare a computation. When you print the final row, print it whole, including the second half, symmetric of the first half.

4. Write a fourth version of the program where you check that you do not overflow the maximum integer value when you compute coefficients. If n is too big, the program should stop, printing an error message. Find the maximum power n for which you can compute the coefficients.