1. Matrix multiplication

1.1 [5 points] Write a program `Main.f90` to read `fortran_demo1/M.dat` as the matrix `M`, and `fortran_demo1/N.dat` as the matrix `N`.

（用服务器跑代码报错：ld returned 1 exit status，本地 VScode 运行代码和结果如下）

D: > Main.f90

```f90
program Main
    implicit none
    integer , parameter :: na = 3 , ma = 5 , nb = 5 , mb = 3
    Real(8)  :: M(3,5), N(5,3) , P (3 ,3)

open(unit=20, file='~/fortran_demo1/M.dat', status='old')

    read(20,*) M

close(20)

open(unit=20, file='~/fortran_demo1/N.dat', status='old')

    read(20,*) N

close(20)
```

M.dat

```
19.48 15.79 19.28
19.28 12.92 15.86
15.86 11.29 14.04
11.93 18.60 18.23
19.28 12.92 15.86

```

N.dat

```
7.72 4.11 1.44 4.80 5.55
5.55 4.80 4.04 0.59 8.58
0.59 8.58 2.26 7.72 4.11

```

1.2 [5 points] Write a subroutine `Matrix_multip.f90` to do matrix multiplication.

D: > Matrix_multip.f90 > ...

```f90
subroutine Matrix_multip (C ,D , P )
    implicit none
    integer  i , j , k
    integer , parameter :: nc = 3 , mc = 5 , md = 3
    real , intent ( in ) :: C ( nc , mc ) , D ( mc , md )
    real , intent ( out ) :: P ( nc , md )
    real :: sum

    do i =1 , nc
    do j =1 , md
    sum = 0.
    do k =1 , mc
    sum = sum + C (i , k )* D (k , j )
    enddo
    P (i , j ) = sum
    enddo
    enddo

end subroutine Matrix_multip
```

1.3 [5 points] Call the subroutine `Matrix_multip()` from `Main.f90` to compute `M*N`; write the output to a new file `MN.dat`, values are in formats of `f9.2`.

D: > Main.f90

```
1   program Main
2       implicit none
3       integer , parameter :: na = 3 , ma = 5 , nb = 5 , mb = 3
4       Real(8)  :: M(3,5), N(5,3) , P (3 ,3)
5
6   open(unit=20, file='~/fortran_demo1/M.dat', status='old')
7
8       read(20,*) M
9
10  close(20)
11
12  open(unit=20, file='~/fortran_demo1/N.dat', status='old')
13
14      read(20,*) N
15
16  close(20)
17
18  call Matrix_multip( M , na , ma ,N , mb , P )
19
20  open(unit=20, file='~/fortran_demo1/MN.dat', status='replace')
21
22      write(20,'(3/(3f9.2))') P
23      ! 3/ : 3 rows;  3f9.2 : f9.2 for 3 times.
24  close(20)
25
26  end program Main
```

Output:

MN.dat

MN.dat

```
1
2
3
4    416.73    352.24    409.77
5    437.19    317.09    386.69
6    384.10    342.38    385.10
7
```

2.1 [5 points] Write a module `Declination_angle` that calculates the *declination angle* on a given date.

[**Hint:** using the "Better formula" from Solar Declination Angle & How to Calculate it]

```
module Declination_angle

    implicit none

    real(8), parameter      :: pi=3.1415926536

contains

    subroutine CalDeclinationAngle(day,sigma)

    integer, intent(in)     :: day
    real(8), intent(out)    :: sigma

!SIND=SIN(angle*pi/180)  COSD=COS(angle*pi/180)

        sigma = ASIND(SIND(-23.44)*COSD(360*(day+10)/365.24+360/pi*0.0167*SIN(360*(day-2)/365.24)))

    end subroutine CalDeclinationAngle

end module Declination_angle
```

2.2 [10 points] Write a module `Solar_hour_angle` that calculates the *solar hour angle* in a given location for a given date and time.

[**Hint:** using the formulas from Solar Hour Angle & How to Calculate it]

```
module Solar_hour_angle

    implicit none

    real, parameter     :: pi = 3.1415926536

contains

    subroutine CalSolarHourAngle(day,time,gma,longitude,timezone,h)

    integer, intent(in)     :: day
    real(8), intent(in)     :: time, longitude, timezone
    real(8)                 :: gma, EOT, OFFSET, LST
    real(8), intent(out)    :: h

        gma = 2*pi/365*(day-1+(time-12)/24)
        EOT = 229.18*(0.000075+0.001868*COS(gma)-0.032077*SIN(gma)-0.014615*COS(2*gma)-0.040849*SIN(2*gma))
        OFFSET = EOT+4*(longitude-15*timezone)
        LST = time+OFFSET/60
        h = 15*(LST-12)

    end subroutine CalSolarHourAngle

end module Solar_hour_angle
```