

Name (please print): _____

Important test instructions – code fragments

Throughout this exam, you will be asked to write a code fragment based on certain assumptions (eg., assume there is a 2-D INTEGER array IA). Any time you are asked to write a code fragment, you should not attempt to write a full working program, but just a section of code that does the required operation. You will often need additional variable declarations (eg., you need a variable for use as a loop counter); in these cases show all declarations of new variables at the beginning of your code fragment (you should assume that **IMPLICIT NONE** is in effect). Only a few of the simpler questions at the beginning will ask for full working programs. Unless it is specified, you should assume that any arrays have been initialized in some legal fashion before reaching your code fragment. Questions that require a full program will ask you to write a **program**, while those that don't will ask you to write a **code fragment**.

1. Write a **program** that will print out every fourth number between 8 and 800 (inclusive) using a counter loop (6 pts)

```
PROGRAM test
  IMPLICIT NONE
  INTEGER :: i

  DO i = 8, 800, 4
    PRINT *, i
  END DO

END PROGRAM test
```

2. Write a **program** that will print out every tenth number between 100 and 10,000 (inclusive) using a logical loop (if you don't remember which loop is which, be sure to write this program using a different style of loop than the one you used in the last problem). (6 pts)

```
PROGRAM test
  IMPLICIT NONE
  INTEGER :: i

  i = 100
  DO
    IF (i .GT. 10000) EXIT
    PRINT *, i
    i = i + 10
  END DO

END PROGRAM test
```

3. Write a **program** that will prompt the user for an integer N . Your program will then print out the square of the first N numbers starting from 1. For instance, if the user entered 5, your output should be:

```
1 squared is      1
2 squared is      4
3 squared is      9
4 squared is     16
5 squared is     25
```

(remember that squaring a number consists of multiplying that number by itself, which is equivalent to the number to the power of 2). You may assume that the user will always enter a strictly positive number (zero or negative numbers will not be entered). (10 pts)

```
PROGRAM test
  IMPLICIT NONE
  INTEGER :: i, N

  PRINT *, 'Enter the last number to square:'
  READ *, N
  DO i = 1, N
    WRITE (*,1000) i, i*i
  END DO

1000 format(i4, ' squared is', i14)
```

4. Write a **program** that reads in a variable-length array from a file using loops (i.e. you cannot use array subsections). The first line of the file provides the length of the array, and each additional line of the file provides exactly one element of the array. The array is known to be of type REAL, but has no known maximum size. The input filename is `array.in`. (12 pts)

```
PROGRAM readarray
  IMPLICIT NONE
  REAL, ALLOCATABLE :: A(:)
  INTEGER :: N, i

  OPEN(unit=10, file='array.in', status='old', action='read')
  READ(10,*), N
  ALLOCATE(A(N))
  do i = 1, N
    READ(10,*) A(i)
  end do
  CLOSE(10)
  PRINT *, A ! just to test
  DEALLOCATE(A)

END PROGRAM readarray
```

5. Write a **code fragment** that uses a loop to sum all the elements of the following array:

```
REAL :: A(2000)
```

You must also print the sum once it is computed. (6 pts)

```
REAL :: mysum
INTEGER :: i

mysum = 0.0
DO i = 1, 2000
    mysum = mysum + A(i)
END DO
PRINT *, 'SUM of A = ', mysum
```

6. Write a **code fragment** that computes and prints out the number of strictly negative elements in the array declared as:

```
INTEGER :: IA(800)
```

(10 pts)

```
INTEGER :: i, nneg

nneg = 0
DO i = 1, 800
    IF (IA(i) .LT. 0) &
        nneg = nneg + 1
END DO
PRINT *, 'Number of negative values in IA is:', nneg
```

7. Write a **code fragment** that copies an M-row by N-column piece of the double precision 2-D array A into the 2-D array B *using loops* (i.e., you cannot use array subsections). You may assume the following declarations:

```
DOUBLE PRECISION :: A(1000,800), B(600,2000)
INTEGER, PARAMETER :: M=400, N=600
```

(10 pts)

```
INTEGER :: i, j

DO j = 1, N
  DO i = 1, M
    B(i,j) = A(i,j)
  END DO
END DO
```

8. Do the same copy, but this time use array subsections in your **code fragment** to perform the copy (i.e., you cannot use loops at all for this question). (4 pts)

```
B(1:M,1:N) = A(1:M,1:N)
```

9. Assume the following array declarations:

```
REAL :: A(1000), POSA(1000), NEGA(1000)
```

Assume **A** has been initialized to random values. Further, assume there are **npos** positive numbers in **A**, and **nneg** negative numbers in **A**. Write a **code fragment** that computes **nneg**, **npos**, and places all the positive numbers in **A** in the first **npos** entries of **POSA**, and places all negative numbers in **A** in the first **nneg** entries of **NEGA**. For this problem, zero will be counted as a positive number. (10 pts)

```
INTEGER :: i, npos, nneg

npos = 0
nneg = 0
DO i = 1, 1000
  IF (A(i) .LT. 0.0) THEN
    nneg = nneg + 1
    NEGA(nneg) = A(i)
  ELSE
    npos = npos + 1
    POSA(npos) = A(i)
  ENDIF
END DO
```

10. Write a **code fragment** including a loop (but only one loop!) that indexes a REAL 1-D array of unknown size (your code should discover the size), and at the end prints out three values computed in this loop:

- (A) The sum of every number less than -1.0 in the array
- (B) The product of every strictly positive number (> 0) in the array
- (C) The number of array elements that are exactly zero

You should assume the array index begins at 1, and that the array name is X. (16 pts).

```
INTEGER :: N, nzeros
REAL :: mysum, myprod

N = size(X)
mysum = 0.0
myprod = 1.0
nzeros = 0
DO i = 1, N
    IF (X(i) .LT. -1.0) THEN
        mysum = mysum + X(i)
    ELSE IF (X(i) .GT. 0.0) THEN
        myprod = myprod * X(i)
    ELSE IF (X(i) .EQ. 0.0) THEN
        nzeros = nzeros + 1
    END IF
END DO
PRINT *, 'Number of zeros in X is:', nzeros
PRINT *, 'Sum of all numbers less than -1 in X is:', mysum
PRINT *, 'Product of all strictly positive numbers in X is:', myprod
```

11. Write a **code fragment** that sorts the array A into greatest-to-least order, assuming the following declaration:

```
REAL :: A(100)
```

(10 pts)

```
REAL :: bigval
INTEGER :: imax, i, j
```

```
DO i = 1, 99
  bigval = A(i)
  imax = i
  DO j = i+1, 100
    IF (A(j) .GT. bigval) THEN
      bigval = A(j)
      imax = j
    END IF
  END DO
  IF (i .NE. imax) THEN
    A(imax) = A(i)
    A(i) = bigval
  END IF
END DO
```

```
.....
.....
..... OR .....
.....
.....
```

```
DO i = 1, 99
  bigval = A(i)
  DO j = i+1, 100
    IF (A(j) .GT. bigval) THEN
      bigval = A(j)
      A(j) = A(i)
      A(i) = bigval
    END IF
  END DO
END DO
```