1. Теория.

Вывод-ввод в файл осуществляется по алгоритму:

open(1, file='1.dat') ! открывает файл 1.dat в папке с программой и связывает его с устройством ввода-вывода 1

read(1,\*) z ! считывает из файла элементы в массив z, данные в файле в произвольном формате

write(1,\*) а ! записывает в файл массив а, с форматированием по умолчанию

write(1,'(20i3)') a ! записывает в файл целый массив а, по 20 элементов в строку с 3 символами на элемент

write(1,'(33ES16.8)') а ! записывает в файл вещественный массив, по 33 элемента в строку с 16 символами на элемент и с 8 знаками после запятой

1.1. Сечения массивов.

Сечение массива позволяет применить процедуру или присваивание не ко всем элементам массива, а только к выбранным по определенному правилу. Существует два способа задачи сечения: с помощью индексного триплета и векторного индекса. Первый способ хорош, если, к примеру, надо применить процедуру к каждому третьему элементу массива. Второй если присваивание или процедура применимы к некому неупорядоченному, но заранее известному, набору элементов массива.

real :: a(9)

a(1:9:3)=3

Каждый третий элемент массива становится равным 3. Триплет имеет вид: (нижняя граница:верхняя граница:шаг). Так, если надо присвоить все элементы с 4 по 6 то вид будет следующий:

a(4:6:1)=3

Ранее не было сказано, но в Fortran можно задавать диапазоны индексов массивов. К примеру можно задать двумерный массив у которого по одному измерению будут индексы от -5 до 5, включая 0, а по другому от 8 до 10:

real :: a(-5:5,8:10)

Если массив задан одним числом, то по умолчанию наименьшим индексом массива является 1. К такому массиву также применимы сечения:

a(-5:5:5,9)=3

присвоение выполнено только для 3 элементов a(-5,9), a(0,9), a(5,9).

Если надо присвоить значения всем элементам строки по сечению то можно записать:

a(:,9)=3 или a(::1,9)

Последняя запись означает, что если не задано значение границы в триплете, оно берется минимальным для нижней, максимальным для верхней и 1 для шага (при этом для границ не пишется значение, а для шага не пишется также и соответствующее дветочие). Шаг может быть и отрицательным, что дает возможность легко поменять порядок элементов массива:

real :: a(-5:5)

a=a(5:-5:-1)

Важно отметить, что при использовании сечения массива используется массив-маска. Это массив того же размера что и преобразуемый сечением, но составленный из логических переменных. True если элемент входит в сечение и False если не входит. Таким образом, при применении сечений есть отличия от применения циклов. Во-первых, при прочих равных, операции с сечениями выполняются значительно быстрее, так как не происходит перебора элементов, а происходит перемножение двух массивов, один из которых логический массив-маска, не требующий большого размера выделенной памяти. Вторым отличием является одновременность присвоения при использовании сечения. К примеру:

real :: a(10)

integer :: i

a=0

do i=2,10,1

a(i)=a(i-1)+1

end do

Даст последовательность от 0 до 9. Тогда как:

real :: a(10)

a=0

a(2:10:1)=a(1:9:1)+1

даст 0 и 9 единиц. Такую особенность надо учесть при операциях с сечениями, которые используют значения самого массива. В некоторых задачах сечение становится неприменимым, зато в некоторых других позволяет избегать создания промежуточных временных массивов.

Для второго способа использования сечений массива используются векторные индексы – одномерные целочисленные массивы:

real a(20)

integer :: v(3)

v=(/ 1, 5, 7 /)

a(v)=2

Даст результатом присвоение 3 элементам (1, 5 и 7) массива а значения 2.

1.2. Операции с массивами.

Далее перечислены функции и процедуры для работы с массивами.

all(a [, dim]) – возвращает .true. если все элементы логического массива а (или его измерения dim, если задано) истинны, иначе возвращает .false.

any(a [, dim]) – возвращает .true. если хотя бы один элемент логического массива а (или его измерения dim, если задано) истинен, иначе возвращает .false.

count(a [, dim]) – возвращает число элементов логического массива а (или его измерения dim, если задано), имеющих значение .true. Результат представлен в integer(4).

size(a [, dim]) – возвращает размер массива. То есть количество его элементов.

shape(a [, dim]) – возвращает форму массива. Число элементов по каждому измерению.

lbound(a [, dim]) – возвращает левую (нижнюю) границу массива.

ubound(a [, dim]) – возвращает правую (верхнюю) границу массива.

maxloc(a, [, dim] [, mask]) – возвращает индексы максимального элемента массива или максимальных элементов по всему массиву, по измерению dim, если оно задано, или если удовлетворяет условию mask (также необязательный параметр). Если несколько элементов содержат максимальное значение, берется первый из них. Параметр mask имеет вид логического массива, форма которого совпадает с формой массива а (или если задано dim, то форму размерности). Данный массив может быть результатом вычисления логического выражения, к примеру maxloc(a, 2, a<4). В итоге все элементы, значение которых больше 4 не будут учитываться. Существует аналогичная функция minloc. Аналогично вводятся функции maxval и minval, которые возвращают значения максимального и минимального элемента, соответственно.

product(a, [, dim] [, mask]) – возвращает произведение всех элементов массива а. Если заданы дополнительные параметры, то с учетом измерения и маски.

sum(a, [, dim] [, mask]) – возвращает сумму всех элементов массива а. Если заданы дополнительные параметры, то с учетом измерения и маски.

Для векторов, существует отдельная функция умножения по правилам линейной алгебры:

dot\_product(v\_1, v\_2) – возвращает скалярное произведение двух векторов (задаются как одномерные массивы).

matmul(a1,a2) – возвращает по принятым в линейной алгебре правилам умножение матриц целого, вещественного, комплексного или логического типа. Матрицы представляют собой массивы. а1 или а2 могут быть вектором.

Возможно 3 случая:

а1 имеет форму (n,m), а а2 – (m,k), тогда результат имеет форму (n,k). Допустимо n=k.

а1 имеет форму (m), а а2 – (m,k), тогда результат имеет форму (k).

а1 имеет форму (n,m), а а2 – (m), тогда результат имеет форму (m).

cshift(a,shift [, dim]) – выполняет циклический сдвиг массива а на shift по необязательному индексу размерности dim. shift обязательно целое, но может быть вектором. Тогда каждое измерение сдвигается на величину элемента вектора. Так:

массив

1 2 3

4 5 6

7 8 9

при cshift(a,1,dim=1)

превратиться в

4 5 6

7 8 9

1 2 3

а при cshift(a,shift=/-1,1,0/,dim=2) в

3 1 2

5 6 7

7 8 9

eoshift(a, shift [, boundary] [, dim]) – выполняет левый или правый (в зависимости от знака shift) сдвиг по заданному направлению dim. boundary – параметр того же типа что и а, которое задает какими элементами будет замещаться пропуски, возникшие в результате сдвига (по умолчанию 0). Может быть массивом, тогда элементы берутся по очереди.

transpose(a) – производит транспонирование матрицы (массива) а.

1.3. Массивы с динамической памятью.

В случае если необходимо на определенном этапе работать с большим массивом, который далее не нужен, полезно использовать массивы с динамической памятью.

real, allocatable :: a(:,:) ! задает двумерный массив, в настоящий момент пустой и не занимающий память

allocate(a(200,500)) ! задает размер массива и выделяет под него память

deallocate(a) ! высвобождает память после окончания работы с массивом.

Операторы allocate и deallocate могут быть использованы в любом месте программы. Начальное описание массива с его размерностью должно быть описано в блоке определения переменных. Впрочем, можно избежать такой сложности, хоть и с усложнением структуры программы и потерей памяти:

real, allocatable :: a(:,:,:,:,:,:,:) ! массив максимальной размерности – 7

allocate(a(200,500,1000,5000,1,1,1)) ! размерности, которые оказались не нужны имеют размер 1.

deallocate(a) ! не стоит забывать высвобождать память, особенно если массив содержит 5Е+11 элементов.

Другой момент, когда полезен такой массив – если один массив необходимо использовать несколько раз с разными размерами.

integer, parameter :: i=2, j=5, n=500, m=25000

real allocatable :: a(:,:)

allocate(a(i,j))

…

deallocate(a)

allocate(a(n,m))

…

deallocate(a)

Данный пример показывает, как один массив можно использовать несколько раз с разными размерами.

2. Задание.

1.1. Сечения массивов.

Продемонстрировать применение сечений массивов вместо циклов. Продемонстрировать принцип работы с массивом-маской. Показать запись в файл через применение сечения массива. Продемонстрировать задачу с применением сечения массива заданного векторным индексом.

1.2. Операции с массивами.

Продемонстрируйте встроенные процедуры работы с массивами, матрицами, векторами.

1.3. Массивы с динамической памятью.

Продемонстрировать работу массивов с динамической памятью. Показать возможность многократного использования массива с динамической памятью при изменении размера массива.