

Л. Ф 05. МАССИВЫ

1. ОСНОВЫ ИСПОЛЬЗОВАНИЯ МАССИВОВ [1]

1.1. Понятие массива

Массивом называется совокупность переменных одного типа, обозначаемая именем с индексами в круглых скобках. Например, вектор с компонентами u_1, u_2, \dots, u_n , можно представить одномерным массивом

$$U(I), 1 \leq I \leq N,$$

а матрицу с компонентами A_{ij} можно представить двумерным массивом

$$A(I,J), 1 \leq I \leq N, 1 \leq J \leq M;$$

Количество индексов в массиве определяет его размерность, и массивы могут быть, соответственно, одномерными, двумерными, трехмерными и т.д.

1.2. Оператор описания массива DIMENSION

Каждый массив должен быть описан в начале программы с помощью *оператора размерности* DIMENSION с указанием предельных значений каждого индекса, которые задаются целыми константами. Это необходимо для того, чтобы зарезервировать соответствующий объем памяти для хранения элементов массива.

Примеры:

```
DIMENSION A(10,30),BK(500),IU(40)
```

Здесь первый массив — двумерный, а второй и третий — одномерные. Первый индекс в массиве A изменяется от 1 до 10, второй от 1 до 30. Во втором и третьем массивах индексы из меняются соответственно от 1 до 500 и от 1 до 40.

1.3. Совмещение описания массива с описанием типа переменных

Описание размерности массива может быть совмещено с описанием типа переменных. Поэтому запись

```
REAL KLM, XF
```

```
INTEGER FR, SS
```

```
DIMENSION KLM(15,15),FR(100),XF(10),SS(5,5)
```

эквивалентна записи

```
INTEGER FR(100), SS(5,5)
```

```
REAL KLM(15,15), XF(10)
```

Описание массива в программе может быть произведено только один раз.

1.4. Размещение элементов массива в памяти ЭВМ

Массив размещается в последовательно расположенных ячейках памяти. Если массив одномерный, то его элементы хранятся в памяти друг за другом, например,

$A(1), A(2), A(3), A(4), \dots$

Во многих языках программирования, например в СИ, элементы двумерного массива располагаются в памяти ЭВМ по строкам, в Фортране — по столбцам. Если массив двумерный, например DIMENSION B(2,4)

		j			
		1	2	3	4
i	1	B(1,1)	B(1,2)	B(1,3)	B(1,4)
	2	B(2,1)	B(2,2)	B(2,3)	B(2,4)

то, его элементы хранятся в памяти в таком порядке

$B(1,1), B(2,1), B(1,2), B(2,2), B(1,3), B(2,3), B(1,4), B(2,4)$.

Тот же принцип сохраняется и для массивов большей размерности: быстрее всего изменяется первый индекс, затем второй и т.д. Это следует учитывать при вводе/выводе и инициализации многомерного массива.

1.5. Оператор PARAMETER

При описании массивов можно предварительно присвоить константам символические имена. Для этого используется оператор PARAMETER вида

PARAMETER (<имя>=<константа>, <имя>=<константа>, ...)

Здесь <имя> — символическое имя, образованное по правилам Фортрана. Тогда, например, запись

DIMENSION A1(100,200), A2(100,200), B(200,100)

эквивалентна записи

PARAMETER (N1=100, N2=200)

DIMENSION A1(N1,N2)A2(N1,N2),B(N2,N1)

Такая запись позволяет упростить изменение размерности при модификации программы.

1.6. Обращение к элементу массива

При обращении к элементу массива необходимо указывать его имя и индексы, которые могут быть либо константами, либо арифметическими выражениями целого типа, например

$$V=A(I,J)+B(3.2*L+7)$$

В этом случае целые переменные I,J,L должны быть заданы в предыдущей части программы.

1.7. Ввод и вывод массива

При вводе и выводе массивов используются два основных способа:

- ввод-вывод массива по имени;
- с помощью цикла.

Первый способ (пример)

DIMENSION A(10), B(3,4)

```
READ(*,*) A,B
```

В этом случае должны быть последовательно введены 10 чисел которые соответствуют элементам массива А, а затем введены 12 чисел, которые соответствуют элементам массива В, согласно их последовательности расположения в памяти.

Второй способ (пример)

```
DIMENSION A(10), B(3,4)
```

```
READ(*,*) N,L,M,(A(I),I=1,N), ((B(I,J),J=1,L),I=1,M)
```

В этом случае числа N,L,M подчиняются ограничениям $N \leq 10$, $L \leq 4$, $M \leq 3$, поскольку в противном случае мы выйдем за границы расположения элементов массивов. Этот способ удобнее, в программе используется меньшее количество элементов массива, нежели то, которое задано в операторах размерности. В этом примере сначала вводятся три целых числа N,L,M, и затем последовательно N элементов массива А, а затем L*M элементе массива В в последовательности

```
B(1,1),B(1,2),...B(2,1),B(2,2),...
```

В этом случае последовательность ввода элементов В не совпадает с порядком их хранения в памяти. Если последний оператор записать как

```
READ(*,*)N,L,M,(A(I),I=1,N),((B(I,J),I=1,M),J=1,L)
```

то порядок ввода элементов массива В и порядок их размещения в памяти будет одинаковый.

Вывод массива отличается от их ввода только заменой оператора READ на оператор WRITE.

1.8. Использование массивов

Рассмотрим следующую задачу.

Ввести с клавиатуры $N \leq 100$ чисел и найти их сумму/произведение. Соответствующая программа может иметь вид

Сумма	Произведение
DIMENSION A(100)	DIMENSION A(100)
READ (*,*)N,(A(I),I=1,N)	READ (*,*)N,(A(I),I=1,N)
S=0.0	P=1.0
DO 5 I=1,N	DO 5 I=1,N
S=S+A(I)	P=P*A(I)
5 CONTINUE	5 CONTINUE
WRITE (*,*)S	WRITE (*,*)P
STOP	STOP
END	END

2. ДОПОЛНИТЕЛЬНЫЕ СВЕДЕНИЯ О МАССИВАХ [2]

2.1. Размещаемые массивы

Составим программу, вычисляющую среднюю оценку студенческой груп-

пы за экзамен по информатике. Для вычисления искомого значения нам потребуется ввести в ЭВМ число студентов в группе и оценки каждого студента по этому предмету. Число студентов будем хранить в простой переменной *ns*.

В отличие от простой переменной массив обладает атрибутом DIMENSION. Так, массив оценок можно объявить:

```
integer marks(40)      ! Массив оценок
```

Как правило, число студентов в группе не превышает 30. Однако на всякий случай мы объявили массив из 40 элементов, предполагая, что даже в самой большой группе не может быть более 40 студентов.

Рассмотренный массив *marks* является статическим — его размер не может быть изменен в процессе вычислений, поэтому мы вынуждены задать его размер с некоторым запасом (чтобы иметь возможность использовать массив для любой студенческой группы). Это приводит к тому, что программа, как правило, занимает больше памяти, чем это требуется на самом деле. Подобного перерасхода памяти можно избежать, если применить *динамический массив*. Динамический массив относится к типу *размещаемых*. Под него каждый раз выделяется столько памяти, сколько нужно.

Работа с таким массивом происходит следующим образом:

- выполняется объявление размещаемого массива. В отличие от статических размещаемых массивы объявляются с атрибутом ALLOCATABLE. Кроме того, каждое измерение размещаемого массива задается в виде двоеточия, например: `integer, allocatable, dimension(:) :: marks`
- определяется размер массива;
- оператором ALLOCATE выделяется память под массив;
- после выполнения вычислений выделенная память освобождается. Это выполняется оператором DEALLOCATE;
- после этого массиву вновь может быть выделена новая область памяти.

Пример.

```
integer ns              ! Число студентов в группе
integer ierr           ! Статусная переменная
integer, allocatable, dimension(:) :: marks
...
print *, 'Введите число студентов в группе'
read *, ns
allocate(marks(ns), stat = ierr)
if(ierr /= 0) stop 'Не удастся выделить память под массив marks'
...                    ! Выполняются вычисления
deallocate(marks, stat = ierr)
if(ierr /= 0) stop 'Не удастся освободить память'
```

При размещении массива параметр STAT= позволяет узнать, удалось ли разместить массив. Этот параметр может быть опущен, но тогда любая неудача при выделении памяти приведет к ошибке этапа исполнения и остановке программы. Параметр указывается в операторе ALLOCATE последним. При удачном выделении памяти целочисленная статусная переменная *ierr* возвращает

нуль, в противном случае возвращается код ошибки размещения. Причиной ошибки может быть, например, недостаток памяти или попытка разместить ранее размещенный и не освобожденный оператором DEALLOCATE объект. Аналогичную роль играет необязательный параметр STAT= и в операторе DEALLOCATE.

В операторе DEALLOCATE можно использовать только ранее размещенные объекты. Попытка освобождения неразмещенного объекта вызовет ошибку выполнения.

При использовании размещаемого массива в качестве локальной переменной процедуры следует в Фортране 90 перед выходом из процедуры выполнить освобождение отведенной для него памяти.

Замечания:

1. В Фортране 95 освобождение выделенной под локальный размещаемый массив памяти происходит автоматически при выходе из процедуры.

2. Хорошей практикой является освобождение оператором DEALLOCATE, выделенной динамической памяти, когда надобность в ней отпадает. Это позволяет избежать накопления неиспользуемой и недоступной памяти.

Чтобы узнать, выделена ли под размещаемый массив память, используется встроенная функция ALLOCATED, возвращающая .TRUE., если массив размещен, и .FALSE. — в противном случае

2.2. Характеристики массивов

Массив характеризуется числом измерений, которых может быть не более семи. Число измерений массива называется его *рангом*. Массив ранга 1 называют *вектором*, а ранга 2 - *матрицей*. Объект ранга 0 является скаляром.

Число элементов массива называется его *размером*. Также массив характеризуется *формой*, которая определяется его рангом и протяженностью (*экстендом*) массива вдоль каждого измерения. Например, оператор

```
real, dimension (2, 3, 10) :: b
```

объявляет массив *b* ранга 3, размера $2*3*10 = 60$ и формы (2, 3, 10). Протяженность массива по первому измерению равна двум, по второму - трем, а по третьему - десяти.

Каждая размерность массива может быть задана *нижней* и *верхней* границами, которые разделяются двоеточием. Например:

```
real, dimension(4:5, -1:1, 0:9) :: c    ! Форма массива (2, 3, 10)
```

Ранг, форма и размер массивов *b* и *c* совпадают. Такие массивы называются *согласованными*.

Нижняя граница и последующее двоеточие при объявлении массива могут быть опущены, тогда по умолчанию нижняя граница принимается равной единице.

2.3. Способы объявления массивов

Помимо использованного способа массив можно объявить и без применения атрибута DIMENSION, например все последующие объявления дают двумерный целочисленный массив:

```
integer, parameter :: m = 2, n = 4
integer a(m, n)           ! или a(1:m, 1:n)
или
integer b
dimension b(m, n)        ! Оператор DIMENSION
или
integer, dimension(m, n) :: c    ! Атрибут DIMENSION
```

Существуют разные способы объявления и размещаемых массивов:

```
integer, allocatable :: a(:, :)
или
integer, dimension(:, :) :: b
allocatable b           ! Оператор ALLOCATABLE
или
integer c
dimension c(:, :)      ! Оператор DIMENSION
alocatable c
или
integer, alocatable, dimension (:,):: d
```

2.4. Конструктор массива

В Фортране можно, используя *конструктор массива*, задать одномерный массив. Конструктор массива имеет вид:

(/ список-значений /)

Пробелы между круглой скобкой и слешем *не допускаются*. Значения в списке разделяются запятыми и должны иметь одинаковый тип и разновидность типа.

Пример массива - буквальной константы: (/1,2, 3, 4, 5 /). Такой массив может быть использован в выражении, например:

```
integer a(5)
a = 2 * (/ 1, 2, 3, 4, 5 /)
write(*, *) a ! 2 4 6 8 10
```

Список-значений может содержать последовательность скаляров, встроенных DO-циклов и массивов любого ранга. Каждое значение списка может быть результатом выражения.

Встроенный DO-цикл имеет вид:

(выражение, dovar = start, stop [, inc])

dovar - целочисленная скалярная переменная (параметр цикла). *start*, *stop*, *inc* - целочисленные константные выражения, определяющие диапазон и шаг изменения *dovar*. Если *inc* отсутствует, то шаг устанавливается равным единице.

Встроенный DO-цикл добавляет в список значений

MAX(INT(stop - start + inc)/inc), 0)

элементов. Выражение может содержать *dovar*. Возможна организация вло-; 1 женных встроенных DO-циклов.

Если в списке появляется многомерный массив, то его значения берутся в порядке их размещения в памяти ЭВМ.

2.5. Сечение массива

В Фортране можно получить доступ не только к отдельному элементу массива, но и к некоторому подмножеству его элементов. Такое подмножество элементов массива называется *сечением массива*. Сечение массива может быть получено в результате применения *индексного триплета* или *векторного индекса*, которые при задании сечения подставляются вместо одного из индексов массива.

Индексный триплет имеет вид:

[нижняя граница] : [верхняя граница] [:шаг]

Каждый из параметров триплета является целочисленным выражением. *Шаг* изменения индексов может быть и положительным и отрицательным, но не может равняться нулю. Все параметры триплета являются необязательными.

Индексный триплет задает последовательность индексов, в которой первый элемент равен его нижней границе, а каждый последующий больше (меньше) предыдущего на величину шага. В последовательности находятся все задаваемые таким правилом значения индекса, лежащие между границами триплета. Если же нижняя граница больше верхней и шаг положителен или нижняя граница меньше верхней и шаг отрицателен, то последовательность является пустой,

Пример.

real a(1:10)

a(3:7:2) = 3.0

! Триплет задает сечение массива с элементами
! a(3), a(5), a(7), которые получают значение 3.0

a(7:3:-2) = 3.0

! Элементы a(7), a(5), a(3) получают значение 3.0

При отсутствии нижней (верхней) границы триплета ее значение принимается равным значению нижней (верхней) границы соответствуют экстенда массива. Так, a(1:5:2) и a(:5:2) задают одно и то же сечение массива *a*, состоящее из элементов a(1), a(3), a(5). Сечение из элементов a(2), a(5), a(8) может быть задано так: a(2:10:3) или a(2::3).

Пример.

real :: a(10), r = 4.5

a(2::3) = 4.0

!Элементы a(2), a(5), a(8) получают значение 4.0

a(7:9) = 5.0

!Элементы a(7), a(8), a(9) получают значение 5.0

a(:) = 3.0

!Все элементы массива получают значение 3.0

a(::3) = 3.0

!Сечение из элементов a(1), a(4), a(7), a(10)

a(::int(r/2)) = 3.0

! Параметр триплета — целочисленное выражение

ние

print *,size(a(4:3))

! 0 (сечение нулевого размера)

! функция SIZE возвращает размер массива

Нижняя граница триплета не может быть меньше нижней границы соответствующего экстенда массива. Так, ошибочно задание сечения $a(-2:5:3)$ в массиве $a(1:10)$. Верхняя граница триплета должна быть такой, чтобы последний элемент задаваемой триплетом последовательности не превышал верхней границы соответствующего экстенда массива. Например, в массиве $a(1:10)$ допустимо сечение $a(3:12:5)$.

В случае многомерного массива сечение может быть задано посредством подстановки индексного триплета (впрочем, так же, как и векторного индекса) вместо одного или нескольких индексов, например:

```
real a(8, 3, 5)
a(1:4:2, 2:3, 4) = 4.0
```

В заданном сечении в первом измерении индекс может принимать значения 1 и 3, во втором - 2 и 3, а в третьем только 4. Таким образом, сечение обеспечивает доступ к элементам $a(1, 2, 4)$, $a(1, 3, 4)$, $a(2, 2, 4)$ и $a(2, 3, 4)$. Поскольку в третьем измерении сечения индекс фиксирован, то сечение является двумерным массивом с формой (2, 2).

Частными случаями сечений двумерного массива являются его строки и столбцы, например:

```
integer a(5, 8)
a(3, :) = 3           ! Сечение - 3-я строка матрицы
a(:, 4) = 7          ! Сечение - 4-й столбец матрицы
```

Векторный индекс является одномерным целочисленным массивом, содержащим значения индексов, попадающих в сечение исходного массива,

Пример:

```
real a(20), b(10,10)      ! vi, vj - целочисленные массивы;
integer :: vi(3), vj(2)  ! используются как векторные индексы
vi = (/1, 5, 7/) vj = (/2, 7/) ! для задания сечений массивов a и b
a(vi) = 3.0              ! (1), a(5), a(7) получают значение 3.0
b(2, vj) = 4.0           ! b(2, 2), b(2, 7) получают значение 4.0
b(vi, vj) == 5.0        ! b(1, 2), b(1, 7), b(5, 2), b(5, 7),
                        ! b(7, 2) и b(7, 7) получают значение 5.0
```

Векторный индекс в отличие от индексного триплета позволяет извлечь в сечение произвольное подмножество элементов массива. Значения индексов должны находиться в пределах границ соответствующей размерности исходного массива. Значения индексов в векторном индексе могут располагаться в произвольном порядке и могут повторяться. Например:

```
real a(10, 8) /80 * 3.0/, b(5)
b = a(3, (/ 5, 3, 2, 7, 2 /))
```

В массив b попадут значения элементов сечения массива a : $a(3, 5)$, $a(3, 3)$, $a(3, 2)$, $a(3, 7)$ и вновь $a(3, 5)$.

Сечения с повторяющимися значениями индексов не могут появляться в правой части оператора присваивания и в списке ввода оператора. Например, присваивание

```
real a(10)
```

$a(/ 5, 3, 2, 7, 2 /) = (/ 1.2, 1.3, 1.4, 1.5, -1.6 /)$

недопустимо, поскольку 1.4 и -1.6 не могут одновременно храниться в $a(2)$.

Размер сечения равен нулю, если векторный индекс имеет нулевой размер. Сечение массива с векторным индексом не может быть внутренним файлом, адресатом ссылки. Если сечение массива с векторным индексом является фактическим параметром процедуры, то оно рассматривается как выражение и соответствующий формальный параметр должен иметь вид связи `INTENT(IN)`. При использовании заданного векторным индексом сечения в качестве фактического параметра в процедуре создается копия этого сечения, которую и адресует соответствующий формальный параметр.

Сечение массива (заданное индексным триплетом или векторным индексом) сохраняет большинство свойств массива и может быть использовано и в качестве параметра встроенных функций обработки массивов, элементных и справочных функций.

2.6. Использование сечений массива

Сечения массивов (наряду с полными массивами) используются в выражениях и в качестве параметров элементных функций. Сечения позволяют существенно улучшить технику программирования. Так, сечениями в ряде случаев заменяются `DO`-циклы

ЛИТЕРАТУРА

1. Самохин А.Б., Самохина А.С. Фортран и вычислительные методы. — М.: Русина, 1994. — 120 с.
2. Бартенев О.В. Фортран для студентов. — М.: Диалог-МИФИ, 1999. — 400 с.