ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ МАЙКЛА МЕТКАЛФА

Эпоха, когда прикладные программы создавались на языке Fortran, почти полностью совпадает с периодом существования компьютеров общего назначения. Это удивительный факт, и с учётом того, что многие другие языки программирования высокого уровня прекратили своё существование, трудно понять, почему получилось именно так. Возможно, исходные принципы проектирования Джона Бэкуса (John Backus) – простота использования и эффективность выполнения – стали двумя решающими факторами. Возможно, сыграла роль преданность языку Fortran сообщества его пользователей, которые всегда старались не отставать от новейших разработок в области технологии программирования и адаптировать язык к постоянно расширяющемуся кругу требований.

В течение нескольких десятилетий Fortran считался вымирающим языком, но, несмотря на все предсказания, оказался на удивление живучим. Более того, в последние годы возобновилась его стандартизация, и последний стандарт Fortran 2008 должен снова продлить жизнь этому языку. С учётом этих нововведений очень жаль, что продолжают существовать старые версии Fortran, как в форме давно устаревших курсов, читаемых неисправимо упрямыми преподавателями, так и в виде вышедших из употребления концепций, о которых постоянно твердят его критики. Современный Fortran – это процедурный, императивный, компилируемый язык с синтаксисом, соответствующим точному представлению математических формул. Независимые процедуры могут быть скомпилированы отдельно или сгруппированы в модули, что упрощает создание крупномасштабных программ и библиотек процедур. В язык включены функциональные возможности для обработки массивов, абстрактные типы данных, динамические структуры данных, средства объектно-ориентированного программирования и параллельной обработки. Fortran способен без затруднений взаимодействовать с C. Таким образом, современный

Fortran, начиная с версии Fortran 95 (так теперь стали обозначаться версии стандарта) – это мощный инструмент. Он в полной мере поддерживает структурное программирование, а средства объектно-ориентированного программирования, появившиеся в стандарте Fortran 2003, стали самым значительным усовершенствованием языка, его главным нововведением. Большинство из упомянутых новых функциональных возможностей описано в данной книге.

Но ни один стандарт до Fortran 2003 включительно не содержал никаких средств, специально предназначенных для поддержки параллельного программирования. Такая поддержка осуществлялась опосредованно, с привлечением вспомогательных стандартов, в частности HPF (High-Performance Fortran), MPI (Message Passing Interface), OpenMP и Posix Threads (Pthreads). Использование библиотек MPI и OpenMP стало массовым явлением, но HPF в конечном счёте не имел особого успеха. Сейчас, после принятия стандарта Fortran 2008 одним из самых сильных свойств современного языка Fortran является непосредственная поддержка параллельного программирования, благодаря введению чрезвычайно востребованного средства: комассивов (coarrays).

Директивы HPF имели форму строк комментариев и распознавались только HPF-процессором. Например, существовала возможность выравнивания трёх совпадающих по форме массивов по четвёртому с обеспечением локальности ссылок. Другие директивы позволяли распределить обработку выравниваемых массивов по нескольким процессорам. С другой стороны, MPI представляет собой универсальную библиотеку процедур для передачи сообщений, а библиотека OpenMP, поддерживающая независимое от платформы параллельное программирование с совместным использованием памяти, состоит из набора директив компилятора, библиотечных подпрограмм и переменных среды, которые определяют поведение программы во время выполнения. Posix Threads – это стандарт, определяющий спецификацию библиотеки для поддержки многопоточности.

В отличие от всех перечисленных средств, главной целью введения комассивов является предоставление синтаксиса, минимально влияющего на внешний вид программы и позволяющего распределить по нескольким процессорам не только данные, как в модели «одна инструкция, много данных» (Single Instruction Multiple Data, SIMD), но и работу в соответствии с моделью «одна программа, много данных» (Single Program Multiple Data, SPMD). От программиста требу-

ется знание лишь небольшого набора новых правил. Работа с комассивами — это самое важное новшество в стандарте Fortran 2008, но кроме него была введена новая форма управляющей конструкции `do concurrent` как способ распараллеливания циклов. Вполне очевидно, что появилась возможность обеспечения полноценного режима параллельного выполнения, не выходя за рамки языка. Всё это также рассматривается и сравнивается в данной книге.

Другие важные нововведения в стандарте Fortran 2008: подмодули (submodules), более удобный доступ к объектам данных, усовершенствованные средства ввода/вывода и управления выполнением, дополнительные внутренние процедуры, в частности, для работы с битами. Fortran 2008 был опубликован в 2010 году, и в настоящее время является действующим стандартом. Будущее языка Fortran определяет его способность обеспечить высокую производительность вычислений, таким образом, комассивы становятся важнейшим инструментом языка.

Но языку программирования трудно выжить, если о нём мало что известно. Причём должны существовать не только учебники по его синтаксису и семантике, но и книги о практическом применении языка для решения реальных задач. Опыт работы, конкретные методики, а также способы наиболее оптимального использования новых функциональных возможностей должны передаваться новому поколению программистов. В наше время, когда языки программирования не являются «вещью в себе», а всё чаще используются совместно друг с другом или в сочетании с разнообразными инструментальными средствами, необходима именно такая книга как «Современный Fortran: практическое использование».

Автор этой книги постоянно сотрудничает с информационным бюллетенем ACM Fortran Forum и является активным участником группы новостей comp.lang.fortran, где публикует множество полезных советов. Его опыт научного программирования приносит пользу сообществу не только в Нидерландах, где он проживает, но и во всём мире, а статьи по обобщённому программированию и использованию шаблонов проектирования в Fortran содержат много свежих идей. Таким образом, квалификация автора вполне позволяет ему написать книгу, подобную этой.

Но «Современный Fortran: практическое использование» — это не просто сборник предыдущих публикаций. Она содержит логически связное изложение основ, и собственные материалы автора по параллельному программированию на Fortran с использованием библиотек

MPI, OpenMP и комассивов (в кратком изложении), а также описывает использование Fortran для создания графических приложений и для взаимодействия с графическими пользовательскими интерфейсами. В книге много подробных примеров кода, на которые читатель может опираться при создании собственных программ.

Эта книга весьма полезна, и я рекомендую её всем программистам, использующим Fortran. Все мы уже убедились в том, что Fortran продолжает существовать и у него есть будущее.

Майкл Меткалф (Michael Metcalf)
Токио, октябрь 2011 г.

ПРЕДИСЛОВИЕ АВТОРА

Я программирую на языке Fortran уже более 25 лет, сначала на FORTRAN IV, немного позже на FORTRAN 77. В последнее десятилетие прошлого века я и несколько моих коллег прослушали курс по Fortran 90, прочитанный Яном ван Оостервийком (Jan van Oosterwijk) в Техническом университете Делфта. Приблизительно в это же время я присоединился к группе новостей comp.lang.fortran, и многому научился в этом дружелюбном сообществе.

В определённом смысле я обычный программист на языке Fortran. Моя основная специальность — физика, а программирование я начал осваивать ещё во время учёбы, но большую часть практического опыта приобрёл во время работы. Я стал программистом по производственной необходимости, а не из-за особого интереса к всё более изощрённым возможностям программирования вообще и к перспективам их применения в Fortran. Позже я начал писать статьи для информационного бюллетеня ACM Fortran Forum, которые стали отправной точкой для данной книги.

Эта книга не научит вас программированию на языке Fortran. Для этого существует множество специальных учебников ([22], [65]). Цель данной книги — показать, как можно использовать современный Fortran для решения задач, существующих в настоящее время, продемонстрировать, например, что методики, широко распространённые в мире объектно-ориентированных языков, таких как Java и C++, вполне применимы в последней версии Fortran. Более того, в книге описываются некоторые приемы решения задач программирования, которые не так-то просто реализовать с помощью других языков.

Если вы знакомы с языком в основном по старым его версиям, существовавшим до появления Fortran 90, несколько первых глав постепенно познакомят вас с операциями над массивами, с перегружаемыми операциями и с некоторыми другими функциональными возможностями, введёнными в этом стандарте. Кроме того, вы увидите, что при использовании Fortran появилась возможность приме-

нения совершенно различных стилей программирования, присущих в основном *функциональным* языкам программирования. В большинстве глав показано, как применять на практике все эти возможности языка.

В данной книге часто встречаются ссылки на программы, которые я написал и опубликовал на веб-сайте SourceForge, или на программы, к созданию которых я причастен в той или иной степени. Не считайте это рекламой конкретного программного обеспечения или навязыванием своего мнения – это просто моя уверенность в хорошем качестве упоминаемых программ. Во всех примерах, код которых был написан не мной, я старался указывать авторов, но я всего лишь обычный человек, и мог забыть пару имён.

Книги, подобные этой, практически невозможно написать в полной изоляции. Я благодарен Майклу Меткалфу (Michael Metcalf) и Яну Чиверсу (Ian Chivers), редакторам бюллетеня ACM Fortran Forum, всем участникам группы новостей comp.lang.fortran, а также Билу Клебу (Bil Kleb), Полу ван Делсту (Paul van Delst), Рольфу Аде (Rolf Ade), Генри Гарднеру (Henry Gardner), Саймону Гирду (Simon Geard), Рихарду Захенвирту (Richard Suchenwirth), Дэниелу Крафту (Daniel Kraft), Рихарду Майне (Richard Maine), Стиву Лайонелу (Steve Lionel), Кэмерону Лэйрду (Cameron Laird) и Клифу Флинту (Clif Flynt). Спасибо им за обсуждения и обзоры моих работ, за дискуссии по различным аспектам программирования на языке Fortran и программирования в целом, за уделённое мне внимание.

У этой книги есть свой веб-сайт http://flibs.sf.net/examples_modern_fortran.html, на котором размещены полные исходные коды всех примеров. Я протестировал их с помощью компиляторов *gfortran* и *Intel Fortran* главным образом в ОС Windows, но проверял и в Linux. Некоторые из этих программ используют самые последние нововведения в стандарте Fortran, поэтому вам потребуется наиболее свежая версия компилятора.

Ариен Маркус (Arjen Markus)
Роттердам, ноябрь 2011 г.

ГЛАВА 1. Введение в современный Fortran

С момента публикации стандарта FORTRAN 77 в 1978 году язык программирования Fortran претерпел множество изменений [61].[1] Вносимые изменения учитывали как новые подходы к методикам программирования, так и новые разработки в области компьютерной аппаратуры. Язык изначально был задуман как максимально эффективный инструмент вычислений. В самом последнем на момент написания этой книги стандарте, Fortran 2008, введена прямая поддержка параллельной обработки средствами самого языка, то есть основная идея эффективности получила дальнейшее развитие [71].

В этой главе приводится общий обзор различных стандартов, появившихся после FORTRAN 77. Не следует считать это попыткой полного описания всех вводимых стандартами новых функциональных возможностей, для этого потребовалась бы отдельная книга или даже несколько книг. Более подробные описания версий стандартов можно найти в книгах Меткалфа (Metcalf) [63], [65] или Брэйнерда (Brainerd) и др. [36].

1.1. Особенности современного Fortran

Стандарт Fortran 90 ввёл весьма существенные изменения по сравнению с повсеместно распространённым стандартом FORTRAN 77: свободный формат исходного кода, операции с массивами, модули, производные типы и т. п. Чтобы понять, что это значило для програм-

[1] Формально в официальных документах наименование этой версии стандарта должно быть записано прописными буквами: FORTRAN 77. Начиная с версии стандарта Fortran 90, наименование языка пишется строчными буквами (буквами в нижнем регистре).

миста, рассмотрим простую задачу: имеется файл с набором чисел, в каждой строке содержится одно значение (для простоты), необходимо построить простую гистограмму, показывающую распределение этих чисел. Программа, решающая такую задачу на FORTRAN 77, могла выглядеть следующим образом:[2]

```
*
* Построение простой гистограммы.
*

      PROGRAM HIST

      INTEGER MAXDATA
      PARAMETER (MAXDATA = 1000)
      INTEGER NOBND
      PARAMETER (NOBND = 9)
      REAL BOUND(NOBND)
      REAL DATA(MAXDATA)
      INTEGER I, NODATA

      DATA BOUND /0.1, 0.3, 1.0, 3.0, 10.0, 30.0,
     &            100.0, 300.0, 1000.0/

      OPEN( 10, FILE = 'histogram.data', STATUS = 'OLD', ERR = 900 )
      OPEN( 20, FILE = 'histogram.out' )

      DO 110 I = 1,MAXDATA
     READ( 10, *, END = 120, ERR = 900 ) DATA(I)
  110 CONTINUE
*
  120 CONTINUE
      CLOSE( 10 )
      NODATA = I - 1

      CALL PRHIST( DATA, NODATA, BOUND, NOBND )
      STOP
*
* Файл не найден и другие ошибки.
*
  900 CONTINUE
      WRITE( *, * ) 'File histogram.data could not be opened'
     &              'or some reading error'
      END

*
* Подпрограмма для вывода гистограммы.
*
      SUBROUTINE PRHIST( DATA, NODATA, BOUND, NOBND )
```

[2] Данный листинг демонстрирует жёстко фиксированный формат FORTRAN 77: позиции 1–5 – метка, позиция 6 – признак продолжения строки, позиции 7–72 – оператор. Именно в этом и состоит главная задача этого листинга. – *Прим. перев.*

```
      REAL DATA(*), BOUND(*)
      INTEGER NODATA, NOBND

      INTEGER I, J, NOHIST

      DO 120 I = 1,NOBND
     NOHIST = 0
     DO 110 J = 1,NODATA
            IF ( DATA(J) .LE. BOUND(I) ) THEN
                   NOHIST = NOHIST + 1
            ENDIF
  110CONTINUE
     WRITE( 20, '(F10.2,I10)' ) BOUND(I), NOHIST
  120 CONTINUE
      END
```

На Fortran 90 ту же программу можно переписать в так называемом *свободном формате (free form)* с использованием различных функций запросов и операций над массивами:

```
! Построение простой гистограммы.
!
program hist
     implicit none

     integer, parameter :: maxdata = 1000
     integer, parameter :: nobnd   = 9

     real, dimension(maxdata) :: data
     real, dimension(nobnd) :: &
            bound = (/0.1, 0.3, 1.0, 3.0, 10.0, &
                   30.0, 100.0, 300.0, 1000.0/)

     integer :: i, nodata, ierr

     open( 10, file = 'histogram.data', status = 'old', &
            iostat = ierr )

     if ( ierr /= 0 ) then
            write( *, * ) 'file histogram.data could not be opened'
            stop
     endif

     open( 20, file = 'histogram.out' )

     do i = 1,size(data)
            read( 10, *, iostat = ierr ) data(i)

            if ( ierr > 0 ) then
                   write( *, * ) 'Error reading the data!'
```

```
                stop
            elseif ( ierr < 0 ) then
                exit ! Обнаружен признак конца файла.
            endif
        enddo

        close( 10 )
        nodata = i - 1
        call print_history( data(1:nodata), bound )

    contains

    ! Подпрограмма для вывода гистограммы.
    !
    subroutine print_history( data, bound )
        real, dimension(:), intent(in) :: data, bound

        integer :: i

        do i = 1,size(bound)
            write( 20, '(f10.2,i10)' ) &
                bound(i), count( data <= bound(i) )
        enddo
    end subroutine print_history

    end program hist
```

Основные различия между программами:

- стандарт Fortran 90 и все последующие стандарты официально разрешали записывать исходный код программы в свободном формате с использованием букв нижнего регистра, хотя и до этого многие компиляторы, соответствующие стандарту FORTRAN 77, допускали такую запись в качестве неофициального расширения;
- ввод оператора `implicit none` заставляет компилятор следить за тем, чтобы все переменные объявлялись с явно указанным типом. Это позволяет устранить большую группу потенциальных ошибок;
- использование *внутренней подпрограммы (internal routine)*, обозначенной ключевым словом `contains`, позволяет компилятору проверить правильность её действительных аргументов, соответствие их количества и типов.[3]
- исключена необходимость использования оператора `GOTO`,

[3] Это только одно из множества полезных свойств внутренних подпрограмм, подробнее см. раздел 1.2.

имеющего дурную репутацию, то есть его можно заменить «более структурным» аналогом `exit` для завершения цикла `do`;
- появилась возможность использования *сечений массивов (array sections)*, например, `data(1:nodata)`, для передачи только требуемой части массива `data`, а также *функции запроса (inquiry function)* `size()`, возвращающей соответствующее количество элементов массива. Кроме всего прочего это сделало ненужными два аргумента подпрограммы, в которых передавались размеры массивов;
- использование стандартной функции `count()` позволило убрать полностью цикл `do` при определении данных гистограммы.

Тем не менее, следует отметить, что первая программа остаётся абсолютно корректной даже с точки зрения современных стандартов Fortran, что является очень важным аспектом использования данного языка. Это означает, что вы можете постепенно переходить к применению новых функциональных возможностей, а не переписывать всю программу целиком.

1.2. Fortran 90

В стандарт Fortran 90 было введено большое число новых функциональных свойств, из которых наиболее известными стали операции с массивами. Но стандарт определяет и другие важные нововведения:

- оператор `implicit none` требует от пользователя явно объявлять все переменные, а компилятор обязан проверять их. Это позволяет избавиться от опечаток в именах переменных;
- *модули (modules)* в сочетании с операторами или атрибутами `public` и `private` представляют собой эффективные средства деления больших программ на более мелкие части, режим доступа к которым определяется явно в исходном коде. Кроме того, модули позволяют формировать чётко определённые интерфейсы к содержащимся в них подпрограммам, что даёт возможность компилятору выполнить весь комплекс проверок и оптимизаций. Если это недоступно, например, при взаимодействии с подпрограммами на другом языке, можно воспользоваться *интерфейсными блоками (interface blocks)*;
- не только основная программа, но также подпрограммы и функции могут содержать так называемые *внутренние под-*

программы (internal routines). В такой подпрограмме доступны все переменные из внешней содержащей её (под)программы, но кроме того внутренняя подпрограмма создаёт новую *область видимости (scope)*, в которой определяются локальные переменные. Это дополнительный способ деления кода на модули;

- современные компьютеры предоставляют разнообразные средства управления памятью, которые не были так широко распространены во времена стандарта FORTRAN 77, поэтому некоторые положения стандарта Fortran 90 связаны с управлением памятью:
  - разрешены *рекурсивные подпрограммы (recursive routines)*, упрощающие реализацию рекурсивных алгоритмов;
  - с помощью операторов `allocate` и `deallocate` программист может вручную регулировать размер массивов в соответствии с условиями решаемой задачи. Массивы, размер которых можно регулировать вручную, делятся на две категории: динамические (`allocatable`) и статические (`pointer`). Первые предлагают больше возможностей для оптимизации, тогда как вторые дают большую гибкость;
  - помимо прямого изменения размеров массива программист может использовать *автоматические массивы (automatic arrays)* – массивы, которые получают свой размер из формальных аргументов, автоматически создаются при входе в подпрограмму или функцию и автоматически удаляются при выходе;
- *операции с массивами (array operations)* – одно из важнейших нововведений в стандарте Fortran 90, поскольку поощряют лаконичный стиль программирования и существенно упрощают задачу оптимизации для компилятора. Эти операции поддерживаются не только в обычных арифметических выражениях, но и с помощью набора обобщённых стандартных функций.

  Можно работать как с целым массивом, так и с некоторой его частью, определяемой начальной и конечной позициями, и *шагом по индексу (stride)* в любом измерении.

  Кроме того, функции, определённые пользователем, могут возвращать массив как результат, расширяя возможности операций с массивами;
- к традиционному старому *фиксированному формату (fixed form)*, присущему языку Fortran с момента его создания, офи-

циально добавлен *свободный формат (free form)*, в котором столбцы 1–6 уже не имеют какого-либо специального предназначения. Свободный формат придаёт исходному коду более современный вид.

- массивы значений можно формировать «на лету» с помощью так называемых *конструкторов массивов (array constructors)*. Это мощный механизм, удобный для заполнения массивов значениями;
- как и многие современные языки программирования Fortran поддерживает определение новых структур данных. Для этого предназначен механизм *производных типов (derived types)*. Любой производный тип состоит из элементов разных типов – простых, таких как `integer` или `real`, или других производных типов. Производные типы можно применять точно так же как простые – передавать их в подпрограммы, использовать в качестве возвращаемых значений или создавать массивы этих типов.

  Кроме того, производные типы можно использовать для создания связных списков, деревьев и других известных абстрактных типов данных;
- *перегрузка (overloading)* подпрограмм и операций, таких как сложение и вычитание, предоставляет возможность расширения языка новыми полнофункциональными типами. Программист может определить обобщённые имена для специализированных подпрограмм и/или переопределить операции +, – и другие для числовых типов, не регламентируемых стандартом (например, для рациональных чисел).

  Фактически это дает возможность определять собственные операции. Простой пример: предположим, что программа работает с плоскими геометрическими объектами, поэтому имеет смысл определить операцию *пересечения* (intersection) для двух таких объектов `object_a .intersects. object_b`, чтобы заменить вызов функции `intersect_objects( object_a, object_b )`;
- теперь функции и подпрограммы могут иметь *необязательные аргументы (optional arguments)*, при этом функция `present()` позволяет определить наличие или отсутствие аргумента. Также можно вызывать функции и подпрограммы с аргументами, указанными в произвольном порядке, при условии передачи имен формальных аргументов, чтобы компилятор мог приве-

сти в соответствие фактические аргументы с формальными. Ещё одно усовершенствование – можно задать *назначение (intent)* любого аргумента: только для ввода, только для вывода или для ввода/вывода. Это дополнительное средство документирования программы, а также информация для выполнения компилятором оптимизации определённого типа;

- *разновидности типов (kinds)* – это специфический для языка Fortran способ определения характеристик простых типов. Например, для переменных типа `real` можно выбирать одиночную или двойную точность не явным указанием типа (`real` или `double precision`), а с помощью ключевого слова `kind`:

```
integer, parameter :: double = &
       select_kind_real( precision, range )
real(kind=double)  :: value
```

- в дополнение к новой конструкции `select/case`, циклам `do while` и циклам без условия `do`, появилась возможность давать имена всем управляющим структурам. Это упрощает документирование начала и конца таких структур. Чтобы пропустить некоторую часть тела цикла `do`, теперь можно воспользоваться оператором `cycle` (с необязательным именем конкретного цикла) – таким способом можно выйти из нескольких вложенных циклов `do`. Подобным образом оператор `exit` завершает выполнение цикла `do`;
- стандарт Fortran 90 определяет большое количество функций и подпрограмм:
  - числовые функции-запросы для получения свойств модели вещественных чисел;
  - функции обработки массивов, в большинстве случаев принимающие выражения с массивами, как один из аргументов. Например, подсчитать число положительных элементов в массиве можно так:

```
integer, dimension(100,100) :: array
…
write(*,*) 'Number of positive elements: ', &
     count( array > 0 )
```

  - функции для работы с символами и функции для операций с битами;

- к усовершенствованиям системы ввода/вывода относится непродвигающий ввод/вывод (nonadvancing I/O), то есть программа может манипулировать не только целыми записями, но также читать или записывать часть записи в одном операторе, а остальную часть записи в другом.

1.3. Fortran 95

Стандарт Fortran 95 представляет собой скорректированную и дополненную версию предыдущего выпуска стандарта, поэтому различия не столь значительны. Тем не менее, следует обратить внимание на следующие важные изменения:

- согласно стандарту Fortran 90, переменные с атрибутом `pointer` не могут инициализироваться. Начальное состояние не определено ни для связанных, ни для несвязанных переменных. В версии Fortran 95 введена функция `null()` для явной инициализации указателей:

```
real, dimension(:), pointer :: ptr => null()
```

- динамические (`allocatable`) локальные переменные без атрибута `save` автоматически удаляются из памяти при выходе из области видимости (при возврате из подпрограммы или из функции). Это вполне безопасно, поскольку после выхода из области видимости таких переменных, выделенная им память становится недоступной;
- технический отчёт, как подготовительный материал для следующего стандарта, описывает, как динамически размещаемые (`allocatable`) компоненты могут входить в состав производных типов и возвращаться функциями. (Этот технический отчёт в дальнейшем стал частью стандарта Fortran 2003 с некоторыми дополнениями.);
- в Fortran 95 были введены типы подпрограмм `pure` и `elemental`. *Элементные подпрограммы (elemental routines)* освобождают программиста от необходимости писать несколько версий подпрограмм для работы с массивами всех требуемых размерностей. Многие функции и подпрограммы стандартной библиотеки уже имели статус `elemental`, означающий, что они работают с отдельными элементами массивов, которые могут передаваться без соблюдения какого-либо порядка.

После ввода стандарта Fortran 95 программистам была предоставлена возможность самим писать такие подпрограммы.

«Чистые» (`pure`) подпрограммы предоставляют компилятору больше возможностей для оптимизации, так как считается, что они не имеют побочных эффектов.

- следует особо отметить появление оператора `forall`, взятого из языка High Performance Fortran. Он был предназначен для расширения возможностей операций над массивами, но на практике далеко не всегда использовался корректно. (Стандартом Fortran 2008 введена более гибкая конструкция `do concurrent`.)

1.4. Fortran 2003

Fortran 2003 представляет собой существенно переработанную и исправленную версию стандарта, и самым главным изменением является введение поддержки объектно-ориентированного программирования.[4] Но в этой версии появились и другие нововведения, например, стандартизация взаимодействия с подпрограммами (функциями), написанными на языке C. Ниже кратко описаны основные нововведения в стандарте Fortran 2003:

- с поддержкой *объектно-ориентированного программирования (object-oriented programming)* были напрямую связаны некоторые новые функциональные возможности:
  - производные типы теперь могут содержать процедуры (функции и подпрограммы), связанные с данным типом (так, что все переменные этого типа могут пользоваться одним и тем же набором процедур) или с конкретной переменной. Такие процедуры автоматически принимают «свою» переменную (или объект в более широком смысле) как один из аргументов, но при этом программисту предоставляется возможность управления передачей объектов. (Эти вопросы будут рассматриваться более подробно в главе 11.);

[4] Стандарт Fortran 90 определяет достаточно много функциональных свойств языка, позволяющих программисту вплотную приблизиться к этому стилю программирования, но в нём отсутствует наследование, зачастую рассматриваемое как одна из наиболее важных характеристик объектно-ориентированного программирования. Поэтому Fortran 90 иногда называли языком, основанным на концепции объектов (object-based).

- появилась возможность на основе производных типов создавать новые типы, то есть «расширять» производные типы. Это механизм наследования в Fortran. *Расширенный тип (extended type)* может содержать новые компоненты (данные и процедуры), а также переопределять процедуры, связанные с родительским типом, но не имеет права изменять их сигнатуру;
- в оператор `select` добавлена возможность выбора по *типу* переменной. Это важно при работе с так называемыми *полиморфными переменными (polymorphic variables)* — переменными-указателями, тип которых меняется в зависимости от типов связанных с ними переменных в тот или иной момент выполнения программы. Для объявления полиморфных переменных используется ключевое слово `class` вместо `type`.[5]
- чтобы обеспечить большую гибкость механизма расширения типов, для процедур добавлена характеристика *абстрактный интерфейс (abstract interface)*. Вместо определения конкретной подпрограммы интерфейсы объявляют лишь «внешний вид» подпрограммы со списком аргументов и типом возвращаемого значения. В дальнейшем это объявление может служить шаблоном для настоящих подпрограмм;

- поскольку *указатели на процедуры (procedure pointers)* чаще встречаются в виде компонентов производных типов, их можно использовать как обычные переменные;
- для получения некоторых особых эффектов была введена концепция *внутренних модулей (intrinsic modules)*. Примером таких особых эффектов может служить управление моделью представления вещественных чисел: режим округления, эффект исключений, связанных с операциями над вещественными числами, а также организация взаимодействий с программами на языке C, так как при этом иногда требуется соблюдение разных соглашений об именовании и формате вызова функций;
- улучшено управление памятью:
  - длина строки символов теперь можно определить с помощью оператора `allocate`;

[5] В [73] сравнивается терминология объектно-ориентированных концепций в языках Fortran и C++.

  - при некоторых условиях динамический массив можно автоматически привести к корректному размеру;
  - допускается перемещать выделенную память из одной переменной в другую с помощью подпрограммы `move_alloc()`. Это упрощает увеличение размера массива;
- появилась весьма полезная возможность *потокового доступа (stream access)* к файлам, которая позволяет читать и записывать содержимое файлов не только целыми записями. В частности, теперь можно читать и записывать двоичные файлы, не имеющие внутренней структуры. Для старых файлов, *не имеющих формата, (unformatted files)* требовалась структура, основанная на записях, что затрудняло взаимодействие с другими языками программирования;
- стандарт Fortran 2003 также регламентирует доступ к системной среде в форме *переменных окружения (environment variables)* и к аргументам командной строки, заданным при запуске программы. Раньше для доступа к этой информации программист был вынужден использовать специфические возможности конкретного компилятора.

1.5. Fortran 2008

В версии стандарта Fortran 2008 самым важным нововведением являются *комассивы (coarrays)*, но все прочие изменения языка были незначительными. Более подробно комассивы рассматриваются в главе 12 [72], [71]. Помимо этого в стандарте определено несколько новых конструкций и ключевых слов, а также новые стандартные функции:

- комассивы представляют собой механизм параллельных вычислений из категории *«разделённое глобальное адресное пространство» (Partitioned Global Address Space, PGAS)*. По существу этот механизм обеспечивает доступность данных нескольким копиям программы и освобождает программиста от обязанности выбирать конкретный способ передачи данных. Компилятор должен сгенерировать код, выполняющий эту задачу рационально и эффективно;
- массивы могут быть определены как *непрерывные (contiguous)*. Это позволяет компилятору повысить степень оптимизации кода, так как элементы массива в памяти следуют друг за другом;

- конструкция `block – end block` определяет локальную область видимости в программе или в подпрограмме, так что можно объявлять новые переменные, которые будут существовать только внутри этого блока;
- модули, введённые в стандарте Fortran 90, являются важным механизмом деления больших программ на части. Но сам по себе этот механизм не может обеспечить высокую степень модульности, поскольку компилироваться должен весь модуль, целиком. В стандарт Fortran 2003 были введены *подмодули (submodules)* для устранения проблем при обработке чрезвычайно больших файлов исходного кода с крупными модулями. Теперь к этому механизму добавилась возможность *импорта (import)*, которая используется в интерфейсных блоках для импорта определений из внешнего модуля, что исключает необходимость определения второго модуля, включающего в себя только требуемые определения;
- ещё один шаг к уменьшению зависимости от оператора `GOTO`, – оператор `exit`, выполняющий переход к концу блока `if` или `select`;
- оператор `do concurrent` можно рассматривать как более гибкую альтернативу оператору/блоку `forall`. Он сообщает компилятору, что данный код может выполняться в параллельном режиме (в данном случае параллельное выполнение осуществляется в рамках одной и той же *копии*, а не между несколькими копиями, как при использовании комассивов);
- *внутренние процедуры (internal procedures)* теперь можно передавать как действительные аргументы, при этом внутренним процедурам доступны переменные в подпрограмме, где определены такие процедуры. Это упрощает решение некоторых проблем с интерфейсами (см. главу 5);
- в набор стандартных функций включены разнообразные функции Бесселя (Bessel functions) и функции запросов при работе с битами.

1.6. Что осталось неизменным

Все описанные выше новые функциональные возможности языка Fortran не повлияли на существующие программы, соответствующие старым стандартам. Исключение составили лишь те свойства, кото-

рые были удалены или выведены из употребления (их применение не рекомендуется).[6]

Старые программы должны успешно компилироваться, при условии отсутствия в них удалённых и давно забытых свойств и возможностей языка, таких как вычисляемый оператор `GOTO`.[7]

В действительности, помимо упомянутой поддержки старого кода, в реализации современного Fortran многое осталось неизменным:

- Fortran нечувствителен к регистру символов, в отличие от многих языков программирования, используемых в наши дни;
- в языке отсутствует понятие *файла* как организационной и структурной единицы исходного кода. В частности, исходный код не может находиться вне программы, подпрограммы, функции или модуля;
- по своей сущности Fortran ориентирован на эффективное выполнение. Это особенно заметно в последнем стандарте Fortran 2008, где все нововведения направлены на то, чтобы помочь компилятору в создании быстрых и эффективных программ.

  Тонкости реализации нововведений могут удивить программистов, использующих языки, подобные C (см. приложение Б);
- тип данных любого выражения или подвыражения зависит только от операндов и операции, но не от контекста. Это оправдано с точки зрения существенного упрощения программы, но опять-таки может давать неожиданные результаты (см. приложение Б). Вот один небольшой пример:

```
real :: r
r = 1 / 3
```

Здесь переменная `r` получает значение `0.0`, а не `0.333333...`, так как сначала выполняется операция деления двух целых чисел, результатом которой является новое целое число, а затем полученное целое число преобразуется в вещественное.

Точно такое же разделение операций и преобразования их результатов справедливо для операций с массивами:

```
integer, dimension(10) :: array
```

[6] Самой значимой из всех удалённых функциональных возможностей является использование переменных типа real для управления циклом do ([68], раздел 2.1.5).

[7] Большинство компиляторов продолжают поддерживать эти устаревшие возможности, поэтому старые программы можно компилировать и запускать.

```
array = 2 * array(10:1:-1)
```

Эта инструкция выполняется (по крайней мере, теоретически) следующим образом:

```
integer, dimension(10) :: array
integer, dimension(10) :: tmp
integer                :: i

! Сначала вычисляется правая часть выражения, и результат
! сохраняется во временном массиве.
do i = 1,10
     tmp(i) = 2 * array(11-i)
enddo

! Теперь результат правой части выражения копируется в массив
! слева от знака равенства.
do i = 1,10
     array(i) = tmp(i)
enddo
```

Это означает, что код, подобный приведённому выше, всегда работает без проблем, даже если в правой части выражения содержатся те же элементы массива, что и в левой, но в другом порядке.