С. Немнюгин
О. Стесик

bhv

ФОРТРАН
в задачах и примерах

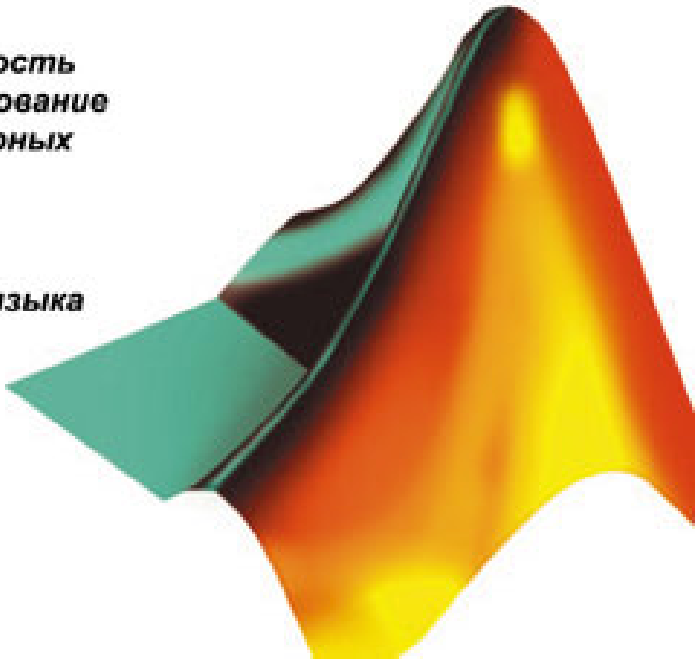
Эффективно решай сложные вычислительные задачи!

Основные возможности языка

Программирование вычислений

Многопоточность и программирование для многоядерных архитектур

Справочник по функциям языка



С. А. Немнюгин
О. Л. Стесик

ФОРТРАН
в задачах и примерах

Санкт-Петербург
«БХВ-Петербург»
2008

УДК 681.3.068+800.92Fortran
ББК 32.973.26-018.1
Н50

Немнюгин, С. А.

Н50 Фортран в задачах и примерах / С. А. Немнюгин, О. Л. Стесик. — СПб.: БХВ-Петербург, 2008. — 320 с.: ил.

ISBN 978-5-94157-873-3

Книга представляет собой сборник примеров программ и задач для самостоятельного решения по программированию на одном из самых эффективных языков разработки вычислительных приложений — языке Фортран. Примеры и задачи различной сложности демонстрируют основные возможности языка. Дается краткое описание OpenMP — стандартного средства разработки программ для многоядерных процессоров. В книге содержится описание встроенных функций языка, что дает возможность использовать ее в качестве справочника по программированию на языке Фортран.

Для программистов

УДК 681.3.068+800.92Fortran
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Юрий Рожко*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Виктория Пиотровская*
Дизайн серии	*Игоря Цырульникова*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 25.06.08.
Формат 60×90 1/16. Печать офсетная. Усл. печ. л. 20.
Тираж 2500 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.60.953.Д.003650.04.08 от 14.04.2008 г. выдано Федеральной службой по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-94157-873-3

Оглавление

Предисловие **1**

Глава 1. Компиляция, выполнение и отладка программ **3**

Как создается программа 4
Компилятор фирмы Intel 6
Компиляторы GNU Fortran 12
Система программирования Compaq Visual Fortran 17
Система программирования Sun Studio 18
Программы-отладчики 19

Глава 2. Элементы языка **25**

Языки программирования 27
Алфавит и лексемы языка Фортран 28
Формат записи исходного текста программы 28
Как устроена программа 31
Типы данных 33
Переменные 34
Константы 35
Массивы 37
Комментарии 38
Операторы 38
Условный оператор *if...then...endif* 40
Условный оператор *if...then...else...endif* 41
Оператор цикла со счетчиком *do...end do* 41
Вопросы и задания 44

Глава 3. Операторы описания 47

Основные сведения 47
Операторы описания для встроенных типов 47
Оператор описания производного типа 49
Неявное определение типа 50
Оператор *IMPLICIT* 51
Атрибуты 52
Структура оператора описания 54
Инициализирующие выражения 57
Типы и разновидности типов данных 58
Вопросы и задания 62

Глава 4. Арифметические выражения 65

Преобразование типов 68
Инициализация переменных 74
Особенности машинной арифметики 75
Оптимизация вычислений 78
Вопросы и задания 82

Глава 5. Логические выражения 85

Отношения 85
Логические выражения 86
Вопросы и задания 90

Глава 6. Циклы 91

Задачи 104

Глава 7. Условные операторы и ветвления 113

Задачи 124

Глава 8. Структура программы 127

Порядок операторов 128
Главная программа 128
Внешние подпрограммы 129
Модули 131

Внутренние подпрограммы ... 135
Параметры подпрограмм ... 136
Интерфейсы подпрограмм ... 142
Области видимости имен и меток ... 146
Задачи ... 146

Глава 9. Массивы ... 151

Подобъекты массивов ... 154
Конструкторы массивов ... 157
Встроенные функции для работы с массивами ... 159
Дополнительные свойства массивов ... 164
Элементные встроенные функции и операции ... 165
Оператор и конструкция *where* ... 166
Массивы-маски ... 167
Оператор и конструкция *forall* ... 168
Автоматические массивы и массивы подразумеваемой формы ... 170
Размещаемые (динамические) массивы ... 171
Задачи ... 172

Глава 10. Ввод и вывод ... 179

Форматирование ввода-вывода ... 184
Задачи ... 189

Глава 11. Файлы ... 193

Задачи ... 213

Глава 12. Встроенные подпрограммы ... 217

Оператор *intrinsic* ... 217
Справочные функции ... 218
Встроенные процедуры определения даты и времени ... 221
Элементные функции ... 222
Математические функции ... 223
Функции преобразования и переноса типов ... 227
Случайные числа ... 228

Операции над массивами 229
Функции редукции массивов 233
Операции с векторами и матрицами 235
Текстовые функции 237
Процедуры для работы с двоичными разрядами 240
Задачи 242

Глава 13. Производные типы и указатели 245

Определение производных типов 245
Атрибуты *public* и *private* 249
Указатели 250
Задачи 257

Глава 14. Программируем на Фортране для многоядерных процессоров 259

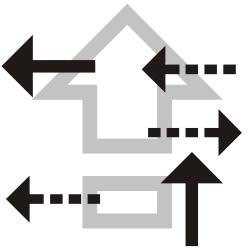
OpenMP-программа 259
Как распараллелить программу с помощью OpenMP 262
Директивы OpenMP 263
Операторы OpenMP 266
Подпрограммы OpenMP 267
Задачи 268

Глава 15. Разные задачи 277

Литература 295

Предметный указатель 297

Глава 1



Компиляция, выполнение и отладка программ

Квалифицированный программист на Фортране не только хорошо знает синтаксис языка, приемы эффективного программирования, но и умеет пользоваться компиляторами, отладчиками и другими вспомогательными инструментами программирования. Знакомство с основами программирования на Фортране мы начнем с краткого обзора наиболее доступных компиляторов и средств отладки. Это позволит читателю сразу приступить к работе с примерами программ в следующих главах книги.

Имеется большое и все возрастающее число компиляторов языка Фортран, как коммерческих, так и свободно распространяемых. Свободно распространяются для некоммерческого использования, например, некоторые компиляторы фирмы Intel, компилятор g95 и т. д. Последний разрабатывается в рамках проекта GNU, целью которого является создание и распространение бесплатного программного обеспечения.

Среди вспомогательных средств разработки программ находятся конвертеры с Фортрана на другие языки, например, C, различные системы отладки программ, поддерживающие работу с Фортраном, а также интегрированные среды, профилировщики и т. д. Среди них следует упомянуть такие программы, как dbx, gdb, ElectricFence и другие.

Как создается программа

Создание программы — сложный процесс, состоящий из нескольких этапов. Начинается он с разработки алгоритма и написания текста программы на наиболее подходящем для решения поставленной задачи языке программирования. Затем создается исполняемый файл программы. В многоступенчатом процессе создания программы можно выделить следующие этапы:

1. *Создание файла с исходным текстом программы*. Для этого используется текстовый редактор, самостоятельный или входящий в состав какой-либо интегрированной среды. *Интегрированная среда* представляет собой специальную программу, обеспечивающую удобный доступ к различным инструментам разработки программ.
2. Файл с исходным текстом программы может обрабатываться препроцессором. Это необязательный этап, чаще всего он пропускается.
3. *Компиляция* — создание объектного файла (или нескольких объектных файлов, если программа достаточно сложная). Объектный файл содержит исполняемый код программы на машинном языке, который, однако, еще не готов к выполнению. В него не включены код запуска программы и коды библиотечных подпрограмм. Для разных частей программы могут быть подготовлены разные объектные файлы.
4. Создание исполняемого файла с помощью программы-*компоновщика*. Компоновщик "собирает" исполняемый файл из объектных файлов программы и необходимых библиотек, а также включает в него код запуска. Компоновку (сборку) завершает процесс создания исполняемого файла.

Во время компиляции выполняется проверка соответствия записи программы синтаксическим правилам языка и, если обнаружена *синтаксическая ошибка*, выдается сообщение, а объектный или исполняемый файл не создается. В диагностическом сообщении обычно содержится числовой код ошибки, ее краткое разъяснение и положение неправильной конструкции в исходном тексте

программы. При получении такого сообщения исходный файл необходимо отредактировать и исправить ошибку.

Компилятор может выводить *предупреждения*. Это происходит, если исходный текст программы не содержит синтаксических ошибок, но какие-то конструкции вызывают у компилятора "подозрение" в правильности их использования. Приведем пример простейшей программы на языке Фортран:

```
program doloo
 j = 0
 do i = 1, k
       j = j + 1
 end do
 print *, j
end program doloo
```

При компиляции этой программы может быть выведено предупреждение:

```
Warning: Variable K is used before its value has been defined
```

Оно означает, что значение переменной `k` в момент ее использования не определено и, хотя формальных нарушений правил записи операторов программы здесь нет, программа, будет работать неправильно. Это пример *логической ошибки*.

Информационные сообщения, содержат предложения по оптимизации кода и сведения по оптимизации, выполненной компилятором. Обычно вывод таких сообщений отключен, но он может быть активизирован с помощью соответствующих ключей компиляции.

По умолчанию любой компилятор выполняет шаги 2, 3 и 4. Остановить процесс на любом этапе или добавить обработку препроцессором можно с помощью ключей компиляции.

В зависимости от типа файла, создаваемого на каждом этапе обработки, его имени присваивается определенный *суффикс* (расширение, то есть часть имени, идущая после точки). Исходный текст обычно содержится в файле, имя которого имеет суффикс f, f90 или f95. Файлы модулей имеют расширение mod. Объектный

файл имеет расширение o. Имя исполняемого файла в MS Windows имеет суффикс exe, а в операционных системах (ОС) UNIX оно может быть любым (по умолчанию a.out).

Процесс создания программы не прекращается после того, как удается получить исполняемый файл. Важнейшим и, как правило, неизбежным этапом разработки является *отладка*. Отладка заключается в поиске логических ошибок реализации алгоритма. Поиск *синтаксических ошибок* берет на себя компилятор. Выполнить проверку правильности работы программы сложнее. Если оказывается, что программа работает не так, как предполагал программист, приходится выяснять, что происходит во время ее выполнения. Для этого необходимо "заглянуть" внутрь программы, посмотреть, как изменяются значения переменных, какие значения передаются в подпрограммы в качестве аргументов и т. д. Помощником программиста в этом непростом деле являются специальные программы-отладчики.

Компилятор фирмы Intel

Компилятор Фортрана фирмы Intel (Intel® Visual Fortran Compiler, адрес сайта **www.intel.com**), который в дальнейшем для краткости мы будем обозначать аббревиатурой *IFC*, предназначен для архитектур Intel® IA-32, Intel® EM64T и Intel® IA-64 (Itanium), и операционных систем Microsoft Windows 2000/XP/Vista, Linux и MacOS.

Среди особенностей компилятора IFC отметим совместимость с *Microsoft Visual Studio* и поддержку спецификации *OpenMP* (Open MultiProcessing, открытый стандарт многопроцессорной обработки), которая позволяет создавать код для параллельного выполнения на многопроцессорных вычислительных системах с общей памятью или компьютерах с многоядерными процессорами. В состав пакета входят: компилятор, отладчик, инструмент визуализации массивов.

Есть два способа использования компилятора IFC — в режиме командной строки и с использованием графического интерфейса

(в частности, Microsoft Visual Studio). Для вызова компилятора из текстовой консоли IFC должны быть заданы пути поиска всех необходимых файлов, включая исполняемые и включаемые файлы, а также библиотеки. Для компиляции и сборки программы в IFC из среды Microsoft Visual Studio ее следует соответствующим образом настроить. Описание настройки следует искать в справочном руководстве по среде.

Запустить IFC в Microsoft Windows XP можно следующим образом. Необходимо зайти в меню **Программы** с помощью кнопки **Пуск** (Start), выбрать подраздел **Intel(R) Software Development Tools**, войти в подменю **Intel(R) Fortran Compiler** и щелкнуть мышью на строке **Fortran Build Environment for Applications**. Далее в открывшемся окне текстовой консоли запустить компилятор с помощью команды:

```
ifort [ключи] имена_файлов [/link библиотека.lib]
```

Ключи являются необязательной частью командной строки и используются для того, чтобы выбрать определенный режим работы компилятора.

> ***Примечание***
>
> В дальнейшем в квадратных скобках будет указываться необязательная часть команды.

При работе в ОС MS Windows ключ IFC начинается с наклонной черты, например:

```
/1 /4I4
```

Регистр букв в ключах имеет значение.

Ключи, указанные в одной команде, применяются ко всем файлам, которые в ней указаны. С ключами могут использоваться значения-аргументы, в качестве которых используются имена файлов, строки, буквы, численные значения. Если строковое значение содержит пробелы, то оно должно быть заключено в кавычки.

`Имена_файлов` в командной строке при запуске компилятора задают те файлы, которые будут обрабатываться компилятором.

Если в строке используется несколько имен файлов, то их следует разделить пробелами.

Ключ `/link` используется для того, чтобы присоединить библиотеку или объектный файл, полученный ранее при раздельной компиляции, или для того, чтобы задать ключи компоновщика, отличные от принятых по умолчанию. Все ключи компилятора должны предшествовать ключу `/link`.

Положение модуля (файл с расширением mod) можно задать с помощью ключа `/module:<путь>`. Если компилируемый файл содержит модули, компилятор создает для каждого из них свой mod-файл. Имя файла совпадает с именем модуля и содержит буквы в верхнем регистре.

При работе над небольшим проектом все файлы программы обычно содержатся в одном каталоге. В этом случае компиляция выполняется обычным образом. Если же проект большой и содержит большое количество файлов, находящихся в разных каталогах, то IFC использует для поиска mod-файлов ключ `/I<каталог>`, например:

```
ifort /c user_mod.f90 /I\usr\svroman\project
```

Результатом выполнения команды:

```
ifort progr.f90
```

является исполняемый файл progr.exe. В результате компиляции создается также объектный файл progr.obj, который сохраняется в текущем каталоге.

Допускается одновременная компиляция нескольких файлов:

```
ifort pr1.f90 pr2.f90
```

В результате выполнения этой команды будут созданы два объектных файла pr1.obj и pr2.obj, будет создан исполняемый файл pr1.exe и все файлы сохраняются в текущем каталоге. Имя исполняемого файла можно задать с помощью ключа `/exe:<файл>`.

Алфавитный список некоторых ключей компилятора приведен в табл. 1.1. Запись вида `/4I{2|4|8}` означает, что у данного ключа есть три варианта: `/4I2`, `/4I4` и `/4I8`, а в записи `/W{n}` `n` — одно из допустимых значений (список допустимых значений приводится в столбце "Описание").

Таблица 1.1. *Некоторые ключи компилятора IFC*

Ключ	**Описание**	**Значение по умолчанию**
`/help`	Вывод справочной информации	`OFF`
`/4{Y|N}b`	Включает (или отключает) расширенный контроль ошибок времени выполнения	`OFF`

`/4{Y|N}f`	Определяет фиксированный формат записи исходного текста для следующих типов файлов:
	`/4Yf: .f, .for, .ftn;`
	`/4Nf: .f90`
		`OFF`

`/c`	Завершает процесс компиляции после создания объектных файлов	`OFF`
`/I<каталог>`	Определяет дополнительный каталог для поиска включаемых файлов, имена которых не начинаются с наклонной черты — символа *слэш* (slash) (*/*)	`OFF`
`/module:<путь>`	Указывает каталог, в котором находятся файлы модулей	`/nomodule`
`/O{n}`	Включает оптимизацию исполняемого файла по производительности. Наиболее "осторожный" режим оптимизации включается при n = 1. По умолчанию используется значение n = 2. Наиболее "агрессивный" вариант оптимизации соответствует n = 3. Включает преобразования циклов, но существенный выигрыш дает не всегда. Более подробную информацию по данной теме можно найти в руководстве пользователя, которое включено в комплект поставки	`OFF`
`/Od`	Запрещает оптимизацию	`OFF`

Таблица 1.1 *(окончание)*

Ключ	Описание	Значение по умолчанию
`/Qprec`	Осуществляет повышение точности операций с плавающей точкой. Скорость выполнения программы уменьшается	`OFF`
`/Zi`, `/Z7`	В объектный код добавляются таблица символов и номера строк, что позволяет применять отладчики на уровне исходного текста	`OFF`

Компилятор IFC интерпретирует тип входного файла по расширению его имени. Возможные варианты приведены в табл. 1.2.

Таблица 1.2. *Интерпретация файлов компилятором IFC*

Суффикс	Интерпретация	Действия
lib	Библиотечный файл, содержащий объектный код	Обрабатывается компоновщиком
f ftn for	Исходный текст на Фортране в фиксированном формате	Обрабатывается компилятором
fpp	Исходный текст на Фортране в фиксированном формате	Обрабатывается препроцессором fpp, а затем компилируется IFC
f90	Исходный текст программы на Фортране 90/95 в свободном формате	Обрабатывается компилятором
obj	Откомпилированный объектный файл	Обрабатывается компоновщиком

Для отладки программы иногда удобно включить в ее текст операторы, выводящие значения некоторых переменных. Эти операторы нужны не всегда, поэтому полезным оказывается один

из ключей отладки `/Qd_lines`, который действует следующим образом. В первой позиции каждой строки, содержащей отладочный оператор вывода, находится символ `D`. При компиляции программы с ключом `/Qd_lines` этот символ замещается пробелом и строка обрабатывается обычным образом, поэтому при выполнении команды:

```
ifort /Qd_lines pro1.f
```

исходный текст:

```
do i = 1, n
a(i) = i**2 + 1
d write (*, *) a(i)
end do
```

читается так:

```
do i = 1, n
a(i) = i**2 + 1
write (*, *) a(i)
end do
```

При отладке полезно запретить все виды оптимизации. Это можно сделать с помощью ключа `/Od`.

Оптимизацией называют такое преобразование программы, которое приводит к увеличению скорости ее работы (производительности) или улучшению других показателей. Оптимизация на этапе компиляции может включаться соответствующими ключами. Среди них `/O1`, `/O2` и `/O3`. Ключи `/O1` и `/O2` соответствуют уровню, на котором оптимизируются: использование регистров, последовательность выполнения команд, а также выполняется исключение общих подвыражений, удаляются неиспользуемые фрагменты кода и т. д. На третьем уровне оптимизируются циклы, используется упреждающая выборка команд, выполняются другие действия. Эта оптимизация применяется в тех программах, в которых велика доля операций с плавающей точкой.

Ключ `/Op` запрещает те виды оптимизации кода, которые могут привести к потере точности. Процессоры Intel обычно хранят результаты операций с плавающей точкой в 80-разрядных регист-

рах. Это превышает разрядность значений с двойной точностью, поэтому при сохранении результатов в памяти выполняется округление. С данным ключом на платформе IA-32 вещественные переменные не хранятся в регистрах, операции не оптимизируются, деление, например, не заменяется умножением на величину, обратную знаменателю. Не выполняется и перегруппировка операндов. При вычислении выражений используются все 80 разрядов, а при присваивании результата переменной с простой и двойной точностью выполняется округление до 32 (в первом случае) или до 64 (во втором случае) разрядов. С ключом `/Qop` соблюдаются и некоторые другие условия.

При компиляции IFC может выполнять оптимизацию использования подпрограмм. Такая оптимизация называется межпроцедурной и включает, в частности, подстановки кода подпрограмм, а также некоторые другие приемы. Для межпроцедурной оптимизации используются ключи `/Qip` и `/Qipo`.

При работе с компилятором для операционной системы Linux изменяется способ задания ключей. Вместо символа слэш "`/`" используется дефис "`-`".

Компиляторы GNU Fortran

Фортран GNU 77 (g77) представляет собой систему программирования, которая создана в рамках проекта GNU (*GNU* — это рекурсивный акроним от "GNU's Not UNIX(tm"), обозначающий проект по созданию свободно распространяемого программного обеспечения) и используется в операционной системе Linux. Систему программирования Фортран GNU 77 в дальнейшем будем для краткости обозначать G77. G77 поддерживает стандарт ANSI Фортран 77, а также некоторые расширения Фортрана, в том числе, частично Фортран 90. Содержит следующие компоненты:

- модифицированную версию компилятора gcc;
- g77 — фронтальную часть системы G77. Она "знает", какие библиотеки необходимы для компоновки программ на Фор-

тране. Команда `g77` выполняет анализ командной строки и после необходимой модификации передает ее команде `gcc`;

- библиотеку времени выполнения libg2c, которая обеспечивает возможности языка Фортран, не поддерживаемые напрямую машинным кодом, сгенерированным на этапе компиляции;
- собственно компилятор, внутреннее имя которого f771. Он не генерирует машинный код напрямую, а создает ассемблерный код, который затем обрабатывается программой as. Программа f771 состоит из двух частей. Одна из них называется *GNU Back End* (GBE), и "умеет" генерировать быстрый исполняемый код для ряда платформ. Вторая часть f771 обрабатывает программы, написанные на Фортране, взаимодействует с GBE, отвечает за правильное использование языка и является источником большинства предупреждений и информационных сообщений. Эту часть называют Fortran Front End.

Перейдем к обзору основных ключей системы G77. Этот перечень неполный. Для детального знакомства с системой читателю придется изучить руководство пользователя и, возможно, другие документы.

Команда `g77` поддерживает все ключи команды `gcc` и наоборот. Некоторые ключи имеют "положительную" и "отрицательную" формы:

`-f<ключ>`

и

`-fno-<ключ>`

соответственно. Здесь `<ключ>` задает конкретный ключ. В первом случае выполняется активизация некоторой функции или режима работы системы, а во втором эта функция (режим) отключаются.

Компиляция с помощью G77 может включать четыре этапа: обработка препроцессором, генерация кода, ассемблирование, компоновка. На трех первых этапах обрабатывается файл с исходным текстом программы, результатом является создание объектного файла. При компоновке все объектные файлы объединяются

в один исполняемый файл. При раздельной компиляции файлов проекта некоторые объектные файлы могут быть подготовлены заранее.

Действия компилятора определяются суффиксом имени исходного файла. Если суффикс имеет вид .f, .for или .FOR, то считается, что файл содержит исходный текст программы на Фортране.

Если суффикс имеет вид .F, .fpp или .FPP, то файлы содержат текст на Фортране, который должен быть обработан препроцессором cpp.

В ОС UNIX обычно используются имена вида file.f и file.F. Если в операционной системе в именах файлов регистры не различаются, то обычно используются имена file.for и file.fpp.

Ключи управления выводом предупреждений обычно начинаются символами `-W`. У каждого из этих ключей есть как положительная, так и отрицательная форма. Последняя начинается с `-Wno-` и отключает вывод предупреждений определенного типа. Некоторые из этих ключей приведены в табл. 1.3.

Таблица 1.3. *Ключи G77, управляющие выводом предупреждений*

Ключ	Описание
`-fsyntax-only`	Выполняет только проверку на наличие синтаксических ошибок
`-w`	Подавляет вывод всех предупреждений
`-Wno-globals`	Подавляет вывод предупреждений об использовании имени одновременно в качестве глобального имени (имя процедуры, функции, общего блока и т. д.) и неявном использовании того же имени в качестве встроенной функции. Подавляет и вывод предупреждений о несоответствии обращений к глобальным процедурам и функциям: разное количество аргументов или разный тип аргументов
`-Wunused`	Выводит предупреждение, если переменная не используется

***Таблица 1.3** (окончание)*

Ключ	Описание
`-Wuninitialized`	Выдает предупреждение об использовании неинициализированной автоматической переменной. Эти предупреждения могут выводиться только во время компиляции с оптимизацией, потому что для них требуется информация, которую можно получить только во время оптимизации. Без ключа оптимизации эти предупреждения не выводятся. Предупреждения не формируются и для массивов, даже если они хранятся в регистрах. Предупреждение может не выводиться и для переменной, применяемой только для вычисления значения, которое больше не используется
`-Wall`	Осуществляет объединение ключей `-Wunused` и `-Wuninitialized`
`-Wsurprising`	Выполняет вывод предупреждений о таких конструкциях, которые могут по-разному обрабатываться разными компиляторами и на разных платформах, приводя порой к неожиданным для программиста результатам. Среди таких конструкций следующие друг за другом символы арифметических операций, ситуация, строго говоря, недопустимая, но в некоторых реализациях Фортрана она разрешена. В руководстве по G77 приводится пример арифметического выражения `2**-2 * 1`, значение которого при компиляции с помощью G77 равно .25, а другие компиляторы могут приводить к результату 0.
`-Werror`	Превращает все предупреждения в сообщения об ошибках с соответствующими последствиями для процесса компиляции

Некоторые из ключей отладки и оптимизации G77 приведены в табл. 1.4.