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

Фортран занимает почетное место среди современных языков программирования. Создателям языка удалось найти замечательный компромисс между удобством программирования и эффективностью программ, написанных на этом языке. Синтаксис языка обеспечивает максимальную эффективность автоматической оптимизации исполняемого кода. Это позволило создать оптимизирующие компиляторы, поставившие вычислительные возможности программ на Фортране вне конкуренции. Язык оснащен богатым набором встроенных математических функций и функций ввода-вывода, что существенно упрощает процесс программирования вычислительных задач. Фортран продолжает развиваться (в настоящее время готовится к выпуску стандарт под условным названием Фортран 2008), оставаясь востребованным языком, на котором пишутся программы, предназначенные для решения сложных задач.

Успех языка во многом был предопределен личностью человека, который руководил его разработкой. Это — Джон Бэкус. "Первую скрипку" в процессе работы над Фортраном, языком, предназначенным для программирования вычислений, играл человек с хорошим математическим образованием и большим опытом численных расчетов.

Фортран постоянно обновляется. Примерно один раз в 10 лет выходит новый стандарт языка, учитывающий современное состояние программирования с одной стороны и пожелания программистов-прикладников с другой. Фортран впитывает те достижения

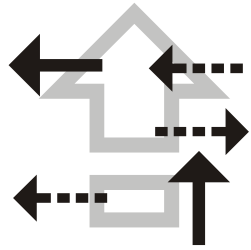
Computer Science, которые действительно необходимы и полезны при программировании вычислений и не ухудшают сколько-нибудь заметным образом их скорость. Строгая стандартизация и постоянное обновление позволяют защитить инвестиции в прикладное программное обеспечение и сделать его универсальным. Таким образом, Фортран сочетает постоянное обновление и строгое следование стандартам.

В настоящее время разработчики программ на Фортране чаще всего используют стандарты Фортран 77 и Фортран 90. Принят и реализован в ряде компиляторов стандарт Фортран 2003. Благодаря возможностям сети Интернет доступны различные, в том числе и бесплатные, компиляторы и интсрументы разработки программ на Фортране.

Настоящая книга предназначена для того, чтобы познакомить читателя с основами программирования на Фортране. Мы ограничиваемся кратким описанием синтаксиса языка, делая упор на примеры программ и задачи. Для ряда задач приводятся решения. Читатель, заинтересованный в более глубоком изучении Фортрана, может обратиться к литературе, список которой приводится в конце книги.

В этой книге мы ориентируемся на Фортран 90, который достаточно распространен в среде прикладных программистов и научно-технических работников.

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-

транe. 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.

Таблица 1.4. *Ключи G77, используемые для отладки и оптимизации*

Ключ	Описание
`-g`	При компиляции с этим ключом генерируется отладочная информация, которая сохраняется в формате операционной системы. С этой информацией может работать, например, отладчик gdb или dbx. Некоторые виды диагностики работают только совместно с ключами оптимизации. Среди них выявление неинициализированных переменных, поскольку в этом случае следует использовать ключи `-O` или `-O2`
`-ffast-math`	Включение "быстрой арифметики", что позволяет увеличить скорость выполнения некоторых программ. Активизируются ключи `-funsafe-math-optimizations`, `-ffinite-math-only` и `-fno-trapping-math`

Таблица 1.5. *Ключи G77, управляющие генерацией исполняемого кода программы*

Ключ	Описание
`-fno-silent`	При компиляции программы в стандартный вывод выводятся имена блоков программы
`-finit-local-zero`	Все локальные переменные и массивы инициализируются нулевыми значениями, за исключением тех, которые используются в `common`-блоках и не передаются в качестве аргумента
`-fno-globals`	Сообщения о проблемах взаимодействия между процедурами, такими, например, как несоответствие типов аргументов, переводятся в разряд предупреждений
`-ffortran-bounds-check`	Проверка выхода значений индексов массива за установленные для них границы во время выполнения программы

Основным ключом поиска каталога с включаемыми файлами, имена которых указаны в операторе `include`, является:

`-I<`*`каталог`*`>`

Пробелы между `-I` и именем каталога не допускаются.

Ключ поиска каталога с библиотечными файлами:

`-L<`*`каталог`*`>`

Он обычно используется совместно с ключом:

`-l<библиотечный_файл>`

Ключи генерации кода позволяют управлять свойствами исполняемого файла программы. Большинство из них имеют как положительную, так и отрицательную формы. Некоторые из этих ключей приведены в табл. 1.5.

Ключ `-c` используется для того, чтобы создать только объектный файл, а ключ `-o` позволяет указать явным образом имя исполняемого файла (по умолчанию имя исполняемого файла в ОС UNIX — a.out).

Фортран G95 поддерживает стандарт Фортран 90/95 (частично и Фортран 2003) и распространяется свободно для разных операционных систем. Вызывается командой:

`g95 [ключи] <файлы>`

Многие ключи этого компилятора совпадают с ключами G77, хотя есть и специфические, в частности:

`-fmod=<каталог>`

позволяет указать каталог, содержащий модули.

Система программирования Compaq Visual Fortran

Система программирования Compaq Visual Fortran (CVF) предназначена для работы в операционной системе MS Windows и включает в свой состав интегрированную среду разработки Microsoft Visual Studio, а также оптимизированную математическую библиотеку Compaq Extended Math Library. Несмотря на то,

что это "заслуженная" система программирования, она все еще используется для разработки программ на языке Фортран.

В CVF имеются несколько типов проектов.

- Первым из них является *проект консольного приложения*. Программа в таком проекте использует только текстовый интерфейс. Запускается консольная программа в отдельном окне, а для взаимодействия с пользователем она использует обычные операции ввода-вывода. Консольная программа является наиболее универсальной и переносимой. Предназначена она обычно для численных расчетов.
- *Проект со стандартной графикой* использует одно графическое окно, в которое могут выводиться графические примитивы. Графика поддерживается библиотекой QuickWin. В программе можно использовать диалоговые окна. Данный тип проектов используется обычно для разработки расчетных программ с простым графическим выводом результатов.
- *Проект QuickWin-графики* позволяет работать одновременно с несколькими окнами. В эти окна могут выводиться различные графики, окна могут расширяться на полный экран или занимать часть экрана. Доступны различные возможности графического интерфейса MS Windows.
- *Проект Windows-приложения* позволяет использовать API Windows (*API* — Application Programming Interface, интерфейс программирования приложений — набор функций, который используется программистами для создания приложений с определенной функциональностью) непосредственно из программы на Фортране. Набор возможностей шире, чем у QuickWin-приложений.

Система программирования Sun Studio

Система программирования Sun Studio содержит различные средства разработки и отладки программ для операционных систем Solaris и Linux. Она распространяется бесплатно и содержит

эффективные компиляторы для языков C, C++ и Фортрана. Sun Studio поддерживает разработку многопоточных приложений на основе OpenMP, включает в свой состав анализатор потоков и другие средства отладки. Описание среды и компиляторов можно найти на сайте **www.sun.com**.

Программы-отладчики

Для отладки программ в среде операционной системы UNIX можно использовать интерактивный отладчик dbx. Сам по себе он имеет довольно примитивный интерфейс, а работа с ним требует определенного навыка. Время, потраченное на его изучение, впоследствии вознаграждается экономией времени, которое уходит на поиск ошибок. Существуют графические оболочки для dbx, можно использовать и интерфейс стандартного для ОС UNIX текстового редактора Emacs.

Отладчик dbx может проследить выполнение программы на уровне исходного кода, построчно, с возможностью получения информации о каждой переменной. Можно также проследить "судьбу" отдельно взятой переменной.

Для работы с отладчиком программа должна быть откомпилирована со специальным ключом. В этом случае в исполняемый файл включается специальная информация о символических именах переменных и их размещении в исходном тексте программы (таблица символов).

При компиляции с ключом отладки необходимо отключить любую оптимизацию, поскольку в результате оптимизации порядок операторов может измениться, и место расположения ошибочной конструкции найти не удастся. Вообще говоря, таблица символов включается в исполняемый файл и при компиляции без ключа отладки, но формат этой таблицы может оказаться несовместимым с тем, который распознается отладчиком dbx.

Для вызова отладчика dbx с его собственным интерфейсом используется команда:

```
dbx <файл>
```

и если запуск отладчика прошел нормально, а программа была откомпилирована с ключом отладки, то на экране появится сообщение вида:

```
dbx version 3.1 for AIX.
Type 'help' for help.
reading symbolic information ...
(dbx)
```

Отладчик ищет файл с именем core (так называемый *дамп памяти*), и лишь затем обращается к исходному тексту программы. Файл core должен находиться в текущем каталоге. Если он не найден, то отладчик загружает программу. Имя файла (маршрутное) можно указать в качестве аргумента команды запуска отладчика. Если во время выполнения программы произошел сбой и был создан дамп памяти, с помощью отладчика можно выполнить "посмертный" анализ события.

Команды (см. табл. 1.6) набираются в ответ на приглашение отладчика. После запуска отладчика программа готова к выполнению. Отладчик сначала ищет файл с главной программой и ожидает ввода очередной команды.

Получить список команд можно с помощью команды `help`. Справку по каждой команде отладчика можно получить, указав ее в качестве аргумента команды `help`. Большинство команд допускает сокращенную форму, например, вместо `print` можно использовать `p`. Завершается работа с отладчиком dbx командой `quit`.

Чрезвычайно полезным инструментом отладки являются *точки останова*. Они устанавливается с помощью команды `stop`. Затем программа запускается на выполнение, которое приостанавливается в точке останова. Команда `cont` позволяет продолжить выполнение программы до ее завершения или до следующей точки останова. Список активных точек останова можно получить с помощью команды `status`, а удаляются они командой `delete`.

В команде `stop` можно использовать условные выражения, например:

```
(dbx) stop at 10 if (i == 5)
[1] if i = 5 { stop } at 10
(dbx) run
```

При работе с отладчиком dbx можно менять значения переменных, например:

```
(dbx) stop at 10
[1] stop at 10
(dbx) run
Running: dumpcore
stopped in main at line 10
10                  printf("Goodbye world! (%d)\n", i);
(dbx) assign i = 5
(dbx) next
Goodbye world! (5)
```

Меняя значения переменных, можно исследовать поведение программы при разных условиях.

Кроме отладчика dbx в среде ОС UNIX имеются и другие, например, отладчик adb, но он уступает отладчику dbx в простоте использования.

Таблица 1.6. *Команды отладчика dbx*

Команда	**Описание**
Выполнение программы и ее трассировка	
`Run`	Результатом выполнения этой команды является запуск программы на выполнение. Управление передается программе, и она выполняется так, как если бы отладчика не было. Если программа требует ввода с клавиатуры, то его необходимо осуществить. Выполнение программы продолжается до наступления одного из следующих событий: • достижение точки останова; • одновременное нажатие клавиш <Ctrl>+<C>; • нормальное завершение работы программы; • сбой в работе программы, завершающийся созданием дампа памяти (core-файл). Дамп памяти представляет собой файл, который содержит мгновенный снимок состояния памяти в момент аварийного завершения работы программы

Таблица 1.6 *(продолжение)*

Команда	Описание
Выполнение программы и ее трассировка	
`Cont`	Эта команда используется для продолжения выполнения программы после паузы
`Trace`	Эту команду можно использовать для того, чтобы проследить изменения переменных в процессе выполнения программы. Выполнение программы при этом значительно замедляется
`stop at <номер_строки>`	Отладчик приостанавливает свою работу перед выполнением оператора в указанной строке исходного текста. Эта операция называется "установкой точки останова". Оператор должен быть исполняемым
`stop in <имя_процедуры>`	Отладчик приостанавливает свою работу каждый раз, когда управление передается в процедуру
`Step`	Результат этой команды — выполнение одной строки исходного текста программы
`Next`	Команда пошагового выполнения без входа в подпрограммы (подпрограммы, тем не менее, выполняются)
Вывод и отображение данных	

`print <выражение>`	Команда выводит значение последующего выражения.
	Данная команда используется для вывода значений переменных, например:
	`(dbx) print a`
	`1.2387997473787516e-05`
	`(dbx)`

`whatis <имя>`	Команда выводит описание указанной переменной или типа
`Assign`	Команда присваивает значение переменной

Таблица 1.6 *(продолжение)*

Команда	Описание
Работа с подпрограммами	
`Where`	Команда осуществляет вывод списка подпрограмм, обращения к которым привели в текущую точку выполнения программы, а также номера строки исходного текста. Данная команда особенно полезна при работе с дампом памяти core. После сбоя программы и создания такого файла можно запустить отладчик и выполнить команду `where`. Отладчик сообщит, какой оператор программы выполнялся непосредственно перед сбоем
`call <процедура>`	Команда вызывает указанную подпрограмму
`Dump`	Выводит имена и значения всех локальных переменных
`Return`	С помощью этой команды выполнение программы продолжается до выхода из текущей подпрограммы, после чего оно приостанавливается
Доступ к исходным файлам и каталогам	
`Edit`	Команда вызывает редактор для редактирования текущего исходного файла
`File`	Команда предназначена для изменения текущего исходного файла
`Func`	Команда предназначена для изменения текущей подпрограммы
`list строка`	Команда выводит строки исходного текста. Каждой строке исходного текста присваивается номер. Если диапазон строк не указан, то они выводятся порциями по 10, начиная с самой первой. Можно указать и диапазон номеров строк
`Use`	Команда задания списка каталогов для поиска файлов с исходными текстами
`/.../`	Команда, выполняющая поиск по направлению к концу файла
`?...?`	Команда, выполняющая поиск по направлению к началу файла

Таблица 1.6 *(окончание)*

Команда	Описание
Вспомогательные команды	
`Help`	Команда для вызова справки по командам
`Source`	Команда позволяет вводить команды из внешнего файла
`Quit`	Команда для завершения работы с отладчиком

Для отладки программ в среде операционной системы Linux используется свободно распространяемый интерактивный отладчик gdb. Его интерфейс похож на интерфейс отладчика dbx. С командами отладчика можно ознакомиться с помощью инструкции:

```
info gdb
```

в среде ОС Linux.

Для отладчиков gdb, dbx, wdb, jdb, xdb и некоторых других имеется графическая оболочка ddd.