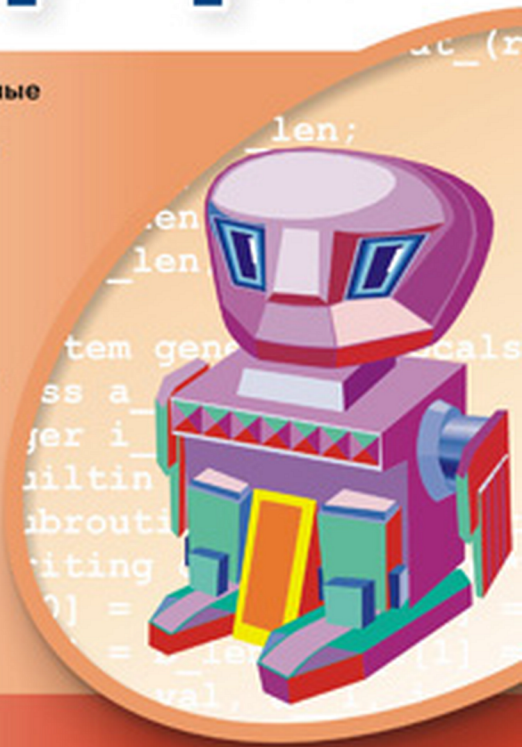
bhv

Сергей Немнюгин
Ольга Стесик

САМОУЧИТЕЛЬ

Современный Фортран

- Синтаксис и основные алгоритмические конструкции языка
- Встроенные математические подпрограммы
- Операторы ввода/вывода и средства работы с файлами
- Компиляция, отладка и оптимизация программ
- Новый стандарт языка



Эффективное средство решения вычислительных задач любой сложности

Сергей Немнюгин
Ольга Стесик

Современный Фортран

САМОУЧИТЕЛЬ

Санкт-Петербург
«БХВ-Петербург»
2004

УДК 681.3.068+800.92фортран
ББК 32.973.26-018.1
Н50

Немнюгин М. А., Стесик О. Л.

Н50 Современный Фортран. Самоучитель. — СПб.: БХВ-Петербург, 2004. — 496 с.: ил.

ISBN 5-94157-302-2

Книга является пособием по изучению языка Фортран. Последовательно излагается синтаксис языка, рассматривается реализация основных алгоритмических конструкций. Особое внимание уделено встроенному математическому аппарату, средствам работы с массивами, операциям ввода/вывода. Изложение следует современным стандартам языка — Фортран 90/95. Приводится описание ряда наиболее распространенных компиляторов языка, вспомогательных средств разработки и отладки программ. Рассматриваются вопросы смешанного программирования на Фортране и C, реализация объектно-ориентированного подхода на Фортране. Дается описание одного из основных средств параллельного программирования — Высокопроизводительного Фортрана. Впервые на русском языке описывается новый стандарт языка — Фортран 2003. В приложениях приведена полезная справочная информация, которая не только поможет в повседневной практической работе, но и станет отправной точкой в дальнейшем знакомстве с одним из наиболее мощных языков вычислительного программирования.

Для программистов

УДК 681.3.068+800.92фортран
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Анатолий Адаменко*
Зав. редакцией	*Григорий Добин*
Редактор	*Леонид Мирошенков*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Виктория Пиотровская*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 25.11.03.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 40.
Тираж 3 000 экз. Заказ №
"БХВ-Петербург", 198005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар № 77.99.02.953.Д.001537.03.02
от 13.03.2002 г. выдано Департаментом ГСЭН Минздрава России.

Отпечатано с готовых диапозитивов
в Академической типографии "Наука" РАН
199034, Санкт-Петербург, 9 линия, 12.

ISBN 5-94157-302-2

Содержание

Памяти Станислава Петровича Меркурьева

Предисловие

Фортран занимает почетное место среди современных языков программирования. Это один из первых языков программирования высокого уровня и с самого своего рождения он предназначался для решения сложных вычислительных задач. В среде прикладных программистов Фортран сначала был встречен скептически, поскольку считалось, что заплатить за удобство программирования на языке высокого уровня придется значительной потерей скорости вычислений. Если речь идет о моделировании сложных процессов или обработке больших объемов информации, скорость вычислений является решающим фактором, определяющим выбор языка, вычислительной платформы и технологии программирования.

Разработчикам Фортрана удалось найти компромисс между удобством программирования и эффективностью программ, написанных на этом языке. Синтаксис языка строился таким образом, чтобы обеспечить максимальную эффективность автоматической оптимизации исполняемого кода. Это позволило в дальнейшем создавать оптимизирующие компиляторы, поставившие вычислительные возможности программ на Фортране вне всякой конкуренции. Язык был оснащен богатым набором встроенных математических функций и функций ввода-вывода, что существенно упрощает процесс программирования вычислительных задач.

Несмотря на свой "почтенный" возраст, Фортран постоянно обновляется. В среднем один раз в 10 лет выходит новый стандарт языка, учитывающий современное состояние программирования с одной стороны и пожелания программистов-прикладников с другой. Один раз в 5 лет выпускается стандарт, который включает относительно небольшие дополнения и изменения. Строгая стандартизация и постоянное обновление позволяют защитить инвестиции в прикладное программное обеспечение и сделать его универсальным. Разработкой стандартов занимается специальный комитет, который собирает и обобщает предложения, а затем выпускает серию проектов стандарта, доступных для всеобщего обсуждения. Трудно назвать какой-либо

другой язык, который сочетал бы постоянное обновление и достаточно строгое следование стандартам.

Темп обновления Фортрана может показаться медленным, однако это избавляет программиста-прикладника от необходимости постоянно знакомиться с новыми версиями языка и дает возможность сосредоточиться на решении задач из своей предметной области. Фортран впитывает те достижения Computer Science, которые действительно необходимы и полезны при программировании вычислений и не сказываются сколько-нибудь заметным образом на их скорости. В этом суть философии Фортрана.

Оптимизирующие компиляторы для Фортрана были включены в список десяти наиболее выдающихся достижений Computer Science XX века, опубликованный IEEE.

Удачная "биография" языка во многом была определена личностью человека, который руководил его разработкой. Это — Джон Бэкус. Джон Бэкус родился 3 декабря 1924 года в США. В 1942 году он окончил школу, но к учебе, по его собственному признанию, относился несерьезно. По окончании школы Джон по совету отца поступил в университет Вирджинии и приступил к изучению химии, но в 1943 году бросил учебу и ушел в армию. В армии Бэкус был направлен на обучение в медицинский госпиталь со специализацией в области нейрохирургии, но учеба не пошла и там. Бэкус считал, что она сводилась к зубрежке, а места размышлениям в ней не было, и через девять месяцев на карьере медика был поставлен крест.

Бэкус переехал в Нью-Йорк. Размышляя о смысле жизни, Джон пришел к неожиданному выводу, что для полного комфорта ему необходим аппарат для высококачественного воспроизведения музыки. Таких аппаратов в то время не было, Бэкус решил взять судьбу в собственные руки и отправился в школу радиотехников. Здесь он встретил первого в своей жизни учителя, который вызвал у него доверие. Учитель попросил Джона помочь ему с расчетом характеристик радиосхем. Сравнительно простой расчет усилителя, в общем нудный и громоздкий, вызвал у Бэкуса интерес к математике.

Бэкус поступил в Колумбийский университет Нью-Йорка и стал изучать математику. Университет Джон закончил в 1949 году. Незадолго до окончания университета он посетил вычислительный центр фирмы IBM, куда вскоре и был принят на работу программистом. Это произошло в 1950 году. Именно в IBM Бэкус возглавил работы по разработке первого языка высокого уровня Фортран (FORTRAN, от FORmula TRANslator — "переводчик формул" на машинный язык). Первая коммерческая версия языка была выпущена в 1957 году.

Вспоминая работу над проектом, Бэкус писал, что разработчики поначалу не знали точно, чего они хотят добиться. Они бурно обсуждали структуру языка и методы синтаксического анализа выражений. Результатом этой работы стал замечательный язык программирования, вскоре завоевавший популярность.

Бэкус известен в истории программирования не только как разработчик Фортрана. В 1959 году он предложил систему обозначений для описания синтаксиса языков программирования высокого уровня. Она называется формой Бэкуса — Наура (BNF).

Таким образом, первую скрипку в процессе работы над Фортраном, языком, предназначенным для программирования вычислений, играл человек с хорошим математическим образованием и опытом численных расчетов в области физики. Как знать, может быть картина современного программирования оказалась бы иной, если бы безвестный учитель математики не попросил Бэкуса помочь ему провести расчеты радиоусилителя!

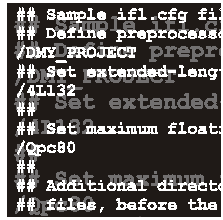
В книге, которую вы, уважаемый читатель, держите в своих руках, содержится описание Фортрана согласно стандартам, которые обычно называют Фортран 90 и Фортран 95. Иногда, и мы постарались это особо оговаривать, речь идет о Фортране 77, который все еще достаточно распространен в среде прикладных программистов и научно-технических работников.

Мы старались избегать ориентации на конкретные компиляторы или среды программирования. Книга содержит примеры-листинги с исходными текстами, иллюстрирующими материал. Мы надеемся, что это облегчит процесс самостоятельного изучения языка. Настоятельно рекомендуем выполнять задания для самостоятельной работы и отвечать на вопросы, которыми завершается большая часть глав книги.

Одна из глав посвящена вопросам взаимодействия между языками Фортран и C/C++. Дается также описание основных особенностей Высокопроизводительного Фортрана и Фортрана 2003. Фортран 2003 — это очередное значительное обновление языка, которое будет официально принято в качестве стандарта в ближайшем будущем.

Мы надеемся, что наш скромный труд принесет пользу программистам, приступающим к изучению Фортрана или использующим его в своей повседневной работе. В последнем случае книгу можно использовать в качестве справочника и сборника полезных примеров. Авторы будут благодарны за замечания и предложения по содержанию книги, которые можно направлять по адресу электронной почты:

club1982@yandex.ru



Глава 1

Элементы языка Фортран

В этой главе мы познакомимся с основными элементами языка Фортран — его алфавитом, специальными словами и символами; рассмотрим структуру программы. Здесь же мы узнаем о встроенных типах Фортрана, важнейших операторах, управляющих ходом выполнения программы, а также об операторах ввода/вывода. Это позволит приступить к написанию собственных, пока еще достаточно простых программ и поможет читать исходные тексты Фортрана. Завершается урок вопросами и упражнениями. В данной главе и далее в книге мы будем сопровождать описание конструкций языка и приемов программирования примерами программ — как относительно простых, так и более сложных. Многие темы данной главы более подробно рассматриваются в следующих главах.

Принципы языков программирования. Договоримся о терминах

В различных языках программирования, несмотря на все различия между ними, иногда довольно значительные, можно найти много общих черт. Можно сказать, что в основу различных языков положены общие принципы, но реализация этих принципов может быть разной. Прежде всего, стандарт (спецификация) любого языка ограничивает набор символов, которые можно использовать в программах. Этот набор называется *алфавитом* языка. Символы алфавита являются элементами более сложных образований — *лексем*. Лексемы играют роль слов — минимальных смысловых элементов языка. Так и в обычном языке буквы алфавита используются для построения слов, каждое из которых несет определенную смысловую нагрузку. Лексемами языка программирования являются:

- имена переменных, подпрограмм и т. д.;
- специальные слова, из которых составляются предложения описания, операторы и другие конструкции языка;
- обозначения операций, например, арифметических;

- буквальные константы;
- разделители, то есть специальные символы, отделяющие друг от друга операторы, элементы списков и т. д.;
- метки.

В каждом языке программирования используются свои правила построения имен. В правилах присвоения имен переменным (и другим объектам) следует обратить внимание на то, какие символы можно использовать в именах, какова максимальная длина имени и различается ли регистр букв.

Специальные слова используются для построения предложений — базовых единиц языка, обладающих смысловой завершенностью. Специальные слова бывают двух видов: зарезервированные и ключевые. *Зарезервированные слова* имеют вполне определенные смысл и назначение, которые определяются спецификацией или стандартом языка. Любая неточность в применении зарезервированных слов является серьезной ошибкой. Назначение *ключевых слов* зависит от контекста, то есть от того, где в программе слово используется.

Любая программа содержит переменные, константы, предложения описания и операторы. *Переменная* связана с областью памяти компьютера, которой присвоено определенное имя и которая наделена определенными свойствами. Иногда говорят, что переменной соответствует одна абстрактная ячейка памяти. Физический размер этой ячейки зависит от типа переменной, то есть от того, какая информация в ней хранится.

Переменная характеризуется набором атрибутов. *Имя переменной* является одним из основных ее атрибутов. Следующий атрибут — *значение переменной*. Содержимое ячейки памяти, связанной с переменной, может изменяться в ходе выполнения программы (собственно, поэтому переменная и называется переменной!). *Тип переменной* определяет вид информации, содержащейся в абстрактной ячейке памяти, а также набор операций и множество ее допустимых значений. Важнейшей характеристикой переменной является и ее *адрес*. В линейной модели памяти это порядковый номер первой физической ячейки памяти, входящей в состав абстрактной ячейки. *Область видимости переменной* — это такая часть программы, в которой операторы могут использовать данную переменную. Обычно говорят о *глобальных* и *локальных* переменных. Глобальные переменные доступны во всей программе, а локальные только в отдельных ее блоках.

Константа отличается от переменной прежде всего тем, что ее значение не изменяется в ходе выполнения программы. Принято различать *буквальные* и *именованные* константы. Буквальные константы представляют собой значения, которые воспринимаются в программе в точности так, как они изображены. Значения могут быть числовыми, символьными или другими. Буквальные константы еще называют литералами.

На именованные константы ссылаются, указывая их имя. Имена назначаются константам обычно по тем же правилам, что и переменным.

Оператор описывает некоторое законченное действие, например, вычисление по математической формуле или последовательность выполнения других операторов программы. Операторы, которые описывают свойства (атрибуты) переменных и других объектов, используемых в программе, называются *предложениями описания* или декларативными операторами. Операторы, управляющие ходом выполнения программы, называются *управляющими операторами*. Управляющие операторы, прежде всего, реализуют основные алгоритмические конструкции, такие как бинарные и многовариантные ветвления (условные операторы, операторы выбора) и циклы. Имеются операторы ввода и вывода информации и т. д.

Программа представляет собой последовательность операторов и других элементов языка, построенную в соответствии с правилами языка и предназначенную для решения определенной задачи. Правила написания программы, определяющие, какие последовательности символов можно использовать в программе, называются *синтаксисом* языка программирования.

Типы языков программирования

Языки программирования используются для записи алгоритмов с целью их последующего выполнения на компьютере. Напомним читателю, что *алгоритм* представляет собой конечный набор действий, расположенных в определенном логическом порядке, позволяющий исполнителю решать любую конкретную задачу из некоторого класса однотипных задач. Исполнителем алгоритма может быть компьютер или другое устройство. Алгоритм для компьютера может быть составлен и записан с разной степенью подробности. Эта степень зависит от того, насколько детально учитывается архитектура компьютера, на котором предполагается выполнять программу. Программист пишет программу для некоторой абстрактной машины. *Абстрактная вычислительная машина* представляет собой упрощенную модель реального компьютера, в которой отсутствуют детали, не имеющие значения для программиста. Чем больше деталей устройства реального компьютера содержится в описании этой машины, тем ниже уровень абстракции. На одном из самых высоких уровней абстракции находится представление о компьютере как машине, состоящей из процессора, памяти и устройств ввода/вывода. Такую архитектуру называют фон-неймановской.

Языки низкого уровня используются для детального описания операций, когда приходится учитывать, например, устройство центрального процессора и других функциональных узлов компьютера. Такими языками являются *машинные коды* и *ассемблеры*. Программа, написанная на ассемблере, получается

подробной и, как правило, достаточно длинной, следовательно, увеличивается вероятность появления ошибок. Для составления программы требуется предварительное изучение архитектуры определенного типа компьютеров, что увеличивает трудоемкость программирования. Программа оказывается привязанной к конкретной архитектуре. Трудоемкость программирования и высокую вероятность ошибок можно считать недостатками программирования на языках низкого уровня. Преимуществом их использования является возможность "выжать" из компьютера все, что можно, прежде всего, максимум быстродействия.

Языки программирования высокого уровня были созданы для того, чтобы преодолеть недостатки низкоуровневого программирования. Они позволяют использовать различные операции, не заботясь о деталях их реализации на компьютере с конкретной архитектурой. Тексты программ при этом оказываются более короткими и универсальными (независимыми от архитектуры), их легче читать, в них проще разобраться, а время их разработки значительно сокращается. Однако объем занимаемой памяти и время выполнения таких программ значительно больше, чем у тех, что написаны на языках низкого уровня.

Языки программирования высокого уровня обычно делят на 4 класса.

1. Императивные (процедурные).
2. Функциональные.
3. Логические.
4. Объектно-ориентированные.

Основными объектами в *императивных* языках являются переменные, операторы присваивания, стандартные алгоритмические конструкции. Императивные языки программирования привязаны к традиционной фоннеймановской архитектуре.

В *функциональных* языках программирования используются функции, значения которых определяются по заданным параметрам. Традиционные переменные, операторы присваивания при этом уже не нужны или, по крайней мере, не обязательны.

В программах, написанных на *логических* языках, нет определенного, фиксированного порядка выполнения правил и шагов алгоритма, а выбор подходящей последовательности возлагается на систему.

Объектно-ориентированные языки упрощают программирование за счет использования технологии объектно-ориентированного программирования (ООП).

Фортран является языком программирования высокого уровня. Он относится к категории императивных языков и изначально создавался для программирования вычислений.

Алфавит, лексемы, специальные слова языка Фортран

Алфавит языка Фортран включает 26 латинских букв:

A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z,

причем регистр букв в программах различается только для строковых констант. Есть также арабские цифры:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9,

символ подчеркивания и специальные символы.

Все специальные слова языка Фортран являются ключевыми. Они могут использоваться в составе предложений или операторов языка, но допускается и произвольное их применение. Последнее, впрочем, следует считать нежелательной практикой и плохим стилем программирования. Его следует избегать.

В табл. 1.1 приведены основные ключевые слова Фортрана.

Таблица 1.1. *Основные ключевые слова языка Фортран*

ADMIT	ALLOCATABLE	ALLOCATE	ASSIGN
ASSIGNMENT	ATEND	BACKSPACE	BLOCKDATA
CALL	CASE	CHARACTER	CLOSE
COMMON	COMPLEX	CONTAINS	CONTINUE
CYCLE	DATA	DEALLOCATE	DEFAULT
DIMENSION	DO	DOUBLE	ELSE
END	ENDFILE	END IF	END SELECT
ENTRY	EQUIVALENCE	EXIT	EXTERNAL
FORMAT	FUNCTION	GO TO	IF
IMPLICIT	INCLUDE	INQUIRE	INTEGER
INTENT	INTERFACE	INTRINSIC	LOGICAL
MAP	MODULE	NAMELIST	NONE
OPEN	OPTIONAL	PARAMETER	PAUSE
POINTER	PRINT	PRECISION	PROCEDURE
PROGRAM	READ	REAL	RECORD
RECURSIVE	RETURN	REWIND	SAVE
STOP	STRUCTURE	SUBROUTINE	TARGET
THEN	TYPE	UNION	USE
WHILE	WRITE		

Лексемами являются обозначения логических операций и логических констант, которые приведены в табл. 1.2. Запись отношений с помощью 4-символьных обозначений характерна для Фортрана 77 и более старых версий языка. Начиная с Фортрана 90, используются более удобные обозначения, приближенные к математической нотации (т. е. правилам записи).

Таблица 1.2. *Логические операции и константы языка Фортран*

Символ	Описание	Символ	Описание
.GT.	Отношение "больше"	.OR.	Логическая операция "ИЛИ" (логическое сложение)
.LT.	Отношение "меньше"	.AND.	Логическая операция "И" (логическое умножение)
.GE.	Отношение "больше или равно"	.NOT.	Логическое отрицание
.LE.	Отношение "меньше или равно"	.FALSE.	Логическая константа "ложь"
.NE.	Отношение "не равно"	.TRUE.	Логическая константа "истина"
.EQ.	Отношение "равно"	.EQV.	Отношение эквивалентности
.NEQV.	Отношение неэквивалентности		

В программах на языке Фортран используются как отдельные специальные символы, так и пары символов, которые имеют специальное значение. Перечень таких символов с описанием некоторых вариантов их применения приведен в табл. 1.3.

Таблица 1.3. *Одиночные и двойные специальные символы языка Фортран*

Символ	Описание	Символ	Описание
=	Оператор присваивания	.	Десятичная точка в буквальных числовых константах или элемент логической операции/константы
+	Арифметическая операция "сложение"	(/ /)	Ограничители в конструкторах массивов
-	Арифметическая операция "вычитание"	' "	Ограничители строковой константы

Таблица 1.3 *(окончание)*

Символ	Описание	Символ	Описание
*	Арифметическая операция "умножение"	:	Разделитель между именем конструкции и первым ее ключевым словом, разделитель при указании диапазона значений индекса, разделитель для ветви ONLY оператора использования USE
/	Арифметическая операция "деление", ограничитель для имени COMMON-блока	!	Начало комментария
( )	Список параметров подпрограммы, индексы массива, циклы		Пробел
,	Разделитель в списках	::	Разделитель в предложениях описания
&	Признак переноса оператора на следующую строку	==	Отношение равенства
;	Разделитель операторов в строке в свободном формате записи программы	=>	Присваивание указателя
//	Объединение (конкатенация) строк	**	Возведение в степень
>	Отношение "больше"	/=	Отношение неравенства
<	Отношение "меньше"	>=	Отношение "больше или равно"
%	Селектор компонента структуры	<=	Отношение "меньше или равно"

Формат записи исходного текста программы

Для записи исходного текста программы на Фортране могут использоваться *фиксированный* и *свободный форматы*. Первый из них характерен для стандарта Фортран 77 и более старых, являясь "наследством" перфокарточной эры программирования, а второй применяется в Фортране 90. Фортран 90 поддерживает также фиксированный формат, что обеспечивает совместимость со старыми стандартами записи.

При записи исходного текста в фиксированном формате строка имеет длину 72 позиции (см. рис. 1.1). Первые пять позиций отведены для меток, а шестая может быть пустой или содержать любой, отличный от пробела символ. В последнем случае строка считается строкой продолжения и при обработке компилятором присоединяется к предыдущей строке программы. Допускается не более 19 строк продолжения одного оператора. Оператор может занимать позиции с 7 по 72, а его длина не может превышать 1320 символов (с учетом пробелов). Пробел в исходном тексте программы игнорируется компилятором и используется только для улучшения читаемости программы.

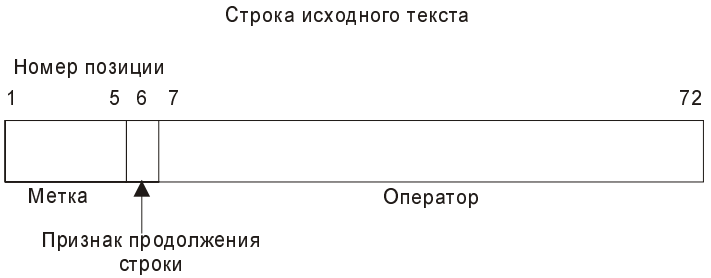


Рис. 1.1. Фиксированный формат записи исходного текста программ на Фортране

В свободном формате записи все позиции строки равноправны, ее длина составляет 132 символа, а максимальная длина оператора — 2640 символов, включая пробелы. При этом пробелы могут быть значимыми символами. Остановимся на правилах их использования в свободном формате.

Пробелы не должны появляться в лексемах, за исключением строковых значений. Например, не допускаются пробелы между звездочками в обозначении операции возведения в степень `**`. Пробелы используются в качестве разделителей между именами, константами или метками и соседними ключевыми словами, именами, константами и метками. В операторах:

```
REAL ELMASS
GO TO 100
DO I=1,10
```

пробелы требуются после слов `REAL`, `TO` и `DO`.

Некоторые ключевые слова обязательно должны отделяться друг от друга пробелами, другие этого не требуют. Например, `BLOCK DATA` можно записать и как `BLOCKDATA`. В табл. 1.4 приводятся ключевые слова, которые требуют использования пробелов и те, в которых пробел не обязателен.

Таблица 1.4. *Использование пробелов в различных конструкциях языка Фортран*

Пробелы не требуются	Пробелы требуются
BLOCK DATA	CASE DEFAULT
DOUBLE COMPLEX	DO WHILE
DOUBLE PRECISION	IMPLICIT спецификация типа
ELSE IF	IMPLICIT NONE
END BLOCK DATA	INTERFACE ASSIGNMENT
END DO	INTERFACE OPERATOR
END FILE	MODULE PROCEDURE
END FORALL	RECURSIVE FUNCTION
END FUNCTION	RECURSIVE SUBROUTINE
END IF	RECURSIVE тип FUNCTION
END INTERFACE	Тип FUNCTION
END MODULE	Тип RECURSIVE FUNCTION
END PROGRAM	
END SELECT	
END SUBROUTINE	
END TYPE	
END WHERE	
GO TO	
IN OUT	
SELECT CASE	

Следующая конструкция, которая может появиться в программе при случайном вводе точки вместо запятой, будет восприниматься как ошибочная:

```
DO 3 I = 1. 3
```

В противном случае (при использовании фиксированного формата) результатом такой опечатки будет синтаксически правильный оператор присваивания значения 1.3 переменной `DO1I`, что является грубой логической ошибкой:

```
DO1I = 1.3
```

Приведем одну из легенд программирования. Считается, что подобная ошибка была допущена в управляющей программе, составленной для бортового компьютера первой американской станции, которую предполагалось отправить к Венере. Компилятор не смог обнаружить ошибку, в результате чего "потеря хвостика запятой" обернулась неудачей дорогостоящего проекта.

Примеры исходных текстов программ в фиксированном и свободном форматах приведены в листингах 1.1 и 1.2.

Листинг 1.1. Пример записи исходного текста программы в фиксированном формате

```
      SUBROUTINE CONVCASE(CIN, COUT, LEN, UPCASE)
      IMPLICIT NONE
      INTEGER :: LEN
      CHARACTER (LEN = *) :: CIN, COUT
      LOGICAL :: UPCASE
!
!  1) CONVERT A CHARACTER STRING TO ALL UPPER CASE OR ALL
!     LOWERCASE.
!
!  2) ARGUMENTS:
!
!     CIN    = INPUT CHARACTER STRING
!     COUT   = OUTPUT CHARACTER STRING IN CONVERTED CASE
!     LEN    = THE CHARACTER LENGTH OF CIN
!     UPCASE = CASE CONVERSION FLAG
!            = .TRUE.  FOR UPPER CASE
!            = .FALSE. FOR LOWER CASE
!
      INTEGER :: I, ILETTR, ILOWR0, IUPPR0
      IUPPR0 = IACHAR('A') - 1
      ILOWR0 = IACHAR('a') - 1
      COUT = ''
      IF(UPCASE) THEN
        DO I = 1, LEN
          ILETTR = IACHAR(CIN(I:I)) - ILOWR0
          IF(ILETTR .GE. 1 .AND. ILETTR .LE. 26) THEN
!           CONVERT TO UPPER CASE.
            COUT(I:I) = ACHAR(IUPPR0 + ILETTR)
          ELSE
            COUT(I:I) = CIN(I:I)
          END IF
        END DO
      ELSE
```

```
        DO I = 1, LEN
          ILETTR = IACHAR(CIN(I:I)) - IUPPR0
          IF     (ILETTR .GE. 1 .AND. ILETTR .LE. 26) THEN
!           CONVERT TO UPPER CASE.
            COUT(I:I) = ACHAR(ILOWR0 + ILETTR)
          ELSE
            COUT(I:I) = CIN(I:I)
          END IF
        END DO
      END IF
      END SUBROUTINE CONVCASE
```

Листинг 1.2. Пример записи исходного текста программы в свободном формате

```
SUBROUTINE CAL_GETENV(var, varval)
! Return as VARVAL the contents of the operating system
! environment variable VAR, if defined.
!
! INPUT:
!    VAR: name of environment variable            [A*]
!
! OUTPUT:
!    VARVAL: contents of enviroment variable
!
 USE MSFLIB
 CHARACTER(LEN = *) VAR, VARVAL
 INTEGER(4) :: lval, lenv
 INTENT(in) :: VAR
 INTENT(out) :: VARVAL
 Lenv = LEN_TRIM(var) !ФОРТРАН 90 intrinsic
 Lval = 0
 IF(lenv .GT. 0) THEN
       lval = GETENVQQ(var(:lenv), varval)
 END IF
 IF(lval .EQ. 0) VarVal = ' '
RETURN
END SUBROUTINE CAL_GETENV
```

Остановимся более подробно на особенностях записи текста в свободном формате. Мы уже знаем, что иногда оператор или другая конструкция

программы не умещается в одной строке. Вообще говоря, стоит избегать слишком длинных конструкций, поскольку они ухудшают читаемость программы. Если все-таки возникла необходимость в переносе строки, то в конце строки, в месте переноса, необходимо поставить знак & (амперсанд), тогда следующая строка считается продолжением данной. Таких строк продолжения может быть несколько. Пример:

```
ISEED = MAX(1 + MOD(INT(TOD), 1000) / 10,&
1 + MOD(INT(TOD), 100))
```

Этот фрагмент эквивалентен следующему:

```
ISEED = MAX(1 + MOD(INT(TOD), 1000) / 10, 1 + MOD(INT(TOD), 100))
```

Если же, напротив, необходимо в одной строке разместить несколько операторов, они отделяются друг от друга символом "точка с запятой" (;), например:

```
A = 2.2; B = 4; C = 7.8
```

Операторы программы следует располагать таким образом, чтобы обозначить и подчеркнуть логику ее работы. Для этого используются пробелы между операторами и их частями, а также отступы — дополнительные пробелы в левой части строки. Пробелы и отступы помогают читателю программы определить уровень подчиненности каждой строки — программные конструкции, находящиеся на верхнем уровне иерархии (например, внешние циклы или условные операторы), набираются с минимальным отступом. Вложенные операторы набираются с отступом. Вложенные операторы следующего уровня набираются с дополнительным отступом и т. д. Компьютерные программы содержат большое количество концентрированной информации, и хорошее форматирование значительно упрощает их чтение. Следует иметь в виду, что на отладку сложной программы может уйти значительное время, иногда больше половины времени ее разработки. Должное внимание к хорошему форматированию, тщательный и обоснованный выбор имен переменных, подпрограмм и других объектов программы, полное документирование программы увеличивает эффективность труда разработчика. При наборе исходного текста не следует размещать в одной строке более одного-двух операторов. Пустые строки можно использовать для выделения логически связанных групп операторов.

Структура программы

Программа на Фортране состоит из *главной программы* и, возможно, некоторого числа *подпрограмм*. Подпрограммы могут быть *функциями* или *процедурами*, внешними, внутренними или модульными. Главную программу и подпрограммы будем называть *программными компонентами*. Различные программные компоненты могут компилироваться раздельно. Это несомненное достоинство языка, которое позволяет вносить изменения в от-

дельные части программы, не выполняя полной перекомпиляции кода. Компиляторы Фортрана обычно тщательно и неторопливо выполняют свою работу — создание исполняемого файла при компиляции большой программы может потребовать значительного времени. В таких случаях большим удобством является то, что после редактирования одной отдельно взятой подпрограммы можно перекомпилировать только ее. Полученный в результате такой "выборочной" компиляции объектный файл используется при окончательной "сборке" программы.

Первым оператором главной программы является ее заголовок `PROGRAM`. За ним следует идентификатор — имя программы:

```
PROGRAM ИМЯ_ПРОГРАММЫ
```

Имя программы обязательно начинается с буквы, затем могут идти буквы, цифры и символы подчеркивания, например:

```
PROGRAM SUMMATION
PROGRAM QUADRATIC_EQUATION_SOLVER
PROGRAM MERCEDES600
```

Максимальная длина имени программы — 31 символ. Это правило распространяется на любые имена в программах на Фортране.

Заголовок программы может быть опущен. Имя программы не должно совпадать с именами переменных или других ее объектов и не связано с именем внешнего файла, содержащего текст программы.

Первым оператором подпрограммы может быть только ее заголовок `FUNCTION` или `SUBROUTINE`. Последней строкой программного компонента должна быть строка с оператором `END`, который является исполняемым оператором. Заключительный оператор главной программы может иметь также следующий вид:

```
END PROGRAM ИМЯ_ПРОГРАММЫ
```

`ИМЯ_ПРОГРАММЫ` является необязательной частью оператора, как, впрочем, и `PROGRAM`.

После заголовка обычно следуют описания переменных, констант, меток, подпрограмм и других объектов, используемых в программе. Эта ее часть называется *разделом описаний*. В простых программах указанный раздел может быть пустым, т. е. не содержать никаких описаний.

Внимание

Простейшая программа на языке Фортран состоит... из единственного оператора `END`! Такую программу можно откомпилировать, компилятор создаст исполняемый файл, только вот единственное действие, выполняемое такой программой, — немедленное завершение работы.

После раздела описаний следует часть, которая выполняет какие-либо действия. Она называется *разделом операторов*. Структура программы изображена на рис. 1.2.

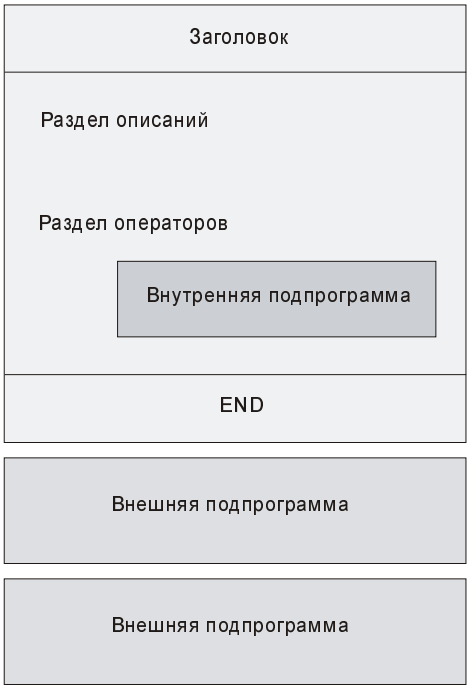


Рис. 1.2. Структура программы на языке Фортран

Раздел описаний состоит из предложений описания переменных, именованных констант, `COMMON`-блоков и некоторых других. Предложений описания переменных может быть несколько, размещаются они между заголовком программного компонента и его разделом операторов. Располагаться предложения описания переменных могут вместе (это — одна из составных частей хорошего стиля программирования), но могут и чередоваться с описаниями других объектов. Предложения описаний должны находиться до предложений `DATA` и исполняемых операторов.

В разделе описаний операторы `IMPLICIT` должны располагаться до всех прочих предложений описания кроме предложений `PARAMETER`. Обычно строка `IMPLICIT` является второй строкой.

Любое предложение описания, которое определяет тип имени именованной константы, должно располагаться до предложения `PARAMETER`. Если в предложении `PARAMETER` используются другие именованные константы, соответствующие им предложения `PARAMETER` должны находиться раньше.

Типы данных

Количество базовых (встроенных или предопределенных) типов Фортрана невелико, и в этом отношении он уступает другим языкам программирования высокого уровня. Есть возможность определять производные типы.

Встроенными являются числовые типы, логический тип и строковый (символьный):

- `INTEGER`;
- `REAL`;
- `COMPLEX`;
- `LOGICAL`;
- `CHARACTER`.

Здесь `INTEGER` — это специальное (ключевое) слово, обозначающее числовой тип, допустимыми значениями которого являются целые числа. Особенностью целого типа является абсолютно точное представление числового значения.

Числовой тип `REAL` представляет вещественные значения. Переменные типа `REAL` не могут принимать значения, сколь угодно близкие к нулю и слишком большие по абсолютной величине. Вещественный тип используется для хранения (приближенных) дробных значений. Представление дробных значений неизбежно является приближенным, что связано с конечностью разрядной сетки компьютера. Арифметические операции с вещественными и комплексными значениями выполняются приближенно. Наличие машинной погрешности следует учитывать при составлении вычислительных программ.

Тип `COMPLEX` соответствует комплексным значениям и представляет собой, фактически, множество упорядоченных пар вещественных значений, первое из которых является вещественной частью комплексного числа, а второе — его мнимой частью. Комплексное значение занимает в памяти в два раза больше места, чем вещественное. Наличие в Фортране комплексного типа — большой подарок программистам-вычислителям, которым достаточно часто приходится пользоваться комплексной арифметикой!

Диапазон допустимых значений числовых типов в Фортране не стандартизирован, но программист может самостоятельно его определить. Механизм подобных определений основан на концепции разновидности типа. У каждого встроенного типа Фортрана есть несколько разновидностей, которые отличаются друг от друга диапазоном значений и физическим размером абстрактной ячейки памяти, соответствующей данному типу. В дальнейшем мы познакомимся со способами определения множеств допустимых значений различных типов.

Множество допустимых значений логического типа `LOGICAL` состоит из двух значений: "истина" и "ложь", которым соответствуют логические константы

.TRUE. и .FALSE. Логический тип и логические операции являются реализацией булевой (двузначной) логики.

Значение символьного типа CHARACTER представляет собой строку символов. Длина строкового значения в Фортране может быть произвольной, она задается с помощью параметра LEN в предложении описания строковой переменной:

```
CHARACTER(LEN = 430) :: Shakespeare_sonet
```

Пробел является значимым символом в символьном значении. Длина символьного значения в байтах равна длине строки. Каждому символу в строке соответствует номер. Номера присваиваются последовательно, начиная с 1 для самого левого символа.

Переменные

При работе с переменными важнейшим является вопрос о том, как сопоставить переменной набор атрибутов и, прежде всего, как определить тип переменной. В Фортране используются два механизма определения типа. Первый из них — *определение типа по имени*. Если первым символом имени переменной или функции являются буквы I, J, K, L, M или N, то по умолчанию предполагается, что переменная имеет тип INTEGER, а если любая другая буква или символ подчеркивания, то подразумевается тип REAL. Этот механизм называется *неявным связыванием имени переменной с ее типом*.

Предложение IMPLICIT позволяет модифицировать (изменить) правила неявного определения типов, а предложение IMPLICIT NONE отменяет действие правила объявления типа по первой букве имени. Отказ от этого, весьма "демократичного" правила гарантирует, что любое имя, отсутствующее в предложениях описания, будет обнаружено компилятором. Таким образом, могут быть исключены ошибки, связанные с появлением "ложных" переменных, образовавшихся в программе в результате опечаток при наборе текста. Авторы настоятельно рекомендуют использовать IMPLICIT NONE!

Второй способ определения типов переменных состоит в использовании предложений описания типов. Этот механизм называют *явным связыванием имен переменных и их типов*.

Предложение описания переменных в Фортране 90 имеет вид:

```
ТИП :: СПИСОК_ПЕРЕМЕННЫХ
```

или

```
ТИП, АТРИБУТЫ :: СПИСОК_ПЕРЕМЕННЫХ
```

В списке переменных имена разделяются запятыми, а ТИП задает общий тип переменных из данного списка, являясь идентификатором типа. Рассмотрим пример описания переменных:

```
INTEGER :: cows, sheeps
REAL, PARAMETER :: salary = 2000
```

Здесь INTEGER, REAL — названия типов, cows и sheeps — имена переменных, а PARAMETER — атрибут объекта salary, определяющий, что он является именованной константой.

В Фортране 90 предложения описания стали более информативными и компактными, чем в Фортране 77, у которого в предложении описания отсутствует разделитель "двойное двоеточие" и нет списка атрибутов, например:

```
REAL PINUMBER, BARRAY
PARAMETER (PINUMBER = 3.14159)
DIMENSION BARRAY(5)
DATA BARRAY /1.0, 2.0, 3.0, 4.0, 5.0/
```

Тот же пример, но переписанный на Фортране 90, умещается в две строки исходного текста:

```
REAL, PARAMETER :: PINUMBER = 3.14159
REAL, DIMENSION(1:3) :: BARRAY = (/1.0, 2.0, 3.0/)
```

В Фортране 90 используются следующие атрибуты объектов:

- PARAMETER — объект является именованной константой;
- PUBLIC — объект является доступным за пределами модуля;
- PRIVATE — объект недоступен за пределами модуля;
- POINTER — объект является ссылкой (указателем);
- TARGET — объект можно использовать в качестве адресата в операторах назначения ссылок;
- ALLOCATABLE — объект является динамическим массивом;
- DIMENSION — объект является массивом;
- INTENT — определяет вид связи для параметра процедуры (т. е. является он входным, выходным или и входным и выходным);
- OPTIONAL — необязательный параметр процедуры;
- SAVE — сохранять значение локальной переменной подпрограммы в промежутке между ее вызовами;
- EXTERNAL — для внешней функции;
- INTRINSIC — для внутренней функции.

Атрибут POINTER нельзя использовать совместно с атрибутами TARGET, INTENT, EXTERNAL и INTRINSIC. Атрибут TARGET нельзя использовать вместе с PARAMETER, а POINTER вместе с ALLOCATABLE. Атрибут TARGET несовместим с атрибутами EXTERNAL и INTRINSIC. Есть и другие ограничения на использование атрибутов. С некоторыми из этих ограничений, а также с назначением различных атрибутов мы познакомимся в дальнейшем.