МИНИСТЕРСТВО ОБЩЕГО И ПРОФЕСИОНАЛЬНОГО ОБРАЗОВАНИЯ РФ

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ИНДУСТРИАЛЬНЫЙ УНИВЕРСИТЕТ

Н.А. Берков
Н.Н. Беркова

УЧЕБНОЕ ПОСОБИЕ

АЛГОРИТМИЧЕСКИЙ ЯЗЫК ФОРТРАН 90

Москва 1998

ББК 32.973-01
УДК 681.3.06

Алгоритмический Язык Фортран 90: *Учебное пособие.* Берков Н. А., Беркова Н.Н.. – М: МГИУ, 1998 г. –96с.

Данное учебное пособие предназначено для студентов МГИУ, изучающих алгоритмический язык ФОРТРАН. Приводится описание основных типов данных и операторов языка Фортран стандарта 1990г. В работе рассмотрено большое число учебных примеров. Особое внимание уделяется графическим функциям. Все приведенные в пособии программы, отлаживались с использованием компилятора Microsoft Fortran 5.10 на ПЭВМ, совместимым с IBM/PC.

Рецензент: Пярнпуу А.А., д.ф-м.н., профессор каф. "Вычислительная математика и программирование" (МГАИ)

Редактор: С.В. Мухин

ЛР № 020407 от 12.02.97

Подписано в печать	Сдано в производство
Формат бумаги 60 x 90/16	Бум. множит.
Усл. печ. л. Уч.-изд. л.	Тем. план 1998 г., № 1-20
Тираж	Заказ

Ротапринт МГИУ , 109280, Москва, Автозаводская, 16

Введение

Алгоритмический язык Фортран занимает особое место среди языков программирования. Появившись на заре вычислительной техники (в 1956г.), он непрерывно развивался и продолжает развиваться, пережив многие более поздние языки. Фортран трижды подвергался стандартизации: в 1966г. – Фортран 66, в 1977г. – Фортран 77 и в 1990г. – Фортран 90. Язык Фортран был разработан для решения научных и инженерных задач и до настоящего времени остается широко распространенным языком среди пользователей, занимающихся вопросами численного моделирования. Это объясняется следующими причинами:

1. Существованием огромных фондов прикладных программ на Фортране, накопленных за предыдущий период.
2. Наличием эффективных трансляторов для различных типов машин. Высокой эффективностью программного кода.
3. Переносимостью на другие типы ЭВМ, работающих в различных операционных системах, что достигается наличием стандартов языка.
4. Простотой конструкций языка.

В данном учебном пособии используется компилятор языка Фортран версии 5.10 фирмы Microsoft, выпущенный в 1991 году, которая практически полностью соответствует стандарту Фортран 90. При этом приводятся только основные, далеко не полные, сведения по операторам, данным и функциям языка Фортран 90. Устаревшие, по мнению авторов, операторы, включенные для поддержания предыдущих стандартов, опущены.

1. Общие сведения

1.1. Алфавит

При записи исходного текста программ используются следующие символы:

1. Прописные и строчные буквы латинского алфавита от A до Z.
2. Арабские цифры от 0 до 9.
3. Специальные символы : ” (“, “) ”, “ + ” , “ - “, “ * ”, “ / ”, “ ‘ ”, “ : :, “ $ ” , “ ! “, точка, запятая и пробел.

Буквы русского алфавита и другие символы могут использоваться только в комментариях и в операторах ввода-вывода.

Примечание. Во всех операторах языка Фортран прописные и строчные буквы считаются одинаковыми, знак пробела игнорируется.

1.2. Синтаксические единицы

Синтаксическими единицами, которые используются при написании операторов (команд), являются константы, ключевые слова, метки, операции и специальные символы.

Константа – это данное, значение которого не изменяется во время выполнения программы.

Имя – это последовательность буквенно-цифровых символов, первым из которых является буква или символ «$». Имена (идентификаторы) служат для обозначения различных элементов программной единицы и должны быть уникальными.

Ключевое слово - последовательность символов, определяющий оператор.

Метка - последовательность не более пяти цифр, одна из которых отлична от нуля. Любой оператор Фортрана может иметь метку, на которую могут ссылаться другие выполняемые операторы.

Операции - определяют действия над данными.

Фортран 90 не требует обязательного разделения синтаксических единиц пробелами. Конец синтаксической единицы определяется из контекста. Для удобочитаемости программы между двумя синтаксическими единицами можно разместить любое количество пробелов. Пробелы могут размещаться и внутри синтаксических единиц, не изменяя их значения (кроме текстовых констант).

Например. «GOTO 123» и «GO TO 1 2 3» - имеет один и тот же смысл.

1.3. Запись программы

Существует два формата записи программы: фиксированный и свободный.

1.3.1. Фиксированная форма записи программы

Текст программы записывается так, что каждая строчка его разделена на 80 позиций. На одной строчке может помещаться только один оператор. Основной текст программы записывается, начиная с 7-ой позиции по 72-ю включительно. Если оператор не помещается на одной строке, то в 6-ой позиции следующей строки (строки продолжения) ставиться любой символ (отличный от пробела) и на ней продолжается запись оператора. Позиции с 73 по 80 могут быть заполнены любыми символами; на действие программы они не оказывают влияния. Позиции с 1 по 5 предназначены для меток операторов. Символ «$» в первой позиции указывает на метакоманду. Звездочка, прописная или строчная латинская С в первой позиции указывает, что данная строка является строкой-комментарием и эту строку компилятор не должен обрабатывать. Комментарий также можно ставить в конце любой строки после символа «!».

1.3.2. Свободная форма записи программы

Если в программе используется метакоманда **$FREEFORM**, то программа вводится в свободном формате; при этом необходимо придерживаться следующих правил ввода программы:

- двойной апостроф в позиции 1 или восклицательный знак указывает на строку комментария;
- операторы и метки могут начинаться с любой (включая первую) позиций;
- для отделения метки оператора от первого символа пробел необязателен;

- если последний символ в строке является знаком минус, то он отбрасывается и следующая строка принимается как строка продолжения. Строка продолжения может начинаться с любой позиции.

Пример.

C Программа в фиксированном формате		
		program primer
8		read *,x !Ввод числа x
		if(x.lt.0) go to 1 2
		print *,‘x больше или = 0.
	1	Введите новое значение.’
		goto 8
12		print *, ‘x <0’,x
		end

$FREEFORM
″ Программа в свободном формате
program primer
8read *,x !Ввод числа x
if(x.lt.0) go to 1 2
print *,‘x больше или = 0. -
Введите новое значение. ’
goto 8
12print *, ‘x <0’,x
end

В примере приведена программа, которая печатает введенное значение переменной x, если оно отрицательное, и просит ввести новое в противном случае. В этой простейшей программе используются две метки 8 и 12, а также символ продолжения в первом операторе печати print.

В этой программе оператор READ вводит значение переменных с клавиатуры в память ЭВМ, а оператор PRINT - наоборот, выводит на экран монитора текст, заключенный в апострофы, и числовое значение переменных из памяти ЭВМ, полученных в ходе работы программы.

1.4. Структура программы

Компилятор языка Фортран обрабатывает программные модули. Программный модуль может быть головной программой, подпрограммой, функцией или блоком данных. Любой из них может быть скомпилирован отдельно, и затем они могут быть скомпонованы вместе. Любой программный модуль начинается с ключевого слова определяющего тип программного модуля: PROGRAM <имя>, SUBROUTINE <имя>, FUNCTION <имя> или BLOCK DATA <имя>(<имя> — это имя программного модуля). Дальше идут операторы программного модуля. Последним оператором является оператор END.

Тело программы состоит из операторов.

Головная программа в качестве первого оператора имеет оператор PROGRAM (его можно опустить).

1.5. Операторы и их классификация

Все операторы делятся на следующие группы: операторы присваивания, операторы управления, операторы ввода-вывода, операторы описания, оператор

DATA, операторы типа программного модуля, операторы отладки и метакоманды.

Операторы присваивания - предназначены для определения значения объектов различных типов.

Операторы управления - позволяют изменять естественный порядок выполнения операторов в программе.

Операторы ввода-вывода - управляют передачей данных между файлами или устройствами ввода-вывода и основной памятью.

Операторы описания - обеспечивают компилятор информацией о свойствах объектов в программном модуле. Эта информация необходима компилятору при распределении памяти и генерации команд.

В программной модуле любого типа должен соблюдаться определенный порядок следования операторов.

Таблица 1.1

<table>
<tr><td colspan="4">$FREEFORM</td></tr>
<tr><td>INCLUDE
Комментрии</td><td colspan="3">PROGRAM,SUBROUTINE, FUNCTION, BLOCK DATA</td></tr>
<tr><td rowspan="5"></td><td rowspan="5">FORMAT</td><td rowspan="3">PARAMETER</td><td>IMPLICIT</td></tr>
<tr><td>COMMON
EQUIVALENCE
EXTERNAL
INTRINSIC</td></tr>
<tr><td>Операторы описания типов переменных</td></tr>
<tr><td rowspan="2">DATA</td><td>Операторные функции
Операторы NAMELIST</td></tr>
<tr><td>Исполняемые операторы</td></tr>
<tr><td colspan="4">END</td></tr>
</table>

Вертикальными линиями в таблице 1.1 разделены группы операторов, которые могут располагаться в любом месте среди операторов группы справа. Например, операторы FORMAT могут располагаться среди операторов IMPLICIT, COMMON и ниже в программном модуле вплоть до оператора END. Оператор DATA может располагаться среди операторных функций, операторов NAMELIST, исполняемых операторов. В любом столбце операторы вышестоящего блока должны предшествовать операторам нижестоящего блока, а внутри блока операторы могут располагаться в произвольном порядке.

2. Основные элементы языка

2.1. Типы данных

Основными элементами языка являются константы, переменные, массивы, структуры, подстроки и выражения. Каждый тип данных в памяти ЭВМ занимает определенную область памяти. В таблице 2.1 представлены простые различные

типы данных и размер занимаемой ими памяти. Сложные структурные элементы будут описаны ниже. Они состоят из ряда простых данных.

Таблица 2.1

Обозначения типа	Длина в байтах	Диапазон изменения
INTEGER*1	1	от -128 до 127 (-2^{-7}до 2^7-1)
INTEGER*2	2	от -32768 до 32767 (-2^{-15} до 2^{15}-1)
INTEGER, INTEGER*4	4	от -2147483684 до 2147483683 (-2^{-31} до 2^{31}-1)
REAL, REAL*4	4	от $-3{,}4*10^{38}$ до $3{,}4*10^{38}$
REAL*8	8	от $-1{,}7*10^{308}$ до $1{,}7*10^{308}$
LOGICAL	4	.TRUE. или FALSE.
CHARACTER	1	Любые символы
CHARACTER*n	n	Любые символы

2.2. Константы

В Фортране применяются арифметические, логические и текстовые константы.

2.2.1. Арифметические константы

Целая константа имеет тип INTEGER*4 и записывается в форме: ±xxx, где xxx - последовательность цифр. У положительной константы знак можно не писать.

Ограничимся пока рассмотрением целых и вещественных (действительных) констант.

Целая константа записывается как целое отрицательное или положительное число или ноль, при этом не используется никакого знака десятичной дроби. Например: 1997 -125 0 1000000.

Часто в программах используются *шестнадцатеричные константы*. При этом перед константой ставят символ #. Например: #1a , #FFFF, #111, #abcdef. В десятичном формате первая константа равна 26, т.к. #1a = 1*16 + 10, #FFFF = $15*16^3 + 15*16^2 + 15*16^1 + 15*16^0$=65535б, #111=$1*16^2 + 1*16^1 + 15*16^0$ =275, #abcdef = $10*16^5 + 11*16^4 + 12*16^3 + 13*16^2 + 14*16^1 + 15*16^0$=11259375.

Вещественные (действительные) константы соответствуют рациональным числам и записываются в той же форме, только вместо десятичной запятой, отделяющей целую часть числа от дробной, в Фортране используется точка. Она обязательно ставится даже в том случае, если дробная часть равна нулю. Знак + можно опускать; можно опускать также целую часть, если она равна нулю.

Наряду с простейшей (обычной) формой, используют также степенную форму вида: $\pm\alpha.\beta e\pm\gamma$, понимая под этим число $\pm\alpha,\beta*10^{\pm\gamma}$,

При этом, если символ e (или E) следует за точкой, точку обычно опускают. Число γ - целая константа.

Примеры:

Число	*Запись на Фортране*	
	Простейшая форма	*Степенная форма*
12,3	+12.3 или 12.3	+1.23e +1 или 12.3e0 или 12.3e1 или 123e-1
-75	-75.	-.75e2 или -75e0
-0,00358	-.00358	-.358e-2 или -3.58e-3
1979	1979.	1.979e3

Замечание. Целые и вещественные константы, представляющие одно и то же число, не взаимозаменяемы, т.к. их представление в машине и действия над ними различны, т.е. записи, например 2 и 2, имеют, вообще говоря, различный смысл. Более подробно об этом написано ниже в разделе 5.4.2. IEEE стандарт хранения чисел в ЭВМ.

Вещественные (действительные) константы двойной точности записываются так же, как и вещественные константы одинарной точности, только вместо символа e или E кодируется d или D $\pm\alpha.\beta d\pm\gamma$. В вещественной константе с одинарной точностью удерживается 7 значащих цифр, а в константе с двойной точностью 17 символов.

2.2.2. Логические константы

Существует всего две логические константы .TRUE. и .FALSE., что означает истина или ложь. При определении начальных значений переменных или массивов логического типа можно использовать краткую форму записи логических констант: F и T (без точек).

2.2.3. Текстовые константы

Текстовая константа записывается в виде непустой последовательности символов, заключенных в апострофы. Значением текстовой константы является последовательность символов между ограничивающими апострофами. Апостроф внутри текстовой константы представляется двумя апострофами, не разделенными пробелами. Каждый символ, включая пробел, занимает один байт. Длина текстовой константы равна количеству символов между ограничивающими апострофами с учетом того, что пара апострофов рассматривается как один символ.

Примеры.

‘Фортран 90’; ‘об’’ем’; ‘integer*4 x’.

2.3. Переменные

Переменная - это поименованное данное, которому отведено определенное место в памяти ЭВМ, к которому можно обратиться по имени.

Идентификатор - это имя (название) переменной величины. Им служит набор букв и цифр, начинающийся с буквы. При этом пробелы игнорируются, то есть учитываются лишь значащие символы. Например, идентификаторы ALFA1 и AL FA 1 считаются одинаковыми. Количество символов в идентификаторе не должно превышать 31 символ.

Переменные величины также могут быть различных типов (Табл. 2.1).

2.3.1. Соглашение по умолчанию для описания типов переменных

В Фортране используется *соглашение по умолчанию*:
если тип переменной не описан явно или неявно, то переменные имена которых начинаются с одной из шести букв I, J, K, L, M, N (**например**: IGREK5, LA, M, K2N3AB и т.д.) являются величинами целого типа, а с любой другой буквы (AL, X, DELTA и т.д.) - величинами вещественного типа.

2.3.2. Операторы описания типов переменных

Оператор IMPLICIT неявно определяет тип переменной по первой букве идентификатора переменной. Рассмотрим этот оператор на примерах.

IMPLICIT REAL*8 (A, D-H, X), CHARACTER*5 (C, Z), INTEGER*2 (I-K), LOGICAL(L)

При использовании этого оператора в программном модуле все неописанные явно переменные, начинающиеся на буквы A, D, E, F, G, H и X, являются вещественными переменными с двойной точностью занимающими в памяти 8 байт. Переменные, идентификаторы которых начинаются на букву C или Z и не описаны явно, являются символьными переменными занимающими 5 байт (5 символов). Переменные, идентификаторы которых начинаются на буквы I, J и K и не описаны явно, являются целыми переменными занимающими 2 байта. Переменные, идентификаторы которых начинаются на букву L и не описаны явно, являются целыми логического типа. Переменные, идентификаторы которых начинаются на буквы M или N и не описаны явно, являются целыми переменными, занимающими 4 байта, т.к. они описаны согласно соглашению по умолчанию. По той же причине переменные, идентификаторы которых начинаются на неуказанные выше буквы и не описаны явно, являются вещественными переменными, занимающими 4 байта.

Операторы явного описание типов переменных описывают тип переменных по именам, перечисленным в конкретном операторе. В этих операторах можно задавать начальное значение переменным, которые присваиваются на этапе компиляции.

Пример.

```
REAL HAMMA, BETTA, X1/0./,PI/3.14159/,I1/25./,L
REAL*8 H/0.66666666666666667/
CHARACTER*10 C/’Фортран 90’/
```

В этом примере явно описаны переменные HAMMA, BETTA, X1, PI, I1 и L как переменные действительного типа. При этом переменным X1, PI и I1 присваиваются начальные значения 0., 3.14159 и 25., соответственно. Переменная H описывается как переменная с двойной точностью и ей присваивается значение 2.d0/3.d0. Переменная C описывается как символьная переменная, под нее отводится 10 байт памяти и присваивается значение Фортран 90.

Оператор PARAMETER присваивает константе идентификатор, который в последующем можно использовать в программном модуле многократно. Опи-

санный идентификатор при этом можно применять во всех конструкциях как обычную константу.

2.4. Операции и выражения

2.4.1. Операции

2.4.1.1. Арифметические операции и выражения

Арифметические выражения образуются из арифметических операндов и арифметических операций. Арифметическими операндами являются константы, имена констант, переменные, элементы массивов, обращения к функциям, а также выражения, заключенные в скобки. Существует пять основных бинарных (операций над двумя операндами) арифметических операций. Для их обозначения используются следующие символы:

1) сложение - +; 2) вычитание - -; 3) умножение - *; 4) деление - /;

5) возведение в степень - **.

В термин "арифметическое выражение" в Фортране вкладывается тот же смысл, что и при обычной математической записи, но при этом можно использовать лишь символы Фортрана, и формулы записываются в одну строку. Знак умножения “*” опускать нельзя. Порядок выполнения операций - обычный. Вместо математических переменных при этом вводятся идентификаторы переменных Фортрана.

Следует отметить, что результатом выполнения первых четырех арифметических операций будет число старшего типа, причем остаток не округляется, а отбрасывается. Например:

2/3 равно 0; 2./3 или 2/3. или 2./3. = 0,6666667.

2.d0/3 = 2/3.d0 = 2.d0/3.d0 = 0,66666666666666667.

Примечание. Основная ошибка начинающего программиста — это применение операции деления двух целых чисел. Следует запомнить, что 1/100 = 2/100 = ... = 99/100 = 0; 100/100 = 101/100 = ... = 199/100 = 1 и т.д.

Для пятой арифметической операции также следует помнить следующие правила:

При возведении в степень x^y показатель степени y может быть либо целого типа, либо вещественного. В первом случае возведение в степень осуществляется по формуле: $x^y = x \cdot x \cdot x \cdot \ldots \cdot x$.

При показателе вещественного типа вычисление производится по формуле $x^y = e^{y \ln x}$, что возможно лишь для " x > 0".

Приведем несколько примеров преобразования арифметических выражений в записи на Фортране:

1) $x + y^{\alpha} - \frac{ab}{cd}$; x+y**alpha-a*b/(c*d)

2) $\frac{x^2 + y}{z * \varepsilon} \cdot z$; (x**2+y)/(z*epsilon)*z

2.4.1.2. Логические операции выражения

Логические выражения составляются из логических операндов и логических операций. Логическими операндами являются логические константы, имена логических констант, логические переменные, элементы логических массивов, обращения к логическим функциям, логические отношения, а также логические выражения, заключенные в скобки. Существует пять основных бинарных (операций над двумя операндами) арифметических операций:

.NOT. логическое отрицание (¬),
.AND. логическое умножение (конъюнкция) (∧),
.OR. логическое сложение (дизъюнкция) (∨),
.EQV. логическая эквивалентность (≡),
.NEQV. логическая неэквивалентность (≢).

Логическая операция .NOT. — унарная операция (т.е. операция над одним операндом), а все остальные бинарные. Результат действия логических операций (таблица истинности) представлена ниже.

Таблица 2.2

Значение операнда		Значение логического выражения				
A	B	.NOT.A	A.AND.B	A.OR.B	A.EQV.B	A.NEQV.B
.TRUE.	.TRUE.	.FALSE	.TRUE.	.TRUE.	.TRUE.	.FALSE.
.TRUE.	.FALSE.	.FALSE	.FALSE.	.TRUE.	.FALSE.	.TRUE.
.FALSE.	.TRUE.	.TRUE.	.FALSE.	.TRUE.	.FALSE.	.TRUE.
.FALSE.	.FALSE.	.TRUE.	.FALSE.	.FALSE	.TRUE.	.FALSE.

Логическим отношением называется выражение, состоящее из двух простых арифметических выражений, связанных одним из шести символов <, ≤, >, ≥, =, ≠, определяющих операции отношения. Т.о., логическое отношение имеет вид A$\Re$B , где A и B - арифметические операнды, а $\Re$ - любая из шести операций отношения. В Фортране для обозначения операций отношения используются следующие комбинации букв с окаймляющими с обеих сторон точками:
.LT. — < ; .LE. — ≤; .GT. — >; .GE. — ≥ ; .EQ. — = ; .NE. — ≠.

Приоритет логических операций с учетом скобок:

1) обращение к функциям (высший приоритет); 2) арифметические и текстовые операции; 3) операции отношения; 4) .NOT.; 5) .AND.; 6) .OR.; 7) .EQV. и .NEQV. одинаковый (низший приоритет).

Примеры.

В программах на Фортране довольно часто используются логические выражения, которые принимают значения истина, если математические переменные x, y, z и т.д. заключены в какой-то диапазон:

1) x∈[0,3].

На Фортране это логическое выражение записывается: x.gt.0.and.x.le.3

$$2)\begin{cases} x > 0 \\ y \le x - y \\ z > x^2 + y^2 \\ x + y + z > 1. \end{cases}$$

На Фортране это логическое выражение записывается:

x.gt.0.and.y.le.x-y.and.z.gt.x**2+y**2.and.x+y+z.gt.1

3) $x \in (-\infty,1] \cup (1,2] \cup (10, \infty)$.

На Фортране это логическое выражение записывается:

(x.le.0).or.(x.gt.1.and.x.le.2).or.(x.gt.10)

Скобки в данном логическом выражении необязательны, т.к. приоритет .and. выше, чем .or. . Здесь скобки используются только для наглядности.

2.4.1.3. Текстовые выражения

Существует всего лишь одна текстовая операция, называемая конкатенацией или сцеплением текстовых строк. Эта операция записывается в виде:

A//B,

где A и B - текстовые операнды, которые могут быть константой, именем константы, переменной, элементом массива, подстрокой, обращением к функции или выражением, заключенным в скобки.

// - операция конкатенации или сцепления.

Значением текстового выражения является строка символов, полученная последовательным дописыванием к значению операнда A значения операнда B. Длина результата равна сумме длин значений операндов. Например: значением текстового выражения ‘Петров’//’ Иван’//’ Леонидович’ будет строка ‘Петров Иван Леонидович’ длиной 22 байта.

2.5. Встроенные функции

В Фортране существует стандартный набор (библиотека) встроенных функций. Встроенные функции составляют часть языка Фортран. Компилятор распознает их имена и автоматически вставляет их объектный код в объектной программы, т.е. в программу, получающуюся после компиляции на машинном языке. Все встроенные функции разобьем на 6 типов: функции преобразования типов данных, функции округления данных, математические функции, символьные функции, битовые функции и встроенные подпрограммы. В данной работе приводятся только основные (по мнению авторов) функции.

2.5.1. Функции преобразования типов данных

В таблицах данного подраздела приведены следующие сокращения типов переменных:

- int - любой целый тип (Integer*1, Integer*2, Integer*4);
- real - любой вещественный тип (Real*4, Real*8);
- gen - int или real;
- char - Character*n;
- dbl - Real*8;
- log - Logical.

- Выражения, записанные в квадратных скобках, необязательны.

Таблица 2.3

Имя функции	Тип аргумента	тип функции	Назначение
int(gen) int1,int2,int4	int,real	int	Преобразует аргумент произвольного типа к целому типу int.
Real(gen)	int, real	Real*4	Преобразует аргумент произвольного типа к вещественному типу.
Float(int)	int	Real*4	Преобразует аргумент любого целого типа к вещественному типу.
Dble(gen)	int,real	Real*8	Преобразует аргумент любого типа к вещественному типу с двойной точностью.
Ichar(char)	char	int	Преобразует аргумент символьного типа в целое значение. Аргумент должен быть одним символом, а его значение есть код этого символа от 0 до 255.
Char(int)	int	char	Обратная функция к функции Ichar.

2.5.2. Функции округления данных

Имя Функции	Тип аргумента	тип функции	Назначение
Aint(real)	real	real	Усечение аргумента. Т.е. дробная часть отбрасывается.
Dint(dbl)	Real*8	Real*8	Усечение аргумента.
Anint(real)	real	real	Округление аргумента по правилам математики.
Dnint(dbl)	Real*8	Real*8	Округление аргумента.
Nint(real)	real	int	Округление аргумента.
Inint(dbl)	Real*8	int	Округление аргумента.

2.5.3. Математические функции

Имя Функции	Тип аргумента	Тип функции	Назначение
Abs(gen)	int, real	Совпадает с аргументом	Вычисление абсолютного значения произвольного аргумента.
Iabs(int)	int	int	Вычисление абсолютного значения аргумента целого типа.
Mod(genA, genB)	int или real	Совпадает с аргументом	Возвращает остаток от деления A на B. Mod(11,2) равно 1.Mod(23,5) равно 3.
Max(genaA, genB, [genC] ...)	int или real	Совпадает с аргументом	Возвращает максимальное значение, введенных аргументов. Число аргументов должно быть ≥ 2.
Min(genaA, genB,[genC] ...)	int или real	Совпадает с аргументом	Возвращает минимальное значение введенных аргументов. Число аргументов должно быть ≥ 2.
Sqrt(gen)	gen	real*8	Вычисляет корень квадратный аргумента, который обязан быть неотрицательным.
Exp(real)	gen	real*8	Вычисляет экспоненту e^x.
Log(gen)	gen	real*8	Вычисляет натуральный логарифм
Log10(gen)	gen	real*8	Вычисляет десятичный логарифм
Sin(gen)	gen	real*8	Вычисляет синус. Аргумент в радианах.
Cos(gen)	gen	real*8	Вычисляет косинус. Аргумент в радианах.
Tan(gen)	gen	real*8	Вычисляет тангенс. Аргумент в радианах.
Cotan(gen)	gen	real*8	Вычисляет котангенс. Аргумент в радианах.
Asin(gen)	gen	real*8	Вычисляет арксинус.
Acos(gen)	gen	real*8	Вычисляет арккосинус.
Atan(gen)	gen	real*8	Вычисляет арктангенс.

2.5.4. Символьные функции

Имя Функции	Тип аргумента	Тип функции	Назначение
Lgt(charA, charB)	char	log	Сравниваются два символа на больше.
Lge(charA, charB)	char	log	Сравниваются два символа на больше или равно.
Lle(charA,	char	log	Сравниваются два символа на меньше.

charB)			
Lle(charA, charB)	char	log	Сравниваются два символа на меньше или равно.
Len(char)	char	int	Определяет длину символьной строки.
Idex(charA, charB, [gen])	char	int	Возвращает номер позиции строки charB в charB. Если введен третий параметр и он =.True., то поиск осуществляется с конца строки.
Len_Trim(char)	char	int	Определяет длину символьной строки без хвостовых пробелов.
Scan(charA, charB,[gen])	char, log	int	Возвращает номер позиции символа charB в строке charB. Если введен параметр log и он =.True., то поиск осуществляется с конца строки.
Verify(charA, charB, [gen])	char, log	int	Возвращает номер позиции символа, не содержащегося в charB в строке charB. Если введен параметр log и он =.True., то поиск осуществляется с конца строки.

2.5.5. Битовые функции

Имя Функции	Тип аргумента	Тип функции	Назначение
IAND(intA, IntB)	int	Совпадает с аргументом	Возвращает результат логического побитового умножения аргументов.
IOR(intA, IntB)			Возвращает результат логического побитового сложения аргументов.
NOT(intA)			Возвращает результат логического побитового дополнения аргумента.
IEOR(intA, IntB)			Возвращает результат логического побитового исключения аргументов.
ISHL(intA, IntB)			Возвращает результат логического побитового сдвига аргумента A на B бит влево, если B>0, и вправо, если B<0.

Пример. Пусть A=240, B=90, а C результат логических операций над A и B.

	Ior(A,B)	Ieor(A,B)	Iand(A,B)	Ishl(A,2)	Ishl(A,-2)
A(битовое)	11110000	11110000	11110000	11110000	11110000
B(битовое)	01011010	01011010	01011010	01011010	01011010
C(битовое)	11111010	10101010	01010000	1111000000	111100
C(десятичн)	250	170	80	960	60

2.5.6. Встроенные подпрограммы

Подпрограммы, в отличие от функций, возвращают несколько значений. Вызов подпрограмм осуществляется определенным оператором CALL, общий вид которого:

CALL имя_п(арг1, арг2, ..., аргn)

где имя_п - имя подпрограммы;

арг1, арг2, ..., аргn - входные и выходные параметры.

После выполнения подпрограммы происходит вычисление выходных параметров.

CALL GETTIM(ihr, imin, isec, i100th)

Определяет показания системных часов. Все параметры являются выходными, и все они имеют тип Integer*2 и обозначают часы (ihr), минуты (imin), секунды (isec) и сотые доли секунды (i10th).

Функция SETTIM(ihr, imin, isec, i100th)

Устанавливает значения системных часов. Все параметры являются входными и обозначают те же значения, что и в подпрограмме GETTIM.

CALL GETDAT(iyr, imon, iday)

Определяет показания системной даты. Все параметры являются выходными и все они имеют тип Integer*2 и обозначают год (iyr), номер месяца (imon) и номер дня в месяце (iday).

Функция SETDAT(iyr, imon, iday)

Устанавливает значения системной даты. Все параметры являются входными и обозначают те же значения, что и в подпрограмме GETDAT.

Функции SETTIM и SETDAT возвращают логическое значение .True., если значения установлены и .False.- в противном случае.

Функция Nargs возвращает число параметров в командной строке при вызове программы на выполнение, включая имя программы. Вызов ее

number_arg=Nargs()

Подпрограмма GETARG возвращает значения указанного параметра. Вызов ее производится оператором

CALL GETARG(n, name, n_simb)

n - номер параметра (от 0 до Narg());

name - возвращаемое значение n-го параметра. Эта символьная переменная должна быть предварительно описана.

n_simb - количество символов в параметре.

Часто, особенно в учебных программах, необходимо вводить большой объем чисел. При этом удобнее применять процедуры обработки псевдослучайных чисел.

Процедура RANDOM возвращает вещественное псевдослучайное число, равномерно распределенное на отрезке [0,1].

Процедура SEED инициализирует генератор псевдослучайных чисел.

Синтаксис:

CALL RANDOM(rand_val)

CALL SEED(seed_val)

3. Основные выполняемые операторы

Для совместности Фортрана 90 с предыдущими версиями оставлены все операторы Фортрана 77 и Фортрана 66. Некоторые из них устарели, поэтому не включены в данную работы, а некоторые исключены ввиду ограниченности объема работы. Кроме того, все выполняемые операторы мы разделим на две группы: основные, т.е. операторы, без которых нельзя писать даже учебные программы, и дополнительные, без которых можно обойтись в простых программах. В данном разделе рассмотрим только основные операторы.

3.1. Операторы присваивания

Операторы присваивания используются для определения значений переменной любого типа элементу массива или подстроки. Это один из наиболее часто используемых операторов в инженерных программах. Существует арифметический, логический и текстовый операторы присваивания.

Общая форма оператора присваивания:

A=E

где *A* - идентификатор переменной, элемент массива или подстрока;

E - арифметическое, логическое или текстовое выражение.

Выполнение оператора состоит из трех этапов:

Вычисляется правая часть, т.е. получается значение арифметического, логического или текстового типа.

Если тип полученного значения отличен от типа идентификатора *A*, то производится перевод полученного значения в тип идентификатора *A*.

Окончательно полученное значение заносится в ячейки памяти, отведенные компилятором для идентификатора *A*.

Для логического оператора присваивания шаг 2) опускается, т.к. правая часть *E* обязана принимать логическое значение .TRUE. или .FALSE.

Для текстового оператора присваивания шаг 2) выполняется следующим образом:

если длина правой части меньше длины идентификатора *A*, то значение выражения *E* дополняется справа пробелами, а если длина выражения *E* превышает длину объекта *A,* то оно усекается справа до длины объекта *A*.

Примеры.

Оператор	Действие оператора
integer x x=34.6/5	Значение переменной x присваивается 6. 1) Правая часть равна 6.92. 2) Переводится в тип идентификатора x, т.е. в целый тип. При этом дробная часть отбрасывается. 3) В ячейки памяти отведенные под компилятором под переменную целого типа x заносится целое число 6.

Real *8 a/3.d0/ a=a+2.d0/3+1./3+2/3	Переменной с двойной точностью a присваивается на этапе компиляции начальное значение 3. Вычисляется правая часть в операторе присваивания, т.е. число с двойной точностью 3 + 0.66666666666666667 + 0.3333333 + 0 = 3.99999996666666667. Полученное число заносится в ячейки, отведенные под переменную a.
Logical L1 read *,x L1=.not.(x.gt.1.and.x.lt.2) print *,L1	Вводится действительная переменная x (read *,x). Вычисляется значение логического выражения L1. Выводится F, если $x \in (1,2)$, или T, если $x \notin (1,2)$.
Character W* 3,x*8,y*12 W=’Abc’ x=’ssssttt’ y=x//’.’//W	Переменной y присваивается текстовое значение ssssttt.Ab. Конкатенация трех символьных цепочек. Переменной y присваивается текстовое значение ssssttt.Abc.

Используя оператор присваивания и простейшие операторы ввода-вывода можно писать простейшие законченные программы.

Пример. Написать программу, которая переводит радианную меру угла в угловую.

```
$FREEFORM
program radgrad
print *,’ Программа переводит радианы в градусы’
print *,’ Введите значение угла в радианах’
read *,x
x=x/3.14159*180.
print *,’ Значение угла в градусах ’,x
end
```

3.2. Операторы управления

Операторы управления предназначены для изменения последовательности выполнения операторов в программном модуле.

3.2.1. Оператор безусловного перехода

Общая форма:

GO TO m

где m - метка оператора данного программного модуля.

Это самый простой оператор. В результате выполнения оператора безусловного перехода происходит передача управления оператору помеченного меткой m, который может быть расположен как выше, так и ниже данного оператора безусловного перехода.

Примечание. Применение операторов безусловного перехода затрудняет чтение программы, а также усложняет процесс оптимизации компилятором объектного кода, поэтому его необходимо применять только в редких случаях. Существует байка: уровень программиста обратно пропорционален количеству операторов безусловного перехода. Этот оператор программисты обязаны были использовать при написании программ на Фортране 66. На самом деле во всех программах, написанных на языке Фортран 90, спокойно можно обойтись без применения этого оператора. Только в том случае, когда в зависимости от каких-то условий необходимо перейти в другой участок программы значительно ниже или выше данного оператора, можно использовать этот оператор. При выходе из внутреннего цикла также необходимо применять этот оператор. В учебных программах, приведенных в данной работе, он не используется. Разработчики нового современного языка JAVA полностью отказались от этого оператора как от "вредного".

3.2.2. Логические операторы if

Операторы перехода **if** предназначены для передачи управления операторам программного модуля в зависимости от выполнения некоторого логического условия.

3.2.2.1. Простой if

Общая форма:

IF(L) S

где L — логическое выражение; S — любой выполняемый оператор, отличный от блочного оператора **if**, оператора **DO** или другого логического оператора **if**.

Оператор работает следующим образом:

1)Вычисляется логическое условие L.

2) Если L =.true., то выполняется оператор S. Затем выполнение программы продолжается со следующего оператора, если только оператором S не является оператор GO TO. Если значение выражение L =.false., оператор S не выполняется, а выполняется оператор, следующий за оператором **if**.

Пример. Вводится номер дня недели. Необходимо напечатать его название. Если день 6 или 7, то напечатать еще и слово (ВЫХОДНОЙ).

```
$FREEFORM
program nameday
character*24 name_day
print *,' Введите номер дня недели'
read *, number_day
if(number_day.eq.1) name_day='Понедельник'
if(number_day.eq.2) name_day=Вторник'
if(number_day.eq.3) name_day='Среда'
if(number_day.eq.4) name_day='Четверг'
if(number_day.eq.5) name_day='Пятница'
if(number_day.eq.6) name_day='Суббота'
if(number_day.eq.7) name_day='Воскресенье'
```

```
if(number_day.eq.6.or.number_day.eq.7) name_day=-
            name_day(1:Len_Trim(name_day))//' (ВЫХОДНОЙ).'
print *,name_day
end
```

Примечание. Данную задачу лучше решать с использованием оператора SELECT, описанного ниже.

3.2.2.2. Блочные операторы if

Блочные операторы **if** предназначены для написания структурированных программ. При их использовании сокращается количество помеченных операторов и количество операторов GO TO, что приводит к более простым программам.

3.2.2.3. Укороченный блочный if

Общий вид:

IF(L) THEN
S
END IF
	где L - логическое выражение; S - блок операторов. Оператор работает следующим образом: Вычисляется логическое выражение L. Если оно истинно, то выполняется блок операторов S, после чего управление передается оператору следующему за END IF.

IF(L1) THEN
S1
S2
IF(L2) THEN
S3
IF(L3) THEN
S4
S5
END IF
S6
END IF! end if(L2)
S7
END IF ! end if(L1)
	Блочные операторы if могут быть вложенными друг в друга. В сложных программах количество вложений может быть достаточно большим, и бывает трудно разобраться, какой оператор END IF закрывает данный блок IF. В таких программах следует применять выравнивание всех операторов, относящихся к одному блоку. А также можно ставить комментарии на некоторые отдаленные END IF. При проверке таких участков программ следует придерживаться правил: любой оператор END IF относится к ближайшему, находящемуся выше незакрытому оператору IF. Количество блочных IF должно соответствовать количеств END IF. Не допускается пересечение блоков IF.

Пример. Вводятся три числа *a, b* и *c,* являющиеся коэффициентами квадратного уравнения. Найти его корни. Предусмотреть все возможные комбинации решений.

```
$freeform
program root2
Parameter (Dnull=0.1d-15)  ! Вводится константа, определяющая машинный нуль
real*8 a,b,c,x1,x2,d
print *,' Введите коэффициенты квадратного уравнения ,a,b,c'
read *,a,b,c
* Дискриминант уравнения
d=b**2-4.d0*a*c
if(d.ge.0.d0.and.abs(a).gt.Dnull) then ! Два корня
```

```
    d=sqrt(d)
    x1=(-b+d)/(2.d0*a)
    x2=(-b-d)/(2.d0*a)
    if(abs(d).ge.Dnull)print *,' x1= ',x1,'  x2=',x2
    if(abs(d).lt.Dnull)print *,' x= ',x1
end if
if(abs(a).le. Dnull) then ! Линейное уравнение, т.к. a=0
    if(abs(b).gt.0. Dnull) print *,'x=',-c/b
    if(abs(b).le. Dnull.and. abs(c).le. Dnull) -
    print *,' Любое x — решение'
    if(abs(b).le. Dnull.and. abs(c).gt. Dnull.) -
    print *,' Нет решений'
end if
if(d.lt.0.) print *,' Дискриминант < 0. Нет решений'
end
```

3.2.2.4. Блочный if

Общий вид:

IF(L) THEN
S
ELSE
S1
END IF
	где L - логическое выражение; S, S1 - блоки операторов.
	Оператор работает следующим образом:
	Вычисляется логическое выражение L. Если оно истинно, то выполняется блок операторов S, после чего управление передается оператору следующему за END IF. Если L - ложно, то выполняется блок операторов S1, после чего управление передается оператору, следующему за END IF.

Рассмотрим решение предыдущего примера с использованием блочного IF.

```
$freeform
program root2
Parameter (Dnull=0.1d-15)
real*8 a,b,c,x1,x2,d
print *,' Введите коэффициенты квадратного уравнения ,a,b,c'
read *,a,b,c
if(abs(a).gt. Dnull) then ! Квадратное уравнение
    d=b**2-4.d0*a*c
    if(d.ge.0.d0) then ! Дискриминант неотрицательный
        if(abs(d).lt.Dnull) then ! Один кратный корень
            print *,' x= ',-b/(2.d0*a)
        else
            d=sqrt(d)
            x1=(-b+d)/(2.d0*a)
            x2=(-b-d)/(2.d0*a)
            print *,' x1= ',x1,'  x2=',x2
        end if
    else
```

```
        print *,' Дискриминант < 0. Нет решений'
    end if ! ***** end  if(d.ge.0.d0)
else ! Линейное уравнение, т.к. a=0
        if(abs(b).gt.0. Dnull) then
            print *,'x=',-c/b
        else
            if(abs(c).lt. Dnull.) then
                print *,' Любое x решение'
            else
              print *,' Нет решений'
            end if
        end if  ! end  **** if(abs(b).gt.0. Dnull)
end if
end
```

3.2.2.5. Составной блочный if

Общий вид:

IF(L) THEN
  S
ELSE IF(L_1) THEN
  S_1
ELSE IF(L_2) THEN
  S_2
.....
ELSE IF(L_N) THEN
  S_N
ELSE
  S_{N+1}
END IF
	где L, L_1, L_2, ...,L_N - логические выражения; S, S_1, S_2, ...,S_N, S_{N+1} - блоки операторов. Оператор работает следующим образом:
	Вычисляется логическое выражение L, если оно истинно, то выполняется блок операторов S, после чего управление передается оператору следующему за END IF; иначе вычисляется логическое выражение L_1, если оно истинно, то выполняется блок операторов S_1, после чего управление передается оператору, следующему за END IF, и т.д. Если все логические выражения L, L_1, L_2, ...,L_N ложны, то выполняется блок операторов S_{N+1}. Т.о., выполняется ближайший блок операторов Si, для которого логическое условие L_i принимает значение истина, после чего происходит выход из блока.

Пример. Вычислить значение функции:

$$y = \begin{cases} 0, & x < -1 \\ \cos\left(\pi x\right), & x \in [-1, 0) \\ x^2 + 1, & x \in [0, 2) \\ 7 - x, & x \in [2, 7) \\ 0, & x \geq 7. \end{cases}$$

```
$freeform
program ifelsefun
print *,'Введите аргумент функции'
read *,x
if(x.lt.-1.) then
   y=0
else if(x.lt.0.) then
```

```
   y=cos(3.14159*x)
else if(x.lt.2.) then
   y=x**2+1.
else if(x.lt.7.) then
   y=7.-x
else
   y=0.
End if
print *,'y=',y
end
```

3.2.3. Оператор выбора

В разветвляющихся на много веток алгоритмах достаточно часто применяется оператор выбора. В зависимости от значения управляющего параметра управление программой передается в определенный блок:

SELECT CASE (key) CASE (Lkey1) S1 CASE (Lkey2) S2 ..... CASE (LkeyN) SN CASE DEFAULT SN+1 END SELECT	где key - управляющий параметр типа int, logical или character*1. Lkey1, Lkey2, ... , LkeyN - список значений, которые могут быть константными выражениями. S1, S2, ... , SN, SN+1 - блоки операторов. CASE DEFAULT является необязательным. Список значений Lkeyi можно задавать либо одним значением, либо списком отдельных значений, разделенных запятыми, либо диапазоном значений, разделенных двоеточием. Например: CASE (5,12,19) - блок выполняется, если управляющий параметр равен либо 5, либо 12, либо 19. CASE (2:10) - блок выполняется, если управляющий параметр заключен в диапазоне от 2 до 12. CASE ('A':'z') - блок выполняется, если управляющий параметр символьного типа является латинской буквой.

Пример. Программа должна вывести количество дней в текущем месяце.

```
$freeform
program number_day
integer*2 ye,mon,day
call getdat(ye,mon,day) ! Считать с системных часов дату
select case(mon)
 case (1,3,5,7,8,10,12)! В эти месяцы 31 день
   n_day=31
 case (4,6,9,11) )! В эти месяцы 30 день
  n_day=30
 case (2)
```

```
    if(mod(ye,4).eq.0) then! Високосный год
      n_day=29
    else
     n_day=28
    end if
end select
print *, ‘ В текущем месяце ‘,n_day,’ дней’
end
```

3.3. Операторы цикла

Для выполнения циклических процессов используются операторы цикла. В Фортране 90 существуют два типа оператора цикла: перечисляемый цикл и цикл с предусловием. Операторы цикла позволяют многократно повторять вычисления некоторого участка программы, называемого областью цикла, при разных значениях некоторых параметров.

3.3.1. Перечисляемый цикл

Общий вид оператора:

```
DO m  dovar = begin, end[,inc]
    Операторы, составляющие тело цикла.
m  S
```

или

```
DO dovar =begin, end[,inc]
    Операторы, составляющие тело цикла.
END DO
```

В программах написанных на современных языках программирования (C++, Pascal, Qbasic, Visual Basic, Java), операторы помечают только в исключительных случаях, т.к. все операторы являются структурированными. Поэтому мы также, в основном, будем использовать структурированный оператор DO.

В этих операторах m - метка исполняемого оператора, обозначающая последний оператор области действия цикла. Этот оператор также входит в тело цикла.

dovar - переменная целого или действительного типа (int или real), называемая параметром цикла.

begin, end – выражения, задающие начальное и конечное значения параметра цикла.

inc - необязательное выражение, задающее приращение параметра цикла dovar называемое шагом параметра цикла.. Если шаг не указывается (по умолчанию), то он принимается равным 1.

Оператор работает следующим образом:

Параметру цикла dovar присваивается начальное значение begin.

Проверяется выполнение условие sgn(inc)*dovar ≤ end *sgn(inc), где sgn - функция определяющая знак аргумента.

Если это логическое условие принимает значение «истина», то выполняются операторы тела цикла и происходит переход на пункт 4. Если логическое усло-

вие пункта 2 принимает значение «ложь», то происходит выход из цикла, т.е. переход на пункт 5.

Параметр цикла получает приращение на шаг inc (т.е. выполняется оператор присваивания dovar=dovar+inc), и происходит переход на пункт 2.

Выполняется следующий за END DO оператор.

Примечание 1. На последний оператор цикла с меткой накладывается множество ограничений. Мы их не перечисляем, т.к. настоятельно советуем применять структурированный оператор с последним оператором тела цикла END DO.

Примечание 2. Внутри цикла нельзя принудительно изменять переменные цикла: dovar, begin, end, inc.

Примечание 3. При использовании вложенных циклов закрывается вначале ближайший к текущему оператору END DO незакрытый цикл.

Примеры.

Пример 1. Найти сумму первых 1000 членов ряда

$$\sum_{n=1}^{1000} \frac{1}{n^2}.$$

		program sum s=0. Do n=1,1000 s=s+1./float(n**2) end do print *,' s=', s end	Данная программа написана в фиксированном формате. Поэтому мы выделяем 6-ую позицию, в которой ставится символ продолжения, а с 1 по 5 позиции ставятся метки операторов.

Пример 2. Написать программу, которая табулирует функцию $y=\sin^2 x$ в диапазоне $[0,\pi]$.

$freeform program tabul real*8 pi/3.1415926535897d0/,h h=pi/10. do x=0.,pi+h/10,h Print *,' x= ',x,' y=', sin(x) *2 end do end	При использовании в качестве параметра цикла переменной действительного типа приращение параметра цикла вычисляется приближенно. Поэтому конечное значение параметра цикла необходимо немного увеличить, чтобы цикл выполнился для конечного значения параметра цикла. В данном примере конечное значение параметра цикла равно не pi, а pi+h/10.

Пример 3. Найти двухмерную сумму $\sum_{i=1}^{80}\sum_{j=1}^{50}\frac{j+i}{i^2+j^3}$.

```
$freeform
program sum2
real*8 s/0.d0/
do i=1,80
do j=1,50
s=s+Dble(i+j)/Dble(i**2+i**3)   !Dble перевод числа в формат с
end do                          ! двойной точностью. Иначе, почти
end do                          !равны нулю, т.к. целое/целое=целое.
print *,s
end
```

3.3.2. Цикл с предусловием

Общий вид оператора:

```
   DO m  WHILE(L)
     Операторы, составляющие тело цикла.
m  S
```

или

```
DO WHILE(L)
     Операторы, составляющие тело цикла.
END DO
```

где L - логическое выражение.

Оператор работает следующим образом:

Операторы, составляющие тело цикла, выполняются до тех пор, пока логическое выражение L принимает значение истина.

Пример 2. Написать игровую программу. Машина загадывает натуральное трехзначное случайное число. Человек должен за меньшее число попыток угадать его.

```
$freeform
program RanNumber
integer*2 hr,sec,i100th, NumberRand, number/0/,ic/0/
Call Gettim(hr,sec,i100th)          ! Получение текущего времени
Call Seed(sec*100+i100th)           ! Настройка ряда случайных чисел
Call Random(rand)                   ! Получение случайного числа ∈[0,1]
NumberRand=900*rand+100             ! Получение натурального трех
"значного случайного числа
″ Выполнять пока не угадали число случайное число NumberRand
Do while(number .ne. NumberRand)!
  Print *, ‘ Введите трехзначное число’
  read *,number
  ic=ic+1
  if(number .gt. NubmerRand) then
    print *,’Много’
  else
   print *,’Мало’
```

```
  end if
end do    ! конец тела цикла
select case (ic)
  case (1:9)
    print *,' Гений! Угадал за ',ic,' попыток'
  case (10:15)
    print *,' Молодец! Угадал за ',ic,' попыток'
  case (16:20)
    print *,' Неплохо. Угадал за ',ic,' попыток'
  case default
    print *,' Плохо. Угадал только за ',ic,' попыток.'
  end select
end
```

3.3.3. Операторы CYCLE и EXIT

Операторы CYCLE и EXIT применяются только внутри тела цикла и относятся только к одному текущему циклу. При встрече оператора CYCLE программа пропускает все невыполненные пока операторы тела цикла и переходит к следующему проходу цикла, начиная с первого оператора тела цикла с новыми значениями параметра цикла для перечисляемого оператора цикла. При встрече оператора EXIT программа завершает выполнение текущего оператора цикла, и управление передается следующему за END DO оператор.

Пример. Вводятся ряд ненулевых действительных чисел. Найти количество и сумму положительных чисел. Ввод чисел прекратить после ввода первого нуля.

```
$freeform
program sumplus
real s/0./, x
integer nplus/0/
do while(.true.)                ! бесконечный цикл!
   read *,x
   if(abs(x).lt.0.1e-30) exit   !Выйти из цикла, если x=0.
   if(x.lt.0) cycle             ! Обойти накопление сумм, если x < 0.
   s=s+x
   nplus=nplus+1
end do
print *,' Количество положительных чисел =',nplus,'  Сумма =',s
end
```

4. Массивы и структуры

4.1. Массивы переменных

Массив - это упорядоченная последовательность данных, занимающих непрерывную область памяти, к которой можно обратиться по имени. Элемент этой последовательности называется *элементом* массива.

4.1.1. Описание массива

При описании массива необходимо указать его тип, размерность и диапазон изменения индексов. *Размерность* массива — это количество измерений. Бывают одномерные (векторы), двухмерные (матрицы или таблицы) и многомерные массивы. Обычно массивы описывают в операторах описания переменных INTEGER, REAL, CHARACTER, LOGICAL и т.д. Описание массива имеет вид: name(d_1,d_2,...,d_n), где name - имя массива; d_1,d_2,...,d_n - диапазон изменения индексов; n - размерность массива. При этом d_i имеют вид (n_1:n_2) (n_1 - нижняя, а n_2 - верхняя граница изменения индексов). Размерность массива n не может превышать семи. В случае, если n_1 равно 1, т.е. нумерация элементов массива начинается с 1, применяется упрощенная форма задания d_i - (n), где n количество переменных в массиве.

Примеры.

Real a(0:100),b(3,3),c(-2:4)

Описываются три действительных массива a, b и c. В массиве a 101 число a_0,a_1, ..., a_{100}. Массив b является матрицей состоящей из 9 элементов $b_{1,1}$, $b_{2,1}$, $b_{3,1}$, $b_{1,2}$,$b_{2,2}$, $b_{3,2}$, $b_{1,3}$, $b_{2,3}$, $b_{3,3}$.В памяти двухмерные массивы хранятся по столбцам, т.е. массив b хранится, как в приведенной выше последовательности. Массив c состоит из семи элементов c_{-2}, c_{-1}, c_0, c_1, c_2, c_3, c_4.

Обращение к элементу массива производится по имени массива, за которым следует в круглых скобках индексы разделенными запятыми.

Например, чтобы обратиться к элементу матрицы b, стоящему в третьей строке и втором столбце, необходимо написать b(3,2).

Пример. Вводится 100 случайных целых чисел в диапазоне от -100 до 1000. Найти минимальное, среднее арифметическое и максимальное значения этих чисел. Получить и вывести новый массив, разделив элементы первого массива на среднее арифметическое первого массива.

```
$freeform
program massivrand
parameter (N=100)     ! Удобно описывать размерности массивов
integer a(N),b(N),max/-2000000000/,min/2000000000/      !-∝ и +∝
real s/0./
integer*2 hr,sec,i100th
Call Gettim(hr,sec,i100th)    ! Получение текущего времени
Call Seed(sec*100+i100th)   ! Настройка ряда случайных чисел
do i=1,N
```

```
Call Random(rand)            ! Получение случайного числа rand∈[0,1]
a(i)=rand*1100-100           ! Получение случайного числа rand∈[-100,1000]
if(a(i).lt.min) min=a(i)
if(a(i).gt.max) max=a(i)
s=s+a(i)
end do
s=s/N
print *,' Минимальное =',min,'Среднее =',s,'  Максимальное = ',max
do i=1,N
b(i)=a(i)/s
print *,b(i)
end do
```

4.2. Подстроки

Подстрока - это непрерывная часть переменной или элемента массива типа CHARACTER. Обращение к подстроке записывается следующим образом:

$S(n_1:n_2)$, где S - имя символьной переменной; n_1, n_2 - левая и правая границы подстроки. Если строка является элементом массива, то обращение к подстроке осуществляется следующим образом:

$S(I)(n_1:n_2)$, где I - номер элемента массива.

Пример 1. Вводится текстовая строка на русском языке. Необходимо все русские маленькие буквы поменять на большие.

```
$freeform
program substring
character L*80,sym*1
character
ALFUP*33/'АБВГДЕУЖЗИЙКЛМНОПРСТУФХЦЧШЩЫЬЪЭЮЯ'/
character alflit*33/'абвгдеужзийклмнопрстуфхцчшщыьъэюя'/
print *,'Введите строку на русском языке'
read '(a)',L                          ! Считать строку
print *,' до преобразования'
print '(1x,(a))',L                    ! Напечатать строку L
n2=Len_Trim(L)                        ! n2 - длина строки
do i=1,n2                             ! цикл по всем буквам строки
  sym=L(i:i)                          ! извлечь i-тый символ из строки
  do j=1,33                           ! поискать i-тую букву среди алфавита
    if(sym.ne.alflit(j:j)) cycle      ! перейти на конец внутреннего цикла
    L(i:i)=ALFUP(j:j)                 ! поменять на большую букву
    exit                              ! и выйти из внутреннего цикла
  end do
end do
print *,' после преобразования'
print '(1x,(a))',L
```

```
end
```

Пример 2. Вводится текстовая строка длиной не более 255 символов на русском языке. Необходимо заменить все слова *мама* на *папа*, если они имеются, и подсчитать количество замен.

```
$freeform
program substring
character L*255
print *,'Введите строку на русском алфавите'
read '(a)',L                          ! Считать строку
print *,' до преобразования'
print '(1x,(a))',L                    ! Напечатать строку L
n2=Len_Trim(L)                        ! n2 - длина строки
do i=1,n2-3                           ! цикл по всем буквам строки
  if(L(i:i+3) .eq. ‘мама’) L(i:i+3)=’папа’
end do
print *,' после преобразования'
print '(1x,(a))',L
end
```

Пример 3. Вводиться строка не более 80 символов. Поменять все слова Дима на слова Дмитрий.

В данном примере необходимо расширить длину строки.

```
$freeform
program substring
character L*80,L1*120
print *,'Введите строку на русском алфавите'
read '(a)',L              ! Считать строку
print *,' до преобразования'
print '(1x,(a))',L        ! Напечатать строку L
n2=Len_Trim(L)                ! n2 - длина строки без хвостовых пробелов
k=1                       ! Номер текущего символа входной строки
k1=1                      ! Номер текущего символа выходной строки
do while(.true.)          ! бесконечный цикл
   if(L(k:k+3) .eq. ‘Дима’) Then
     L1(k1:k1+6)=’Дмитрий’
     k=k+4`
     k1=k1+7
  else
    L1(k1:k1)=L(k:k)
    k=k+1
    k1=k1+1
  end if
  if(k.gt.n2) exit        ! Выход из цикла
end do
```

```
print *,' после преобразования'
print '(1x,(a))', L1(1:k1)
end
```

Пример 4. С клавиатуры вводится фамилия, имя и отчество студента. Необходимо вывести фамилию и инициалы.

```
$freeform
program Namestring
character Fam*15,Name*12,Name2*18,d*45,d1*23/' '/
print *,'Введите фамилию, имя и отчество'
read '(a)',Fam,Name,Name2          ! Прочитать фамилию, имя и отчество
d=Fam//' '//Name(1:1)//'. '//Name2(1:1)//'.'   !Соединить в одну цепочку "фамилию и  инициалы
n=Len_Trim(d)                  ! n - длина строки без хвостовых пробелов
k=1
do i=1,n
 if(d(i:i+1).ne.'  ')  then                  ! n – Если не два не пробела
  k=k+1
  d1(k:k)=d(i:i)
 end if
end do
print *,d1(1:k1)
end
```

4.3. Структуры

В современных языках программирования появились сложные, вводимые пользователем типы данных, называемые *структурами.* Допустим, сведения о сотруднике учреждения являются типом данных, в котором определены необходимые характеристики: фамилия, имя, отчество, домашний адрес, телефон, возраст, время принятия на работу, день рождения и т.д. Вместо того чтобы для каждого из этих сведений вводить отдельную переменную, можно ввести одну сложную структурированную переменную.

4.3.1. Оператор STRUCTURE

Общий вид оператора STRUCTURE:

STRUCTURE /type_name/ element1 element2 ……. elementn END STRUCTURE	где type_name - имя типа структурированной переменной; element1, element2, ... , elementn - описание простых переменных, образующих сложную структурированную переменную. Эти простые переменные называются полями. При обращение к любому полю структурированной переменной указывается имя структуры и через “точку” имя поля.

Оператор STRUCTURE вводит новый пользовательский тип переменных структурного типа. Для того чтобы использовать переменную структурного ти-

па, необходимо при помощи оператора описания переменных структурного типа RECORD создать объект этого типа.

4.3.2. Оператор RECORD

Общий вид оператора:

RECORD /type_name/ name1(d1),name2(d2), ... , namen(dn)

где - type_name имя типа данных; name1, name2, ... , namen - имена переменных, а d1, d2, ..., dn - размерности массивов переменных типа type_name, если переменные не массивы, то (di) опущены.

Оператор RECORD сообщает компилятору, сколько байт памяти необходимо отвести под переменные, описанные в операторе.

Пример. Ввести внутри программы переменную, описывающую сведения о человеке:

Фамилию, имя, отчество, телефон, возраст. Для трех человек заполнить внутри программы операторами присваивания значения этих переменных. Подсчитать и вывести на монитор средний возраст и среднюю длину их фамилий.

```
$freeform
program struct1
parameter (N=3)
character d*255
″ Введение нового типа переменных
structure/full_name/
  character*15 first_name   !Фамилия
  character*20 last_name    !Имя
  character*20 p_name       !Отчество
  character*9 tell          !Номер телефона
  integer*2 ye              !Возраст
end structure
record /full_name/ Name(N)        ! Описание масс. Перемен. типа full_name
″ Заполнение полей структуры данными
Name(1).first_name="Иванов"
Name(1).last_name="Петр"
Name(1).p_name="Андреевич"
Name(1).tell="277-xx-27"  !Не звонить
Name(1).ye=45
Name(2).first_name="Петрова"
Name(2).last_name="Мария"
Name(2).p_name="Николаевна"
Name(2).tell="277-xx-77"  !Не звонить
Name(2).ye=17
Name(3).first_name="Александров"
Name(3).last_name="Петр"
Name(3).p_name="Владимирович"
Name(3).tell="277-xx-77"  !Не звонить
```

```
Name(3).ye=60
do i=1,N
  s=s+ Name(i).ye
end do
print *,'Средний возраст =',s/n
″ Все фамилии в одну переменную, чтобы можно обрабатывать
d=Name(1).first_name// Name(2).first_name// Name(3).first_name
k=0
do i=1,Len_Trim(d)          ! Количество букв во всех фамилиях
  if(d(i:i).ne.' ') k=k+1
end do
" Float(k) - перевод целого числа k в   формат  числа действительного
print *,' средняя длина фамилии = ' ,Float(k)/Float(N)
end
```

Пример. Ввести из файла N переменных приведенного в предыдущем примере типа. Отсортировать эти переменные по возрастанию указанного ключа. В качестве ключа сортировки предусмотреть либо первые буквы фамилии, либо первые буквы имени, либо первую цифру телефона, либо возраст.

```
$freeform
program strsort
include 'strname.str'
record /full_name/ Name(100)
print *,',Введите номер ключа сортировки'
read *,number_key
open(5,file='struct1.dat')
read (5,*) N
read (5,'(1x,a15,a20,a20,a9,i6)') (Name(i).first_name, -
Name(i).last_name,Name(i).p_name,Name(i).tell,Name(i).ye,i=1,N)
print *,'До сортировки'
call sprint(Name,N)
call sort(Name,N,number_key)
print *,'После сортировки'
call sprint(Name,N)
end
subroutine sprint(Name,N)
include 'strname.str'
record /full_name/ Name(N)
100 format(1x,'Фамилия',10x,'Имя',15x,'Отчество',12x,'Телефон',' Возраст')
print '(1x,a15,a20,a20,a9,i6)',(Name(i).first_name, -
Name(i).last_name,Name(i).p_name,Name(i).tell,Name(i).ye,i=1,N)
return
end
subroutine sort(Name,N,number_key)! Сортировка методом пузырька
```

```
include 'strname.str'
record /full_name/ Name(N)
record /full_name/ t
integer funnum_simb
do i=1,N-1
  imax=i
  t=Name(i)
  nt=funnum_simb(t,number_key)
  do k=i+1,N
    if(funnum_simb(Name(k),number_key) .lt. nt) then
      imax=k
      t=Name(k)
      nt=funnum_simb(t,number_key)
    end if
  end do
  Name(imax)=Name(i)
  Name(i)=t
end do
return
end
function funnum_simb(t,number_key)
include 'strname.str'
record /full_name/ t
integer funnum_simb
character c,ALFUP*39
DATA ALFUP/'-0123456789АБВГДЕЖЗИЙКЛМНОПРСТУФХЦШЩЬЪЭЮЯ'/
select case (number_key)
  case(1) ! Номер ключа сортировки
    c=t.first_name(1:1)
  case(2)
    c=t.last_name(1:1)
  case(3)
    c=t.p_name(1:1)
  case(4)
    read(t.tell,'(i3)') funnum_simb
    return
  case(5)
    funnum_simb=t.ye
    return
  case default
    print *,' Неверен ввод ключа -  number_key= ',number_key
    stop
end select
do i=1,39
```

```
  funnum_simb=i
  if(c.eq.ALFUP(i:i)) return
end do
print *,'незнакомый символ ',c
end
```

Прежде чем запустить программу на компиляцию, необходимо ввести в файл strname.str описание элемента типа full_name. Т.е. следующее содержимое:

```
structure/full_name/
  character*15 first_name   !Фамилия
  character*20 last_name    !Имя
  character*20 p_name       !Отчество
  character*9 tell          !Номер телефона
  integer*2 ye              !Возраст
end structure
```

4.4. Динамические массивы

Довольно часто при написании программы программист не знает количество элементов в массиве, который будет использоваться в программе. Для выхода из этой ситуации возможны несколько путей.

Первый путь - описать массив с максимально возможной размерностью и предусмотреть останов программы и сообщение о причинах останова в случае превышения размерности массива. Для сложных программ, требующих большого объема памяти, этот путь неприемлем. При этом на этапе компиляции для каждого такого массива (такие массивы называются статические) отводится оперативная память, соответствующая размеру массива.

Второй более оптимальный путь - использование динамических массивов.

В языке Фортран 90 массив может быть размещен динамически, т.е. размер каждого измерения может устанавливаться на этапе исполнения, а не во время компиляции программы. Для любого динамического массива необходимо, чтобы он был описан с атрибутом ALLOCATABLE и было задано число его измерений.

Например:

INTEGER Arr[ALLOCATABLE](:,:,:)

Здесь описывается трехмерный целочисленный динамический массив.

Arr - идентификатор массива. ALLOCATABLE - атрибут массива, который указывает, что данный массив является динамическим и что для него не надо отводить место на этапе компиляции.

Естественно, в программе необходимо указать, в каком месте, и какого размера ввести динамический массив. Это производится оператором **ALLOCATE**.

4.4.1. Оператор ALLOCATE

Динамически размещает массив, ранее объявленный с атрибутом ALLOCATABLE.

Общий вид:

ALLOCATE(name(d_1,d_2,...,d_n)[,STATE=ierr])

где name - имя массива; d_1,d_2,...,d_n - диапазон изменения индексов; n - размерность массива; ierr - целочисленная переменная, возвращающая состояние, получаемое при выполнении попытки размещения.

Оператор ALLOCATE создает верхнюю и нижнюю границы каждого измерения и резервирует необходимую память.

В операторе ALLOCATE может быть задано более одного массива; они разделяются запятыми. Последним должен быть задан параметр STATE= .

Попытка переразместить ранее размещенный массив приводит к ошибке этапа исполнения. Любая неудача при выполнении операции размещения приводит к ошибке этапа исполнения, если не задана операция state =. Переменная ierr возвращает значение ноль, если операция размещения была выполнена успешно; иначе возвращается номер ошибки этапа исполнения.

4.4.2. Освобождение использованной памяти. Оператор DEALLOCATE

После использования размещаемого массива, заданного с помощью ALLOCATE, оператор DEALLOCATE возвращает область памяти, которую занимал массив.

Общий вид:

DEALLOCATE(nameList[,STATE=ierr])

где - nameList -список имен размещаемых массивов; если указано несколько имен, то они разделяются запятыми. ierr - целочисленная переменная, возвращающая статус попытки размещения.

Аналогично оператору ALLOCATE попытка переразместить массив, который не был ранее размещен, приводит к ошибке этапа исполнения. Любая неудача при выполнении операции размещения приводит к ошибке этапа исполнения, если не задана операция state =. Переменная ierr возвращает значение ноль, если операция переразмещения была выполнена успешно; иначе возвращается номер ошибки этапа исполнения.

Если на размещаемый массив задается ссылка в тот момент, когда он не является размещенным, результаты могут быть непредсказуемыми.

Пример 1.

```
$FREEFORM
program NumbArr
integer*2 n
integer*2 Arr_1[allocatable](:), Arr_2[allocatable](:)
print *,' Введите размер массива '
read *,n
if((n.gt.0).and.(n.le.300)) then
  allocate(Arr_1(0:n-1),Arr_2(n),stat=ierr)
  print *, ' Array = '
```

```
  do i=0,n-1
    Arr_1(i)=i
    Arr_2(i)=2*i
    print * ,' ',Arr_1(i),Arr_2(i)
  end do
  deallocate(Arr_1,Arr_2)
else
  print *,' Error!!! '
end if
end
```

Программа динамически, в интерфейсе с пользователем создает массивы Arr_1 и Arr_2 , заполняет массив Arr_1 числами натурального ряда от 0 до n, а массив Arr_2 четными числами от 0 до 2n.

Пример 2. Ввести N случайных чисел и подсчитать среднее значение их, а также количество чисел больших среднего.

```
$freeform
program rand1
integer*4 a[ALLOCATABLE](:)   ! описание динамического массива
integer*2 error,sec,i100sec,hr,min
print *,' введите число N'
read *,n
ALLOCATE(a(n),STAT=error) ! создание динамического массива
call gettim(hr,min,sec,i100sec) ! узнать текущее время
call seed(sec*i100sec+hr+min)! построить ряд псевдослучайных чисел
s=0.
do i=1,N
  call random(r)  ! получить очередное псевдослучайное число
  a(i)=r*10000 ! получить число в диапазоне 0 - 10000
  s=s+a(i)
end do
s=s/N
print *,' Среднее значение ',n, -
' случайных чисел в диапазоне 0 - 10000 =',s
k=0
do i=1,N
  if(a(i).Gt.s) k=k+1
end do
print *,' Количество чисел больших среднего значения = ‘,k
deallocate(a)   ! Освободить память занимаемую массивом a
stop
end
```

5. Операторы организации функций и подпрограмм

5.1. Операторные функции

Часто в программе приходится обращаться к одной и той же математической формуле при различных значениях параметров. В этом случае удобно один раз описать эту формулу при помощи операторной формулы, а затем при необходимости вызывать ее с различными значениями аргументов. Для этого перед первым выполняемым оператором необходимо определить операторную функцию в виде:

F_name(x1,x2, ... , xn) = E

где F_name - имя операторной функции; x1,x2, ... , xn - ***формальные параметры*** (аргументы); E - формула для математического выражения. Формальными параметрами являются идентификаторы переменных простых типов. Для вызова операторной функции на выполнение необходимо написать ее имя и в круглых скобках через запятую перечислить значения параметров, при которых необходимо вычислить математическое выражение.

Параметры, при которых происходит вызов операторной функции, называются фактическими параметрами. Фактическими параметрами могут быть константы, простые переменные, элементы массивов, обращения к полям структурных элементов, обращения к встроенным функциям, другим операторным функциям, функциям пользователей, а также арифметические выражения перечисленных величин.

Операторная функция возвращает число простого типа.

Пример. Написать программу для вычисления по введенным длинам сторон треугольника значения его углов.

```
$freeform
program tringl_ugol
real*8 a,b,c,x
ugol(a,b,c)=acos((a**2+b**2-c**2)/(2.d0*a*b))
Radgrad(x)=x/3.14159d0*180.d0          ! Перевод в градусы
print *,' Введите три стороны треугольника'
1 read *,a, b, c
if(a+b.le.c .or. b+c .le.a .or. a+c.le.b) then
  print *,'Введенные числа не образуют треугольник'
  go to 1
end if
print *,'Углы в радианах'
print *,' alfa=', ugol(b,c,a),' beta=', ugol(a,c,b),' hamma=',ugol(a,b,c)
print *'Углы в градусах'
print *,' alfa=', Radgrad(ugol(b,c,a)),' beta=', Radgrad(ugol(a,c,b)),'-
hamma=',Ragrad(ugol(a,b,c))
end
```

В данном примере используются две операторные функции: ugol - для вычисления угла между сторонами треугольника по теореме косинусов и Radgrad - для перевода угла, заданного в радианах, в угол, заданный в градусах.

5.2. Подпрограммы-функции

Подпрограмма-функция является отдельной программной единицей и может быть откомпилирована отдельно от основной программы и записана в библиотеку подпрограмм. Подпрограмма-функция может вычислить только одно число, являющееся значением функции в зависимости от значений аргументов. Т.е. это отображение n-мерного пространства множества значений аргументов на одномерное пространство множества значение функции.

5.2.1. Оператор FUNCTION

Подпрограмма-функция определяется ключевым словом FUNCTION. Общая форма подпрограммы-функции:

```
тип FUNCTION имя( a1,a2,...,an)
  неисполняемые операторы (операторы описания).
  исполняемые операторы, среди которых должен быть оператор.
  присваивания: имя= значение или выражение.
END
```

где тип - тип функции; ai - формальные параметры (аргументы).

Формальные параметры могут быть именем простой переменной, именем структуры, массивом, именем другой подпрограммы-функции или подпрограммы.

Замечание. Список параметров может отсутствовать, но скобки обязаны быть.

Вызов подпрограммы-функции осуществляется так же, как и встроенной функции и оператора-функции.

5.2.2. Оператор RETURN

Если подпрограмму-функцию необходимо прерывать внутри, то можно использовать оператор возврата RETURN. При встрече этого оператора происходит завершение работы подпрограммы-функции и передача управления в то место программы, откуда была вызвана подпрограмма-функция. При отсутствии оператора RETURN подпрограмма-функция выполняется до оператора END.

Пример. Написать функцию, которая находит среднеарифметическое значение одномерного массива размерности n. Написать программу, тестирующую полученную функцию.

```
$freeform
program sredn
real*4 a[ALLOCATABLE](:)   ! описание динамического массива
integer*2 error
print *,' Введите количество элементов'
read *,n
```

```
allocate(a(n),STAT=error) ! создание динамического массива
do i=1,n
  read *,a(i)
end do
print *,funsredn(a,n)
end
real function funsredn(mas,n)
real*4 mas(n)
s=0.
do i=1,n
  s=s+mas(i)
end do
fun2=s/n
end
```

5.2.3. Особенности использования в качестве формальных параметров массивов

Память для формальных параметров определяется компилятором в вызывающих подпрограмму-функцию программных модулях (основной программе, другой подпрограмме-функции или подпрограмме), а в подпрограмму-функцию передается только адрес начала массива. Поэтому при описании массивов в подпрограммах-функциях есть некоторые особенности.

Если формальным параметром является одномерный массив, то его размерность можно описать через другой формальный параметр, или поставить символ «*», или указать минимальную размерность «1». В предыдущем примере можно было вместо оператора real*4 mass(n) поставить оператор real*4 mass(*) или real*4 mass(1).

Двухмерные массивы хранятся в памяти по столбцам, т.е. быстрее изменяется первый индекс. При описании мы обязаны указать точное значение количества строк.

Например: integer mass(10,*).

Здесь описывается формальный параметр, являющийся двухмерным массивом, имеющим 10 строк и произвольное количество столбцов.

Примечание. Основной ошибкой в сложных программах является ошибка адресации. Это когда в основной программе массив описан меньшей размерности, чем используется в подпрограммах или подпрограммах-функциях.

Пример.

```
$freeform
program sredn
real*4 a(10)
print *,' Введите количество элементов'
read *,n
print *,fun1(a,n),fun2(a,n),fun1(a,n))
end
real function fun1(mas,n)
```

```
      real*4 mas(*)
      do i=1,n
           mas(i)=i
      end do
      s=0.
      do i=1,n
           s=s+mas(i)
      end do
      fun1=s/n
      end
* Функция для определения среднего отклонения массива от его
* среднеарифметического   значения ss
      real function fun2(mas,n,ss)
      real*4 mas(*)
      s=0.
      do i=1,n
           s=s+abs(mas(i)-ss)
      end do
      fun2=s/n
      end
```

В приведенной программе под массив а отводится 10 элементов. Вызывается функция fun1, которая заполняет массив **а** и вычисляет его среднеарифметическое значение. Другая функция вызывается после выполнения первой и вычисляет среднее по абсолютной величине отклонение элементов массива от его среднего значения. Эта программа будет прекрасно работать, только если $n \leq 10$. При $n > 10$ программа будет вести себя неоднозначно и неправильно, т.к. при работе функции fun1 происходит затирание некоторой информации, относящейся к другим данным. В лучшем случае программа «зависнет», а в худшем исказит результаты. Такие ошибки очень трудно найти, используя даже различные отладчики.

5.3. Подпрограммы

Подпрограммой называется поименованная часть программы, имеющая формальные параметры. Подпрограмма является отдельной программной единицей и может быть откомпилирована отдельно от основной программы и записана в библиотеку подпрограмм. Подпрограмма является расширением подпрограммы – функции, и ее можно рассматривать как математическое отображение n-мерного пространства множества значений аргументов на m-мерное пространство множества значений отображения. Обычно подпрограмма имеет входные и выходные параметры. Входные параметры — это такие параметры, которые остаются без изменения в подпрограмме. Выходные параметры, это параметры которые при вызове подпрограммы имеют неопределенное значения, а в ходе выполнения программы получают определенное значение, зависящее от входных параметров. Правда, часто параметры бывают одновременно и входными, и выходными.

5.3.1. Оператор описания подпрограммы. SUBROUTINE

Подпрограмма определяется ключевым словом SUBROUTINE. Общая форма подпрограммы почти такая же, как и подпрограммы-функции:

```
SUBROUTINE имя( a1,a2,...,an)
неисполняемые операторы (операторы описания)
исполняемые операторы
END
```

где имя - имя подпрограммы; ai - формальные параметры (аргументы).

При вызове программа выполняется от первого выполняемого оператора до первого встреченного оператора RETURN или до оператора END, который является последним оператором в подпрограмме. В подпрограмме имя подпрограммы не несет на себе числовую информацию.

5.3.2. Оператор вызова подпрограммы. CALL

В отличие от подпрограммы-функции подпрограмма вызывается отдельным оператором вызова CALL следующим образом:

CALL ИМЯ(y1,y2,....,yn)

где y1,y2,...,yn - фактические параметры, которые могут быть константами (если формальные параметры являются только входными параметрами), простыми переменными любого типа, элементами массивов, массивами, структурированными элементами, арифметическими выражениями, именами подпрограмм или подпрограмм-функций, именами встроенных функций или подпрограмм. Естественно, между формальными и фактическими переменными должно быть соответствие типов.

5.3.3. Примеры подпрограмм

Пример 1. Три матрицы A, B и C порядка 5 заполняются случайными числами в диапазоне от 0 до 50. Вычислить нормы матриц AB, AC и BC. Норму матрицы равна корню квадратному от суммы квадратов всех элементов матрицы.

```
Program norma_matr
parameter (N=5)
integer a(N,N),b(N,N),c(N,N),d(N,N)
integer*2 sec,i100sec,hr,min
real NormaMatr,r
call gettim(hr,min,sec,i100sec) ! узнать текущее время
call seed(sec*i100sec+hr+min)!  построить ряд псевдослучайных чисел
do i=1,N
  do j=1,N
    call random(r)  ! получить очередное псевдослучайное число
    a(i,j)=int(r*50.)
    call random(r)
    b(i,j)=int(r*50.)
    call random(r)
    c(i,j)=int(r*50.)
  end do
end do
Call printMatr(a,n,'матрица a=')
Call printMatr(b,n,'матрица b=')
Call printMatr(c,n,'матрица c=')
call MultMatr(a,b,d,n)
Call printMatr(d,n,'матрица ab')
print *,'Норма матрицы AB=', NormaMatr(d,n)
call MultMatr(a,c,d,n)
Call printMatr(d,n,'матрица ac')
print *,'Норма матрицы AC=', NormaMatr(d,n)
call MultMatr(b,c,d,n)
Call printMatr(d,n,'матрица bc')
print *,'Норма матрицы BC=',NormaMatr(d,n)
end
"Подпрограмма для вычисления произведения двух матриц C=AB
subroutine MultMatr(a,b,c,n)
integer a(n,n),b(n,n),c(n,n)
do i=1,n
  do j=1,n
    s=0. ! Скалярное произведение i-той строки на k-тый столбец
    Do k=1,n
      s=s+a(i,k)*b(k,j)
    end do
    c(i,j)=s
  end do
end do
end
```

```
" Функция для вычисления нормы матрицы a
real function NormaMatr(a,n)
integer a(n,n)
S=0.
Do i=1,n
  do j=1,n
    S=S+a(i,j)**2
  end do
end do
NormaMatr=sqrt(S)
end
subroutine printMatr(a,n,s)! Вывести на экран массив
character*10 s
integer a(n,n)
print *,s ! Вывести заглавие 10 символов
do i=1,n
  print *,(a(i,j),j=1,n)! Это неявный цикл. См. раздел Ввод-вывод
end do
read * ! ждать пока нажмем любую клавишу
end
```

В данном примере приведены две подпрограммы MultMatr и printMatr и одна функция NormaMatr, которые вызываются несколько раз с различными фактическими параметрами. В подпрограмме MultMatr четыре формальных параметра a, b, c и n. Первые три параметра являются двухмерными массивами. При этом a, b и n только входные параметры, т.к. внутри подпрограммы они не изменяются, а массив d является выходным параметром, т.к. он получается внутри подпрограммы.

В подпрограмме printMatr два формальных параметра a и s. Эта программа печатает на одной строке заглавие, а на следующих n строках матрицу построчно. После того как матрица напечатана, подпрограмма ждет нажатия любой клавиши. Это необходимо для изучения результатов печати.

Необходимо отметить, что для перехода к решению задачи с матрицами другого порядка необходимо всего лишь изменить одну цифру 5 в операторе PARAMETER (N=5). Правилом хорошего тона считается стремление к разумному универсализму программы.

5.3.4. Использование имен подпрограмм и подпрограмм-функций в качестве формальных операторов

Часто при решении математических задач необходимо вычислить значение некоторого выражения для различных функций, входящих в это выражение, т.е. вычислить функционал. При этом удобнее включить имя функции в формальные параметры, а затем полученную подпрограмму или подпрограмму-функцию вызывать при различных значениях фактического параметра. Нагляднее будет рассмотреть этот случай на примере вычисления определенного интеграла.

В этом случае в подпрограмме или подпрограмме-функции необходимо включить формальный параметр, определяющий формальное имя математической функции f, и описать этот параметр как переменную типа EXTERNAL. При этом в вызывающей программе необходимо также сообщить компилятору, что данный фактический параметр является особым, а именно программным модулем. Это делается оператором описания переменных INTRINSIC, если это встроенная подпрограмма-функция или подпрограмма и оператором EXTERNAL, если не встроенная.

Пример. Вычислить два числа:

1) $\int_0^1 \sin 2x + 1dx + \int_1^e \ln^2 xdx + \int_0^\pi tgxdx.$

2) $\int_0^\pi \sin^2 xdx.$

```
program integral
intrinsic tan
external fln,fsin,fsin2
print *,intgr(0.,1.,fsin)+intgr(1.,exp(1.),fln)+intgr(0,3.14159,tan)
print *,intgr(0.,3.14159,fsin2)
end
```

```
Function fsin(x)
Fsin=sin(2.*x)+1.
End
Function fln(x)
Fln=alog(x)**2
End
Function fsin2(x)
fsin2=sin(x)**2
end
```

```
 function intgr(a,b,F)
parameter (N=10000)
h=(b-a)/N
S=0.
Do i=1,N
   S=S+F(x)
x=x+h
end do
intgr=h*(S+(F(a)+F(b))/2.)
end
```

5.3.5. Способы передачи формальных параметров

По умолчанию формальные параметры, используемые в подпрограммах и подпрограммах-функциях языка Фортран, передаются по ссылке или адресу. Это означает, что для переменных не создается копия внутри подпрограммы или подпрограммы-функции, а все вычисления происходят в ячейках, отведенных компиляторам для формального аргумента. После выхода из подпрограммы или подпрограммы-функции значения фактических параметров соответствуют значению в подпрограмме. Этот метод имеет достоинства и недостатки. К достоинству следует отнести экономию памяти и время на создание копий данных. К недостаткам - иногда фактическими параметрами являются константы

или арифметические выражения. Программист должен при этом следить, чтобы такие фактические параметры не изменялись внутри подпрограммы, иначе могут быть непредсказуемые ошибки.

Во многих языках программирования (C++, Паскаль) параметры передаются по значению. Это означает, что после входа в подпрограмму или функцию создаются копии формальных параметров. Затем выполняются операторы тела подпрограммы со значениями копией параметров, а после возврата в вызвавший ее программный модуль происходит восстановление значений фактических параметров.

Для передачи формальных параметров по значению необходимо таким параметрам присваивать атрибут Value.

Например: real x[Value],y[Value].

В некоторых случаях необходимо подчеркнуть, что формальный параметр передается по ссылке. Тогда такой параметр необходимо описать с атрибутом Reference.

Например: Character*6 a[Reference],b*8[Reference],c*12 **.**

Рекомендуется по значению передавать входные параметры, а по ссылке выходные и параметры которые являются одновременно и входными и выходными.

Пример. Рассмотрим простейшую подпрограмму с двумя формальными параметрами a и b. Подпрограмма должна переставлять значения формальных параметров местами. В программе, представленной ниже, приведены три подпрограммы: Swap и SwapReference, SwapValue. Первые две передают оба параметра по ссылке, а третья по — значению. Ниже приведены результаты работы программы. На входе в подпрограммы значения фактических параметров a=1, b=2. Внутри подпрограмм параметры правильно обменивают свои значения. После возвращения в вызывающую программу подпрограмма SwapValue не меняет значения фактических параметров.

```
$freeform
subroutine Swap( a, b )
integer*2 a,b,c
c=a
a=b
b=c
print *,'Внутри подпрограммы  Swap       a=',a,'  b=',b
end
subroutine SwapReference( a, b )
integer*2 a[Reference],b[Reference],c
c=a
a=b
b=c
print *,'Внутри подпрограммы SwapReference a=',a,'  b=',b
end
```

```
subroutine SwapValue(a, b)
integer*2 a[Value],b[Value],c
c=a
a=b
b=c
print *,'Внутри подпрограммы SwapValue    a=',a,'  b=',b
end
program prswap
integer*2 a/1/,b/2/
call Swap(a,b)
print *,'После подпрограммы Swap          a=',a,'  b=',b
a=1
b=2
call SwapReference(a,b)
print *,'После подпрограммы SwapReference  a=',a,'  b=',b
a=1
b=2
call SwapValue(a,b)
print *,'После подпрограммы SwapValue      a=',a,'  b=',b
end
```

Вывод программы

```
 'Внутри подпрограммы Swap                a=      2    b=    1
  'После подпрограммы          Swap       a=      2    b=    1
 'Внутри подпрограммы SwapReference  a=      2    b=    1
'После подпрограммы   SwapReference  a=      2    b=    1
''Внутри подпрограммы SwapValue       a=      2    b=    1
'После подпрограммы SwapValue          a=      1    b=    2
```

5.4. Общие области памяти

5.4.1. Оператор COMMON

Обмен информацией между программными модулями внутри одной основной программы может производится с использованием общих областей. При этом увеличивается скорость работы программы, т.к. не надо затрачивать время на реализацию связи между фактическими и формальными параметрами. Кроме того, использование общих областей в достаточно сложных программах, требующих много оперативной памяти, позволяет многие рабочие (вспомогательные) массивы (такие, как массив D в приведенном в 5.3.3 примере) хранить в общем участке памяти.

Общая форма оператора:

COMMON /имя1/ L1 /имя1/ L2 ..../имяn/ Ln

где имяi - имя общего блока; Li - список переменных входящих в i-й общий блок. На имя общего блока накладываются те же ограничения, как и на любой

идентификатор. В любом программном модуле может быть один непоименованный общий блок. Тогда символы // для этого блока могут быть опущены.

Пример .

```
Character s(20)*10,s1*8
common a(50) /block1/ s,s1 cb/ x,y,z
```

В этом операторе описаны три общих блока: непоименованный общий блок длиной 200 байт, содержащий массив a; блок с именем block1, занимающий 208 байт, содержащий символьный массив s и символьную переменную s1; блок с именем cb , содержащий три переменные действительного типа.

5.4.2. Оператор эквивалентности. EQUIVALENCE

Указывает, что две и более переменных или массива совместно используют одну и ту же область памяти.

Данный оператор обычно применяется для экономии памяти. В одной и той же области памяти на разных этапах работы программы хранятся различные массивы разных типов и длин. Рассмотрим данную функцию оператора эквивалентности на следующем примере.

```
Real*8 A(8500), b(50,50), c(2000)
Integer K(10000)
Real D(2000)
Equivalence  (A(1),D(1)), (A(1001),K(1)), (A(6001),b(1,1),c(1))
```

В этом примере на месте массива A, занимающего 68Кб, помещены четыре массива: D, K, c и b. Массив D занимает 8000 байт и размещен в тех же ячейках памяти, что и начало массива A. При этом D(1) не равно A(1), т.к. оба элемента в памяти хранятся различным образом. Массив K занимает 40 Кб и хранится внутри массива A начиная с 8001-го байта. Два массива b и c занимают тот же участок памяти, что и массив A, начиная с 48001-го байта. При этом значения элементов A(6001), b(1,1) и c(1) равны между собой, т.к. все они имеют тип и хранятся в памяти в одном и том же месте.

5.4.3. IEEE стандарт хранения чисел в ЭВМ

Как известно, все числа хранятся в памяти ЭВМ в двоичном коде. С появлением персональных ЭВМ был введен стандарт хранения чисел для машин с процессором x086, имеющий название IEEEE. В последнее время большинство ЭВМ перешли на этот стандарт.

5.4.3.1. Целые переменные

В соответствии со стандартом, целые числа в памяти хранятся в двоичной системе счисления, в дополнительном коде:

Первый бит соответствует знаку числа. Он равен 0 для положительного числа и 1 для отрицательного числа. Если число отрицательное, то все биты, кроме первого, инвертируются (заменяются 0 на 1, а 1 на 0) и к полученному числу прибавляется 1.

Пример. Написать программу, которая выводит по битам целую переменную, находящуюся в оперативной памяти.

```
$freeform
```

```
program bitinteg ! Побитовый вывод числа типа integer*2
parameter (size=16)
integer*2 ii
character*16 str
do i=1,size
  str(i:i)='0'
end do
print *,' Введите целое число типа integer*2 '
read *,ii
k=#8000   ! В двоичном виде = 1000 0000 0000 0000
do i=1,size
  ibit=iand(ii,k) ! Побитовое и. Ibit ≠ 0, если i-тый бит числа ii = 1
  if(ibit.ne.0) str(i:i)='1'
  k=ISHL(k,-1)  ! Сдвиг числа k на 1 бит вправо
end do
print '(2x,4(4a1,1x))', (str(i:i),i=1,size)
end
```

В таблице представлены результаты вывода некоторых чисел.

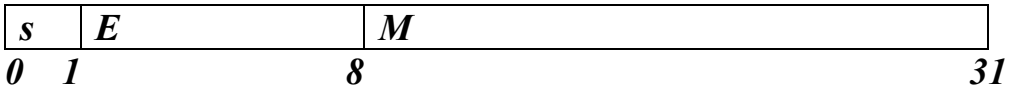
Число	Представление в памяти	Число	Представление в памяти
0	0000 0000 0000 0000	1	0000 0000 0000 0001
32767	0111 1111 1111 1111	-1	1111 1111 1111 1111
-2	1111 1111 1111 1110	-3	1111 1111 1111 1101
-32767	1000 0000 0000 0001	-32768	1000 0000 0000 0000

5.4.3.2. Вещественные переменные

Вещественное число **a** представляются в следующем виде:

Первый бит отводится под знак числа s. s=0 для положительных чисел и s=1 для отрицательных.

Под порядок отводится 8 бит. Порядок хранится в смещенном виде. К порядку прибавляется число 126. Под мантиссу отводится 23 бита. Первый бит мантиссы всегда равен 1, поэтому он не хранится в памяти. Таким образом, длину мантиссы можно считать равной 24. В случае, когда порядок равен 255, код числа равен либо $(-1)^s\infty$, если мантисса равна 0, либо код переполнения (NaN), если мантисса не равна 0. Если порядок равен 0 и мантисса равна 0, то данное число является нулем с плавающей запятой.



Пример. Написать программу, которая выводит по битам вещественную переменную, находящуюся в оперативной памяти.

```
$FREEFORM
program bitfloat
```

```
character*32 str ! в эту переменную выводим побитовое представление числа
equivalence (b,ii)      ! переменная целого типа (ii) и вещественного занимают
                                                    ! одну и ту же память в 4 байта
do i=1,32         ! Всем битам присвоить 0
  str(i:i)='0'
end do
print *,' ,Введите переменную вещественного типа'
read *,b
k=#80000000   ! переменная в шестнадцатеричном формате в двоичном она
! равна 1000 0000 0000 0000 0000 0000 0000 0000
do i=1,32
  ib=iand(ii,k) ! Побитовое и = 0 если i-тый бит переменной b =0
  if(ib.ne.0) str(i:i)='1'
  k=ISHL(k,-1)  ! сдвинуть на 1 бит вправо
end do
" Вывести на монитор двоичный вид действительного числа.
Print '(2x,8(4a1,1x))',(str(i:i),i=1,32)
end
```

Благодаря оператору EQUIVALENCE, действительная переменная b и целая переменная ii занимают одну и ту же область памяти. В программе вводится действительная переменная b, а затем производится обработка переменной ii, как переменной целого типа. Для переменной целого типа есть побитовые функции, которые позволяют определить любой бит. Для примера приведем несколько примеров вывода данной программы.

Действительное число 1. в памяти храниться в виде:

0011 1111 1000 0000 0000 0000 0000 0000.

Если его читать как целое, то получается число #3f800000 = $3*16^7 + 15*16^6 + 8 * 16^5 = 1\ 065\ 353\ 216$.

Согласно стандарту, первый бит определяет знак числа. 0 соответствует знаку +. Последующие 8 бит определяют смещенный порядок $0111\ 1111_2$ = #7f = $7 * 16^1 + 15 = 127$. Чтобы получить действительный порядок, необходимо из смещенного порядка вычесть 126. Все отображаемые биты мантиссы равны 0.

Если перевести 1. в нормализованный двоичный вид, получаем, $1. = 0.1*2^1$. Т.е. мантисса и порядок равны 1. К порядку прибавляется смещение 126, а в мантиссе первый бит не хранится.

Приведем некоторые примеры вещественных чисел.

Чис-ло	Нормализо-ванный вид	Смещен-ный Порядок	Мантисса	Двоичный код
0,5	$0.1 * 2^0$	126=0111 1110	0.1	0011 1111 0000 0000 0000 0000 0000 0000
0,25	$0.1 * 2^{-1}$	125=0111 1101	0.1	0011 1110 1000 0000 0000 0000 0000 0000
-5	$0.101 * 2^3$	129=1000 0001	0.101	1100 0000 1010 0000 0000 0000 0000 0000
127	$0.1111111*2^7$	133=1000 0101	0.1111111	0100 0010 1111 1110 0000 0000 0000 0000

Максимальном положительным вещественным числом является следующее число

0111 1111 0111 1111 1111 1111 1111 1111.

Максимальный порядок равен 254-126 = 128. Максимальная мантисса равна

0,1111 1111 1111 1111 1111 1111 = $1-2^{-24}$. Т.о., максимальное число = $(1-2^{-24})$ $2^{128} \approx 3,4*10^{38}$.

5.4.4. Оператор INCLUDE

Выполняет вставку содержимого заданного файла.

Общий вид:

include’<имя файла>’

где <имя файла> - имя конкретного, уже существующего файла, содержимое которого будет включено в программу.

Данный файл должен находится в текущем каталоге или в каталоге, заданном в переменной окружения MS DOS INCLUDE.

Компилятор рассматривает содержимое включаемого файла как часть программного файла и выполняет его компиляцию немедленно. Включаемые файлы зачастую содержат различные процедуры и функции, объявления «общих» блоков и операторы extern, interface и intrinsic. Включаемые файлы также могут содержать другие операторы include — это называется встраиванием включаемых файлов. Компилятор предоставляет пользователю возможность встраивать любую комбинацию до 10 операторов include.

Этот оператор удобно использовать, когда один и тот же набор функций или данных используется для нескольких программ. Тогда их можно выделить в отдельный файл и подключать этот файл при необходимости.

6. Операторы ввода-вывода

К операторам ввода-вывода относятся операторы, управляющие обменом информацией между оперативной памятью машины и внешними по отношению к памяти устройствами. Обмен этот может идти в большинстве случаев в двух направлениях и даже для одних и тех же устройств может осуществляться в разных формах. В операторах ввода-вывода должно быть указано, что и в какой последовательности вводится или выводится и в какой форме.

Операторы ввода и вывода по своей структуре близки и имеют следующий вид:

read(n,f,ERR=m1,END=m2)<список переменных> - оператор ввода;

read f, <*список переменных*> - оператор ввода со стандартного потока ввода;

write(n, f,ERR=m1,END=m2)<список переменных> - оператор вывода;

print f,<список переменных> - оператор вывод в стандартный поток вывода.

Где n - номер канала ввода или вывода; f - спецификация формата; m1, m2-метки операторов, на которые передается управления при ошибочном и нормальном завершении работы оператора; <список переменных > - вводимые или выводимые переменные. Если в списке несколько переменных, то они разделяются запятыми. Стандартными потоками ввода-вывода являются клавиатура и экран монитора. При помощи простых операций их можно перенаправить. Параметры ERR и END необязательны.

В качестве спецификации формата f может быть метка оператора FORMAT, символьная переменная или символьная константа.

Ранее мы уже использовали бесформатный ввод и вывод, когда в операторах read или write вместо номера устройства ввода вывода и метки оператора format располагаются символы ‘*’(звездочка)

Пример: Бесформатный ввод с клавиатуры числа a и вывод на экран числа на 1 большего.

```
program add0
integer*2 a
print *, ' Введите a '
read *,a
print *,' a+1= ',a+1
end
```

При бесформатном выводе компилятор сам устанавливает количество символов под одно число, количество знаков после запятой и другие данные. Это бывает удобно при малом объеме вывода. Однако при оформлении отчетов необходимо выводить результаты в удобном для обработки виде. Поэтому применяют форматный вывод. Для ввода информации в подавляющем большинстве случаев удобнее бесформатный ввод. При использовании форматного ввода-вывода необходимо задавать формат ввода-вывода. В этом случае программист явно описывает местоположение и форму вводимых или выводимых переменных в записи (строке).

Формат можно задать при помощи оператора FORMAT либо в виде строковой переменной либо в виде символьной переменной.

6.1. Оператор FORMAT

Общий вид оператора:

m FORMAT(c_1,c_2, . . . , c_n)

где m - обязательная метка оператора; c_1,c_2, . . . , c_n - список спецификаций формата.

Для каждого типа данных существует своя спецификация: для целых величин - спецификация I и G; для вещественных величин - F, E и G; для вещественных величин двойной точности - D и G; для логический величин - L и G; для текстовых величин спецификация A.

6.1.1. Спецификация I

Общая форма спецификации I:

Iw

где w - беззнаковая целая константа, показывающая количество символов (длину поля), отводимых для ввода или вывода целых величин. Если элемент занимает меньше места, чем отведено под него, то число печатается в правой части отведенного поля, а левые позиции остаются незаполненными. Если при выводе отведенного места не хватает, то вместо знака и левых значащих цифр будет отпечатан символ *.

Примеры печати по спецификации i4:

число	Будет напечатано
237	237
-25	-25
-1999	*1999
-78953	*953

Если при вводе число занимает больше w символов, то считываются только w левых символов.

Например, запись

-1234567

По спецификации I5 будет считана как -1234.

Таким образом, при задании спецификации необходимо учитывать порядок вводимых или выводимых чисел.

Заметим, что с помощью спецификации i11 можно ввести или вывести любую целую величину, поскольку диапазон изменения целых чисел от (-2147483648) до 2147483647.

6.1.2. Спецификация F

Общая форма спецификации F:

Fw.d

где w - беззнаковая целая константа, показывающая количество символов (длину поля), отводимых для ввода или вывода целых величин, включая знак числа и десятичную точку, а d - число цифр после десятичной точки.

Спецификация f предназначена для вещественных чисел без показателя степени. Во всех случаях w>d для положительных чисел и w>d+1 для отрицательных.

Если при выводе целая часть числа занимает больше w-(d+1) символов, то в отводимом для записи числа поле будут напечатаны *.

Пример: печать чисел по спецификации f6.3

Число	Печать
3.14	3.140
-0.00231	-0.001
981.3	*1.300

6.1.3. Спецификация E

Вещественные числа с порядком представляются с помощью спецификации E, общий вид которой:

Ew.d

где w — общая длина поля, а d — число цифр после десятичной точки в мантиссе, причем w должно быть больше d+6, так как в поле надо записать: знак числа, десятичную точку, мантиссу, символ E, знак порядка и величину порядка.

Пример: в спецификации E11.4 записать числа

Число	Печать
$-0.256\ 10^{-11}$	-0.2560E-11
$2.3\ 10^{2}$	2.3000E 2

В спецификации E могут быть записаны любые вещественные числа, поэтому ее хорошо использовать, особенно при выводе, когда неизвестен порядок величины. При выводе число округляется до значения, которое определяется количеством значащих цифр в мантиссе при заданной спецификации.

6.1.4. Спецификация D

Спецификация вещественных величин двойной точности имеет вид:

Dw.d

где w- общая длина поля, а d- число цифр после десятичной точки в мантиссе. Эта спецификация аналогична спецификации E, и к ней применимы те же правила, с учетом того, что символ D определяет показатель степени.

Пример: печать в D16.9

Число	Печать
-0.256789012	-0.256789012D-11
10^{-11}	
$2.345\ 10^{2}$	0.234500000D 3

6.1.5. Спецификация L

Спецификация логических величин имеет вид:

Lw

где w - длина поля.

При вводе логических величин поле просматривается слева направо, и если первым встречается символ T, то переменной присваивается значение .true. , иначе переменной присваивается значение .false.

При выводе символы T или F располагаются в крайней правой позиции поля, а остальная часть поля заполняется пробелами.

Отметим, что спецификации должны соответствовать типу вводимых или выводимых данных.

6.1.6. Спецификация G

Спецификация G используется для ввода-вывода целых, вещественных и логических переменных. Ее формат

Gw.d

При выводе целых и логических чисел она эквивалентна спецификациям Iw и Lw. При выводе вещественных чисел в зависимости от величины числа и параметров w и d числа выводятся либо в формате Fw.d, либо в формате Ew.d. При этом действует правило: если число не умещается в формате с фиксированной запятой (формате F), то оно печатается в формате с плавающей запятой (формате E).

6.1.7. Спецификация A

Спецификация A предназначена для ввода-вывода текстовых переменных. Формат спецификации Aw или A, где w – количество позиций ввода-вывода. Если w не указано, то предполагается, что данное занимает столько позиций, какова длина соответствующего элемента ввода-вывода. Если w меньше длины данного (l), то данное усекается слева на (l-w) позиций, а если больше, то слева дополняется (w-l) пробелами.

6.1.8. Текстовые константы и управляющие символы

Текстовые константы и H-константы используются для вывода сообщений, заголовков и другой постоянной текстовой информации. Текстовые константы заключаются в кавычки ‘текст’. H-константы имеют вид nHтекст, где n – длина текстовой константы. Длина текстовой константы и H-константы не должны превышать 256 символов.

Спецификация X имеет вид nX и предназначена для пропуска n позиций.

Символы управления переходом на новую строку /, \.

При встрече символа / (слеш) происходит переход к следующей строке, а при встрече символа \ (обратный слеш) происходит запрещение перехода к следующей строке, и следующий вывод производится в конец данной строки.

Пример. Написать программу, которая разбивает строку длиной 255 символов на слова.

```
$freeform
program str
character*255 L
read '(a)',L
print 3
do i=1,255
  if(L(i:i).ne.' '.and.i.lt.255.and.L(i+1:i+1).eq.' ') then
  ″ Найден последний символ в слове
    print 2,L(i:i) ! вывести его и перейти к новой строке
    print 3
```

```
      else if(L(i:i).ne.' ') then
      ″ напечатать символ вначале или внутри слова не переходя на новую строку
         print 1,L(i:i)
       end if
      end do
      1FORMAT(A1,\)
      2FORMAT(A1)
      ″ пропустить первый управляющий символ строки не переходя на новую
"       строку
      3FORMAT(1x,\)
      end
```

6.1.9. Группы спецификаций

Пусть $P_1,...,P_k$ — список переменных в операторе ввода или вывода, а $S_1,...,S_l$ — список спецификаций в соответствующем операторе format, который связан с данной записью (строкой).

Количество символов в строке однозначно определяется списком спецификаций. Каждой спецификации в порядке слева направо отводится соответствующее поле в строке. При этом спецификации пробелов при вводе-выводе и спецификации текста при выводе присутствуют в соответствующих местах строки, без каких либо изменений.

Числовые и логические спецификации из списка $S_1,...,S_l$ в порядке слева направо ставятся с соответствие переменным $P_1,...,P_k$, которые согласно этим спецификациям представляются в строке. Если список переменных меньше, чем количество числовых и логических спецификаций, то ввод-вывод осуществляется на первой строке до конца списка спецификаций, а последующие переменные вводятся или выводятся с новой строки вместе с переходом в начало списка спецификаций, и т.д. до конца списка переменных.

Запись повторяющихся спецификаций и групп спецификаций

Запись

m format(...,S,S,...,S,...)

N раз

эквивалентна записи:

m format(...,NS,...)

А запись

m format(...,S_1,...,S_n,..., ,S_1,...,S_n,...)

N раз

эквивалентна записи

m format(...,N(S_1,...,S_n),...)

При форматном выводе первый символ интерпретируется как управляющий. Если в качестве первого символа стоит символ 0, то происходит переход через две строки; 1 – переход в начало следующей строки; + – не происходит

переход к следующей строке; любой другой символ – переход к следующей строке.

Примеры.

Пример: Бесформатный ввод с клавиатуры числа a и форматный вывод на экран числа на a+1, b=a+2 и c=a+3.

```
program redd1
integer*2 a,b,c
write (*,*) ' Введите целое число a'
read (*,3) a
b=a+2
c=a+3
write (*,4) a+1,b,c
3    format (I2)
4    format (2x,' 1-e Число=',i2,2(' следующее число=',I3))
end
```

6.1.10. Ввод-вывод массивов

При вводе и выводе массивов используется два основных способа - ввод-вывод массива по имени или с помощью неявного «цикла».

1-й способ:

```
program arr1
integer*2 A(10),B(3,4)
read(*,*) A,B
end
```

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

Второй способ (неявный цикл):

```
integer*2 A(10),B(3,4)
read(*,*) (A(I),I=1,10),((B(I,J),J=1,4),I=1,3)
```

Этот способ удобнее, когда в программе используется меньшее количество элементов массивов, нежели то, которое задано в операторах размерности.

Например:

```
integer*2 A(10),B(3,4)
read(*,*) n,l,m,(A(I),I=1,n),((B(I,J),J=1,l),I=1,m)
```

В этом случае числа n, l, m подчиняются ограничениям $n \le 10$, $l \le 4$, $m \le 3$, поскольку в ином случае мы выйдем за границы расположения элементов массивов.

В этом примере сначала вводятся 3 целых числа n, l, m и затем последовательно n элементов массива A, а затем l*m элементов массива B по столбцам, т.е.:

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

В этом случае последовательность ввода элементов B не совпадает с порядком их хранения в памяти. Если записать:

read(*,*) n,l,m,(A(I),I=1,n),((B(I,J),J=1, m),I=1, l)

то порядок ввода элементов массива B, и порядок из размещения в памяти, будет одинаковый.

Вывод массива отличается только заменой оператора read на write.

Так, строка программы:

write(*,*) A,B

последовательно выводит массивы A и B.

Примечание. Неявный цикл используется только в списке ввода-вывода. Общий вид его таков: (имя(p), p=$i_н$, $i_к$, h), где имя — имя массива; p — параметр цикла; $i_н$, $i_к$, h — начальное, конечное значения и шаг параметра цикла. Если h=1, его можно опустить.

6.1.11. Ввод-вывод, управляемый именованным списком. Оператор NAMELIST

Для ввода данных очень удобно использовать именованные списки. При помощи оператора NAMELIST/<имя> / <список переменных>, где <имя> — имя списка, описывается список. Затем при помощи оператора READ(N,<имя>) считываются переменные из списка.

Среди исходных данных должен быть блок

&<имя>

p1=<значение>,p2=<значение>,…, pn=<значение>,

/

Переменные в списке исходных данных могут быть в произвольном порядке. Часть переменных можно не вводить. Тогда их значения при вводе не изменяются. Этим приемом широко пользуются для присвоения переменным значения по умолчанию. При вводе массивов можно применять кратный ввод. Это когда n переменным присваивается одинаковое значение. В этом случае после числа одинаковых переменных ставится символ * (звездочка), а затем значение этих переменных.

Пример. Написать участок программы, который вводит массив **A** размерностью 100. Первые 50 элементов =1, а остальные 0.

```
REAL A(100)
NAMELIST /L1/ A
OPEN(7,FILE='TESTname.DAT')
READ (7,L1)
……..
END
```

Исходные данные для данной программы должны быть следующими:

```
&L1
A=50*1.,50*0.,
/
```

В файле данные должны быть введены со второй позиции, или, более правильно, после первой позиции.

6.2. Ввод-вывод с использованием файлов

Данные часто бывает удобно хранить в файлах, а не вводить их каждый раз с клавиатуры. Можно иметь несколько файлов с данными и считывать данные то из одного, то из другого. Иногда удобно записывать результаты работы программы в файл, чтобы сохранять их там.

Полное описание операторов ввода-вывода в файлы достаточно сложно и объемно. Мы ограничимся только основными операторами и их параметрами.

6.2.1. Открытие и закрытие файлов. Операторы OPEN, CLOSE

Опишем процедуру организации ввода-вывода с использованием файлов. В этом случае файл, с которого будет осуществляться ввод-вывод, нужно связать с каналом ввода-вывода. Для этого используется оператор открытия файла вида

open(n, file = ’<имя файла>’)

где n - номер канала, которым может быть любое целое число, меньше 32767, а имя файла является именем файла, который уже существует (при вводе из файла) или будет создан (при выводе в файл).

Если указанное имя файла - пустое имя (file = ’ ’), то выполняются следующие шаги:

1) Программа пытается считать имя файла из списка имен, указываемых в командной строке (если они там есть), используемой для активизации программы. При исполнении последовательных операторов open выполняется считывание последовательных параметров из командной строки.

2) Если таких операторов open больше, чем параметров в командной строке (или параметры не указаны), то программа выводит просьбу ввести имя файла с клавиатуры, примерно в следующей форме:

File name missing or blank - Please enter name UNIT n?,

где n - номер канала.

Пример. Форматный ввод с клавиатуры числа a и вывод в файл read2.dat числа на 2 большего.

```
      integer*2 a
      write (*,*) ' Введите целое число a'
      read (*,3) a
      open(12, file='read2.dat')
      write (12,*) 'число a=',a
      write (12,4) a+2
3     format (I3)
4     format ( ' a+2=',I3)
end
```

program redd2

Аналогично осуществляется и чтение из файла.

Пример. Форматный ввод из файла read4.dat числа a и вывод в файл read2.dat числа на 2 большего.

```
      program redd1
      integer*2 a
      open(11, file='read2.dat')
      open(12, file='read4.dat')
      read (12,3) a
      write (11,*) 'число a=',a
      write (11,4) a+2
3     format (I3)
4     format ( ' a+2=',I3)
      end
```

Файл может быть отделен от канала ввода-вывода с помощью оператора закрытия файла, который имеет вид

close(n)

где n - номер канала.

Одновременно с закрытием канала файл может быть удален, если использовать оператор:

close(n, status =’delete’)

После закрытия канала его номер становится свободным и может быть использован при открытии нового канала с другим файлом .

В программе может быть любое количество каналов ввода-вывода, которые могут открываться операторами open в любой части программы. Файлы, связанные с каналами ввода-вывода, должны находится в том же каталоге, где помещен запускаемый файл программы.

Иногда бывает удобно не связывать номер канала ввода-вывода с именем конкретного файла, т.е. номер канала n присутствует в операторе read или write, но он не открыт оператором open. Тогда мы получаем сообщение, аналогичное случаю, когда указанное имя файла - пустое имя (file = ’ ’),

File name missing or blank - Please enter name UNIT n?,

где n - номер канала.

После этого вопроса на клавиатуре нужно набрать имя файла, которое вы хотите связать с номером канала n ввода-вывода.

6.2.2. Внутренние файлы

Большинство файлов на языке Фортран являются внешними, т.е. находящимися на внешних устройствах (жестких дисках, лазерных дисках, дискетках и т.д.). Иногда удобнее вывести результаты не на внешнее устройство, а в текстовую переменную. Внутренним файлом является символьная переменная, символьный элемент структуры, символьная подстрока, символьный массив или не символьный массив. Применение внутренних файлов рассмотрим на примере.

Пример. Написать подпрограмму для вывода вещественного числа, которая выводит заглавие переменной до 10 символов, и значение числа, содержащего ровно 7 значащих цифр.

```
$DEBUG  !отладочный режим (при ошибке выводит номер строки, где была
        ! ошибка)
  program test  ! Тестовая программа
  call printreal("Число Пи=",3.14)
  call printreal("Число 1=",0.000000012345671)
  call printreal("Число 2=",0.00000012345671)
  call printreal("Число 3=",0.12345671)
  call printreal("Число 4=",0.012345671)
  call printreal("Число 5=",12345671.)
  call printreal("Число 6=",1234.5671)
  call printreal("Число 7=",1234567.1)
  call printreal("Число 8=",12345671)
  end
  Subroutine printreal(S,a)
  Character S*10, form*6,ftm*13
  Integer d,e
  n=Alog10(abs(a))!Порядок числа
  if(n.gt.0) n=n+1
  If(n.Lt.-7.Or.n.Gt.7) Then
    form=' E15.7'
  Else
    d=9
    if(n.lt.0) d=9-n
    e=7-n
    form='f 9.1 '
    write(form(2:3),100) d   !Вывод в подцепочку ширины поля
    write(form(5:6),100) e   ! Вывод в подцепочку количество знаков после
                             ! запятой
  End If
  ftm='(1x,a,'//form//')' ! Формат вывода
  Print ftm,S,a
  100   format(i2)
  end
```

7. Графическое программирование

Необходимо различать текстовый и графический режимы.

Текстовые режимы разделяют текстовый экран на строки и столбцы.

Если экран поддерживает текстовый режим, отображающий 25 строк и 80 столбцов, то строки нумеруются значениями от 1 до 25 начиная с верхней строки. Столбцы нумеруются от 1 до 80 слева направо. Номер строки всегда указывается прежде номера столбца. Процедуры работы с текстом используют координаты в виде строк и столбцов.

В графических режимах экран разделен на пиксели. Пиксель - точка на экране, имеет атрибут цвета и яркости.

Ниже будет показано, как определяется соответствующий графический режим, как он активизируется и как писать типичные графические программы.

Введем несколько определений:

База-точка с координатами (0,0)-левый верхний угол экрана. В этой точке начинаются оси абсцисс (x) и ординат (y).

Горизонтальное направление представляется осью абсцисс. Координаты меняются слева направо.

Максимальное значение по этой оси - 640 (пиксель).

Вертикальное направление представляется осью ординат. Координаты меняются от 0 до 480 (сверху вниз).

Графические процедуры обеспечивают возможность установки точек, рисования линий и геометрических фигур, изменения цветов и т.п.

В данном разделе описываются основы графического программирования с использованием графической библиотеки исполнительной системы компилятора языка Фортран фирмы Microsoft. Библиотека содержит полный набор графических функций и поддерживает графику, основанную на элементах изображения, координатную графику и символьные шрифты.

Программа пользователя, использующая графические процедуры, компонуется с библиотекой graphics.lib , являющейся составной частью стандартного пакета Фортрана.

В данной главе содержатся сведения об основных категориях библиотечных процедур и их возможностях.

Разработка всех графических программ может быть представлена в виде следующих шагов:

Подключение графической библиотеки.

Установка видеорежима.

Определение параметров видеоконфигурации.

Создание и манипуляция графическими фигурами.

Восстановление первоначальной конфигурации перед выходом из программы.

7.1. Основные функции графического режима

Каждая программа, использующая графическую библиотеку, должна также явно объявлять любые процедуры. Т.е. или написать интерфейс для каждого вызова процедуры, или интерфейс для включаемых файлов fgraph.fi и fgraph.fd.

Эти файлы содержат объявления, необходимые для доступа ко всем библиотечным графическим процедурам. Для использования их следует включить файл fgraph.fi в начало каждого исходного файла программы пользователя, а файл fgraph.fd в каждую подпрограмму, которая вызывает графическую процедуру.

После подключения графической библиотеки следует установить видеорежим, который разрешает выполнение графических процедур.

Прежде чем программа начнет выполнять процедуру рисования графических элементов, надо специфицировать тип используемого монитора, разрешение экрана, решить, будет ли использоваться цветное изображение или черно-белое.

Замечание. В графическом режиме программы пишутся только в фиксированном формате. Это вызвано тем, что подстановочные файлы fgraph.fi и fgraph.fd написаны в фиксированном формате, т.е. Freemode использовать нельзя.

7.1.1. Функции установки графического режима SETVIDEOMODE

Графический режим устанавливается при помощи функции SETVIDEOMODE. Синтаксис этой функции: dummy=setvideomode(mode), где dummy переменная типа integer*2, которая возвращает данная функция. Если dummy=0, то при выполнении функции произошла ошибка. Параметр mode означает тип графического адаптера компьютера.

Нами дальше используется следующие значения параметра mode:

$MAXRESMODE – выбирается режим с максимальным разрешением для данного типа компьютера.

$DEFAULTMODE – убрать графический режим. Используется для возвращения в стандартный текстовый режим. Надо использовать при завершении работы в графическом режиме

Приведем простейшую программу, работающую в графическом режиме. Для этого рассмотрим еще одну достаточно важную графическую функцию **setpixel,** которая рисуют всего лишь одну точку на экране в указанном месте.

7.1.2. Изображение точки на экране setpixel

Синтаксис : dummy=setpixel(x,y)

x,y - координата точки.

Пример. Нарисовать прямую горизонтальную линию посредине экрана.

```
INCLUDE  'FGRAPH.FI'
program diagonal
INCLUDE  'FGRAPH.FD'
dummy=setvideomode( $MAXRESMODE )     ! инициализировать
                                                !графический режим
do i=0,639
  dymmy=setpixel(i,240)    !поставить точку (пиксель) на экране (i,240)
                                !коорд. точки
end do
read(*,*)! ждать пока нажмем любую клавишу
dummy=setvideomode($DEFAULTMODE) ! убрать графический режим
end
```

В данном примере рисуются одна: прямая горизонтальная линия. Цикл выполняется 640 раз. Внутри цикла ставится одна точка. Набор из 640 точек образуют прямую линию.

7.1.3. Установка цвета пикселя

Синтаксис dummy=setcolor(color)

Функция setcolor устанавливает текущий цвет символов в соответствии с параметром color.

Функция setcolor принимает значение integer*2 в качестве параметра, являющегося индексом цвета.

Индекс цвета, принимаемый по умолчанию, - наивысший пронумерованный индекс цвета в текущей палитре.

Пример: Нарисовать синусоиду и ось x разными цветами.

```
INCLUDE  'FGRAPH.FI'
program grsin
INCLUDE  'FGRAPH.FD'
dummy=setvideomode( $MAXRESMODE )
do i=0,628
dummy=setcolor(i/40+1)
dummy=setpixel(i,240)
dummy=setpixel(i,240-int2(240.*sin(i/100.)))
end do
read(*,*)!
dummy=setvideomode($DEFAULTMODE) !
end
```

7.1.4. Получение конфигурации графической среды. GETVIDEOCONFIG

Процедура getvideoconfig возвращает текущую конфигурацию графической среды в виде структуры videoconfig, которая определена в файле fgraph.fd .

Эту процедуру следует вызывать перед вводом видеорежима для предотвращения возможных проблем несовместимости.

Ее вызов имеет вид:

Call getvideoconfig(video)

Полезно знать, какие показатели можно определить с ее помощью, поэтому ниже приведена структура videoconfig.

Структура videoconfig имеет следующий вид:

```
STRUCTURE/ videoconfig/
  INTEGER2* numxpixels !Число пиксель по оси x
  INTEGER2* numypixels ! Число пиксель по оси y
  INTEGER2* numtextcols !Допустимое число столбцов
  INTEGER2* numtextrows ! Допустимое число строк
  INTEGER2* numcolors !Число индексов цвета
  INTEGER2* bitsperpixel !Число разрядов на пиксель
  INTEGER2* numvideopage !Число видеостраниц
  INTEGER2* mode !Текущий видеорежим
  INTEGER2* adapter !Активный видеоадаптер
  INTEGER2* monitor !Активный дисплейный монитор
  INTEGER2* memory !Память видеоадаптера в Кбайтах
END  STRUCTURE
```

Поля структурной переменной типа videoconfig инициализируются при выходе из процедуры getvideoconfig.

В любом текстовом режиме, включая монохромный, функция getvideoconfig возвращает 32 значения для заданного количества допустимых цветов, где цвета со значениями от 0 до 15 интерпретируются как нормальные, а от16 до 31 - как мерцающие цвета (с номерами от 0 до 15).

Пример: Написать программу, которая выводит число точек на мониторе.

```
      INCLUDE  'FGRAPH.FI'
      program vid_conf
      INCLUDE  'FGRAPH.FD'
      RECORD /videoconfig/ vc
      IF( setvideomode( $MAXRESMODE ) .EQ. 0 )
     +   STOP 'Error:  не могу определить графическую моду монитора'
      CALL getvideoconfig( vc )
c     определение числа точек на мониторе
      maxx = vc.numxpixels-1
      maxy = vc.numypixels-1
      dummy=setvideomode($DEFAULTMODE) ! убрать графический режим
```

```
write(*,*) ' Число точек на мониторе'
write(*,*) 'По оси x',maxx
write(*,*) 'По оси y',maxy
end
```

7.2. Изображение графических элементов

Процедуры графической библиотеки компилятора языка Фортран позволяют рисовать геометрические фигуры, используя задаваемую пользователем систему координат. Они используют заданные атрибуты: стиль изображения линий, маску заполнения цветом, цвет фона и цвет изображения.

Фигуры центрируются и ограничиваются по размерам при помощи ограничительного прямоугольника, который задается верхним левым и правым нижним углами. Центр прямоугольника (место пересечения диагоналей) становится центром фигуры.

Ниже приведенные процедуры помогают нарисовать достаточно сложные фигуры.

7.2.1. Рисование эллиптических дуг. ARC

Dummy=arc(x1,y2,x2,y2,x3,y3,x4,y4)

Процедуры arc выполняют рисование эллиптических дуг.

Центр дуги есть центр ограничивающего прямоугольника, определяемого точками (x1,y1) для левого верхнего угла и (x2,y2) для правого нижнего. Дуга начинается в точке ее пересечения с вектором, определяемым точкой (x3,y3), и заканчивается в точке пересечения с вектором, определяемым точкой (x4,y4).

Дуга рисуется с использованием текущего цвета. Поскольку дуга не определяет замкнутую область, то она не заполняется цветом.

Пример: программа, которая рисует текущим цветом дугу на экране.

```
INCLUDE   'FGRAPH.FI'
program arrc
INCLUDE   'FGRAPH.FD'
INTEGER*2         status
status = setvideomode( $MRES16COLOR )
status = arc( 80, 50, 240, 150, 80, 50, 240, 150 )
READ( *, * )     ! Нажать ENTER для выхода из графического режима
status = setvideomode( $DEFAULTMODE )
END
```

7.2.2. Рисование линии. LINETO

Dummy= lineto(x,y)

Функция lineto рисует линию из текущей позиции до заданной точки (x,y). Линия рисуется с использованием текущего цвета и стиля изображения. При отсутствии ошибок функция lineto устанавливает текущую позицию в точку (x,y) .

7.2.3. Изменение текущей позиции. MOVETO

Call moveto(x1,y1,xy)

Функция moveto выполняет перемещение без рисования из текущей позиции в заданную точку (x1,y1).

Функция moveto возвращает координаты предыдущей позиции в виде структуры xycoord, определенной в файле fgraph.fd.

Функция moveto часто используется в паре с функцией lineto.

Пример: Подпрограмма, которая рисует линию из точки с координатами (x1,y1) до точки с координатами (x2,y2).

```
subroutine line(x1,y1,x2,y2)
INCLUDE  'FGRAPH.FD'
RECORD /xycoord/ xy
integer*4 x1,y1,x2,y2
call moveto(int2(x1),int2(y1),xy)
dummy=lineto(int2(x2),int2(y2))
end
```

7.2.4. Рисование эллипса. ELLIPSE

Dummy= ellipse(control, x1,y2,x2,y2)

Функция ellipse выполняет процедуру рисования эллипса.

Ограничивающая линия отображается текущим цветом. Центр эллипса - это центр ограничивающего прямоугольника, определяемого точками (x1,y2) для левого верхнего угла и (x2,y2) для правого нижнего.

Параметр control может быть одной из символьных констант:

$GFILLINTERIOR - Заполнение эллипса текущим цветом с использованием текущей маски заполнения.

$GBORDER - Рисование только контура эллипса.

7.2.5. Рисование прямоугольника. RECTANGLE

Dummy= rectangle(control,x1,y1,x2,y2)

Функция rectangle выполняет рисование прямоугольника определяемого точками (x1,y1) - для левого верхнего угла и (x2,y2) - для правого нижнего.

Параметр control аналогичен описанному ранее для процедуры ellipse.

Прямоугольник может быть окрашен с использованием текущего цвета и текущей маски заполнения. В случае применения процедуры floodfill для заполнения цветом прямоугольника он должен быть нарисован с использованием сплошной линии.

7.2.6. Очистка экрана. CLEARSCREEN

Call clearscreen(area)

Процедура clearscreen очищает экран, заполняя ее цветом фона (текущим).

Параметр area может быть одной из следующих символических констант (определенных в файле fgraph.fd):

$GCLEARSCREEN - Очистка и заполнение цветом всего экрана.

$GVIEWPORT - Очистка только текущей области просмотра.

$GWINDOW - Очистка только текущего текстового окна.

Пример: Рисуется прямоугольник, потом экран очищается и другим цветом рисуется эллипс.

```
      INCLUDE  'FGRAPH.FI'
      program clear
      INCLUDE  'FGRAPH.FD'
      RECORD /videoconfig/ vc
      IF( setvideomode( $MAXRESMODE ) .EQ. 0 )
     +   STOP 'Error:  не могу определить графическую моду монитора '
      CALL getvideoconfig( vc )
      dummy = setcolor(3)
      dummy = rectangle( $GFILLINTERIOR, 10, 10, 200, 100)
      read(*,*)  !Ждать нажатия ENTER
      call clearscreen($GCEARSCREEN)
      dummy = setcolor(4)
      dummy = ellipse( $GFILLINTERIOR,10 ,10, 200,100)
      read(*,*)
      call clearscreen($GCEARSCREEN)
      end
```

7.2.7. Рисование многоугольника. POLYGON

dummy= polygon(control, lppoints,cpoints)

Функция выполняет рисование многоугольника, координаты точек которого заданы в структуре lppointds.

RECORD /xycoord/ lppoints(*)

Параметр cpoints задает число точек многоугольника.

Параметр control аналогичен описанному ранее для процедуры ellipse.

Пример: Нарисуем на экране 16 звездочек и раскрасим их разными цветами.

```
      INCLUDE  'FGRAPH.FI'
      program star
      INCLUDE  'FGRAPH.FD'
      integer*2 dummy,height,width,yc,xc,side
      RECORD /videoconfig/ vc
      RECORD / xycoord /     poly  (5)
```

```
      IF( setvideomode( $MAXRESMODE ) .EQ. 0 )
     +   STOP 'Error '
      CALL getvideoconfig( vc )
      height=50
      width=50
      r = width
      pi=3.14159
      rad = 144 * pi / 180
      k=0
      DO i = 1, 4
       xc=i*640/4-640/8
       DO j = 1, 4
        yc=j*480/4-480/8
        DO side = 1, 5
         poly(side).xcoord = xc+INT2( COS( side*rad ) *r )
         poly(side).ycoord = yc+INT2( SIN( side*rad ) *r )
        END DO
        k=k+1
        dummy = setcolor( INT4( k ) )
        status = polygon( $GFILLINTERIOR, poly, 5 )
        dummy = setcolor( INT4( k+1 ) )
        dummy = floodfill(xc,yc,k )
       END DO
      END DO
      read(*,*)
      dummy=setvideomode($DEFAULTMODE)
      end
```

7.2.8. Рисование сектора эллипса. PIE

Dummy=pie(control, x1, y1, x2, y2, x3, y3, x4, y4)

Процедура pie выполняет рисование выреза фигуры эллипса.

Центр эллиптической дуги есть центр ограничивающего прямоугольника, определяемого точками (x1,y1) для левого верхнего угла и (x2,y2) для правого нижнего. Дуга начинается в точке ее пересечения с вектором, определяемым точкой (x3,y3), и заканчивается в точке пересечения с вектором, определяемым точкой (x4,y4).

Вырез рисуется с использованием текущего цвета.

Параметр control может быть одной из символьных констант:

$GFILLINTERIOR - заполнение выреза текущим цветом с использованием текущей маски заполнения.

$GBORDER - рисование только контура выреза.

Пример: Рисуется контур выреза.

```
      INCLUDE   'FGRAPH.FI'
      program pie1
```

```
INCLUDE   'FGRAPH.FD'
INTEGER*2         status
status = setvideomode( $MAXRESMODE )
status = pie( $gborder,80, 50, 240, 150, 80, 50, 240, 150 )
READ( *, * )     ! Press ENTER to exit
status = setvideomode( $DEFAULTMODE )
END
```

7.2.9. Закрашивание замкнутой области. FLOODFILL

Dummy= floodfill(x,y,boundary)

Функции заполняют цветом область, используя текущий цвет и маску заполнения. Функция floodfill начинает заполнение цветом в точке с координатами (x,y).

Если эта точка лежит внутри фигуры (круга эллипса, квадрата, многоугольника, замкнутой ломанной и т.д.), то внутренняя область заполняется цветом, иначе заданным цветом окрашивается фон. В случае применения процедуры floodfill для заполнения цветом замкнутой фигуры, нарисованной при помощи функции lineto, фигура должна быть нарисована с использованием шаблона «сплошной линии».

Точка должна находится внутри или вне заполняемой цветом фигуры, но не на ее границе. «Закрашивание» выполняется во всех направлениях, заканчиваясь на цвете границы, задаваемом параметром boundary.

7.3. Установка атрибутов фигур

Процедуры вывода, выполняющие операции рисования графических образов, не специфицируют данные о цвете или стиле изображения линий. Эти данные задаются при помощи следующих процедур.

7.3.1. Установка цвета фона: SETBKCOLOR

Dummy=setbkcolor(color)

Функция setbkcolor возвращает текущий индекс цвета фона. По умолчанию принимается значение 0.

Функция setbkcolor устанавливает текущий цвет фона в соответствии со значением параметра цвета color.

Значение цвета фона задается символическими константами, определяемыми во включаемом файле fgraph.fd.

Например, setbkcolor($green) устанавливает цвет фона в графическом режиме в значение «зеленый». Указанные символьные константы реализованы для обеспечения удобства при определении и обработке наиболее часто используемых цветов.

В текстовом режиме функция setbkcolor не влияет на уже имеющееся на экране изображение; она воздействует только на последующий вывод. В графическом режиме элементы изображения фона будут немедленно изменятся.

Пример: Рисуется прямоугольник, потом экран очищается и другим цветом рисуется эллипс. Сначала цвет фона устанавливается в голубой, а затем после очистки экрана — в красный.

```
      INCLUDE  'FGRAPH.FI'
      program cler
      INCLUDE  'FGRAPH.FD'
      RECORD /videoconfig/ vc
      IF( setvideomode( $MAXRESMODE ) .EQ. 0 )
     +   STOP 'Error:  не могу определить графическую моду монитора'
      CALL getvideoconfig( vc )
      dummy = setcolor(3)
      dummy = setbkcolor($blue)
      dummy = rectangle( $GFILLINTERIOR, 10, 10, 200, 100)
      read(*,*)
      call clearscreen($GCEARSCREEN)
      dummy = setbkcolor($red)
      dummy = setcolor(7)
      dummy = ellipse( $GFILLINTERIOR,10 ,10, 200,100)
      read(*,*)
      call clearscreen($GCEARSCREEN)
      end
```

Вообще названия цветов во многом совпадают с их английскими названиями.

Вот список цветов:

Темные цвета:

$BLACK -черный
$BLUE - голубой
$GREEN - зеленый
$CYAN - циановый
$RED - красный
$MAGENTA - фиолетовый
$BROWN- коричневый
$WHITE - белый
$GRAY - серый

Соответствующие светлые цвета:

$LIGHTBLUE
$LIGHTGREEN
$LIGHTCYAN
$LIGHTRED
$LIGHTMAGENTA
$YELLOW - желтый
$BRIGHTWHITE

7.3.2. Установка шаблона для рисования линии. SETLINESTYLE

Call setlinestyle(mask)

Тип или способ изображения линии (сплошная, пунктир и т.д.).

Процедура setlinestyle осуществляет выбор маски (стиля изображения линий), используемой для рисования линий .

Маска представляет собой 16-разрядное число, в котором каждый бит представляет элемент изображения в рисуемой линии. Если разряд установлен в значение 1, то соответствующий элемент изображения устанавливается в цвет линии (текущий цвет). Если бит равен 0, элемент изображения остается без изменений. Маска действительна по всей длине линии. По умолчанию принимается маска #FFFF - сплошная линия.

Примеры шаблонов рисования линий:

#FFFF – сплошная линия (двоичный вид этого числа 1111 1111 1111 1111)

#FFOO – длинные штрихи (1111 1111 0000 0000)

#FOFO – короткие штрихи (1111 0000 1111 0000)

Вообще возможен выбор любой комбинации 1 и 0. Шестнадцатиразрядное двоичное число также может быть представлено как 16-разрядное или десятичное число.

Пример: шестнадцатеричная константа #AA3C эквивалентна двоичному представлению 1010 1010 0011 1100. Однако возможно использовать десятичное значение 48580. Т.к. $\#AA3C=10\cdot16^3+10\cdot16^2+3\cdot16^1+12\cdot16^0=48580$.

Пример: Изображение прямоугольника пунктирными линиями.

```
      INCLUDE  'FGRAPH.FI'
      program cler
      INCLUDE  'FGRAPH.FD'
      RECORD /videoconfig/ vc
      IF( setvideomode( $MAXRESMODE ) .EQ. 0 )
     +   STOP 'Error:  не могу определить графическую моду монитора'
      CALL getvideoconfig( vc )
      dummy = setcolor(2)
      call setlinestyle(#f0f0)
      dummy = rectangle( $Gborder, 10, 10, 200, 100)
      read(*,*)
      end
```

7.3.3. Установка маски заполнения. SETFILLMASK

Call setfillmask(mask)

Процесс раскрашивания графических фигур контролируется при помощи текущей маски заполнения цветом.

Процедура setfillmask устанавливает текущую маску заполнения цветом.

Маска представляет собой массив размером 8x8, где каждый разряд (бит) представляет элемент изображения (пиксель).

Установка бита в значение 1 окрашивает пиксель в текущий цвет, а значение 0 оставляет пиксель без изменения. Маска повторяется для всей закрашиваемой области. Если маска заполнения не установлена или параметр mask в качестве всех значений имеет нули, то при операциях заполнения используется только текущий цвет.

Если маска не установлена, то используется только текущий цвет.

Пример: 4 раза перерисовывается прямоугольник с использованием различных масок заполнения.

```
      INCLUDE  'FGRAPH.FI'
      program cler
      INCLUDE  'FGRAPH.FD'
      INTEGER*1           fill(8)
      RECORD /videoconfig/ vc
      IF( setvideomode( $MAXRESMODE ) .EQ. 0 )
     +   STOP 'Error:  не могу определить графическую моду монитора'
      CALL getvideoconfig( vc )
      dummy = setcolor(2)
      k=0
      do i=0,3
       do j=1,9!Создание маски
         k=k+1
         fill(j)=k+1
       end do
       CALL setfillmask( fill )
       dummy = rectangle( $Gfillinterior, 10, 10, 200, 100)
       read(*,*)
       call clearscreen($GCEARSCREEN)
      end do
      end
```

7.4. Функции отображения текста

Следующие процедуры обеспечивают вывод на экран текста как в графическом, так и в текстовом режимах.

В отличие от стандартных операторов ввода-вывода перечисленные процедуры распознают границы окна вывода текста там, где реализованы окна.

7.4.1. Вывод текста на экран. OUTTEXT

Call outtext(text)

Процедура outtext отображает строку текста text на экран. Эта процедура работает в любом экранном режиме.

Текст выводится начиная с текущей позиции курсора. Возвращаемое значение отсутствует.

Пример. Рисуется и подписывается прямоугольник, затем экран очищается и рисуется и подписывается эллипс.

```
      INCLUDE  'FGRAPH.FI'
      program ttt1
      INCLUDE  'FGRAPH.FD'
      RECORD /videoconfig/ vc
      RECORD / rccoord / curpos
      IF( setvideomode( $MAXRESMODE ) .EQ. 0 )
     +   STOP 'Error:  не могу определить графическую моду монитора'
      CALL getvideoconfig( vc )
      dummy = rectangle( $GFILLINTERIOR, 50, 50, 300, 200)
c     Вывод текста в текущей позиции
      CALL outtext( 'Прямоугольник' )
      read(*,*)
      call clearscreen($GCEARSCREEN)
      dummy = ellipse( $GFILLINTERIOR,10 ,10, 200,100)
      CALL outtext( 'Эллипс' )
      read(*,*)
      call clearscreen($GCEARSCREEN)
      dummy=setvideomode($DEFAULTMODE) ! убрать графический режим
      end
```

7.4.2. Получение текущей позиции вывода текста. GETTEXTPOSITION. Установка текущей позиции вывода текста. SETTEXTPOSITION

Oldposition=gettextposition()

Call settextposition(row,column,rccoord)

Функция gettextposition возвращает текущую позицию вывода текста в виде структуры rccoord, определенной в файле fgraph.fd.

Текстовая позиция с координатами (1,1) определяется как верхний левый угол текстового окна.

Функция settextposition выполняет переразмещение текущей позиции текста, *выводимого при помощи outtext,* для отображения символа в точке с координатами (row,column).

Предыдущая текстовая позиция возвращается в виде структуры rccoord, определенной в файле fgraph.fd.

При использовании write одновременно с графическими процедурами settextposition и outtext возможны конфликты.

Пример: Аналогичен примеру 1, но подпись «прямоугольник» пишется на первой строке, со второй позиции, а подпись «эллипс» пишется на 20-й строке с 5-й позиции.

```
      INCLUDE  'FGRAPH.FI'
      program ttt2
      INCLUDE  'FGRAPH.FD'
      RECORD /videoconfig/ vc
      RECORD / rccoord / curpos
      IF( setvideomode( $MAXRESMODE ) .EQ. 0 )
     +   STOP 'Error:  не могу определить графическую моду монитора'
      CALL getvideoconfig( vc )
      dummy = rectangle( $GFILLINTERIOR, 50, 50, 300, 200)
      CALL settextposition( 1,2, curpos )
      CALL outtext( 'Прямоугольник' )
      read(*,*)
      call clearscreen($GCEARSCREEN)
      dummy = ellipse( $GFILLINTERIOR,10 ,10, 200,100)
      CALL settextposition( 20,5, curpos )
      CALL outtext( 'Эллипс' )
      read(*,*)
      call clearscreen($GCEARSCREEN)
      dummy=setvideomode($DEFAULTMODE) ! убрать графический режим
      end
```

7.4.3. Получение индекса текущего цвета текста. GETTEXTCOLOR. Установка цвета вывода текста. SETTEXTCOLOR

Oldfgr=gettextcolor()

Функция gettextcolor возвращает индекс текущего цвета.

По умолчанию принимается наивысший явный индекс цвета текущей палитры. Возвращается индекс текущего цвета текста.

Функция settextcolor устанавливает текущий цвет текста.

Dummy=settextcolor(index)

где index - новый индекс текста (по умолчанию он соответствует максимальному индексу цвета).В цветном режиме index$\in$[0:31], где цвета со значениями от 0 до 15 интерпретируются как нормальные, а от16 до 31 - как мерцающие цвета (с номерами от 0 до 15).

Функция settextcolor устанавливает текущий цвет текста, *выводимого при помощи outtext*. Она не оказывает воздействие на цвет текста, выводимого оператором write или процедурой outgtext.

Пример: Аналогичен предыдущему примеру, но подпись «прямоугольник» и подпись «эллипс» пишутся разными цветами.

```
      INCLUDE  'FGRAPH.FI'
      program ttt1
      INCLUDE  'FGRAPH.FD'
```

```
      RECORD /videoconfig/ vc
      RECORD / rccoord / curpos
      IF( setvideomode( $MAXRESMODE ) .EQ. 0 )
     +   STOP 'Error:  не могу определить графическую моду монитора'
      CALL getvideoconfig( vc )
      dummy = rectangle( $GFILLINTERIOR, 50, 50, 300, 200)
      CALL settextposition( 1,2, curpos )
      dummy=settextcolor(10)
      CALL outtext( 'Прямоугольник' )
      read(*,*)
      call clearscreen($GCEARSCREEN)
      dummy = ellipse( $GFILLINTERIOR,10 ,10, 200,100)
      CALL settextposition( 20,5, curpos )
      dummy=settextcolor(12)
      CALL outtext( 'Эллипс' )
      read(*,*)
      call clearscreen($GCEARSCREEN)
      dummy=setvideomode($DEFAULTMODE) ! убрать графический ре-
      жим
      end
```

Перечисленные процедуры не обеспечивают возможностей форматирования текста. Если пользователю необходимо распечатать целые значения или значения с плавающей точкой, он должен выполнить преобразование значений в строку (используя внутренний оператор write) перед тем, как будет вызвана одна из указанных процедур. Текстовые процедуры специфицируют все позиции на экране в символьных координатах строк и столбцов.

7.5. Шрифты и текст

Шрифт представляет собой стилизованные символы текста конкретного размера. Графическая библиотека компилятора языка Фортран поддерживает множество разнообразных шрифтов, что обеспечивает возможность более гибкого контроля и управления размещением текста, нежели оператор write.

Символы могут быть нарисованы одним из двух способов: как «битовые» (так называемая «картинка» буквы) или как «векторные» символы (последовательность линий и дуг, образующих букву).

Битовые шрифты не могут быть масштабированы, (т.е. уменьшены или увеличены с произвольным коэффициентом масштаба). Векторные шрифты могут быть масштабированы.

В графической библиотеке компилятора языка Фортран реализовано 6 основных шрифтов.

Шрифты графической библиотеки.

Таблица 7.1.

Тип шрифта	Отображение	Размеры шрифта	Разрядка
Courier	Битовый	10x8, 12x9, 15x12	Фиксированная
Helv	Битовый	10x5, 12x7, 15x8, 8x9, 22x12, 28x16	Пропорциональная
Tms Rmn	Битовый	10x5, 12x6, 15x8, 16x9,29x12, 26x16	Пропорциональная
Modern	Векторный	Масштабируемый	Пропорциональная
Script	Векторный	Масштабируемый	Пропорциональная
Roman	Векторный	Масштабируемый	Пропорциональная

***Замечание**:* Размер битовых шрифтов задается в размерах элементов изображения. Точный размер любого символа шрифта зависит от разрешения экрана и типа дисплея.

Данные для шрифтов содержатся в файлах с расширениями .fon, Имя файла указывает на его содержание. Файлы .fon тождественны файлам .fnt, используемым в операционной среде Microsoft Windows. Если пользователь работает в указанной среде, то он может использовать любой из файлов с расширением .fnt для работы с функциями шрифтов компилятора Фортран.

Использование шрифтов в графической программе во многом похоже на использование графики. Там, для рисования фигуры сначала надо установить графическую моду, цвет фигуры, фон, маску, и т.д., потом после рисования графический режим отключается. Так и при использовании шрифтов в графической программе необходимо:

Выбрать шрифт (см. registerfonts)

Установить текущий шрифт из регистра (см. setfont)

Отобразить текст, используя текущий установленный шрифт (см. outgtext).

Затем, после использования, отключить шрифты (см. unregisterfonts)

Все эти функции и некоторые другие будут приведены ниже.

7.5.1. Регистрация доступных шрифтов. REGISTERFONTS

Dummy=registerfonts(filename)

Шрифты, которые будут использоваться в программе, должны быть размещены в памяти. Этот процесс называется регистрацией шрифтов. Регистрационный список предоставляет информацию о доступных .fon - файлах. Операция реализуется процедурой registerfonts.

Функция registerfonts выполняет инициализацию шрифтов графической библиотеки.

Параметр filename - это спецификация пути и имя файла с расширением .fon.

Файлы шрифтов должны быть зарегистрированы до того, как будет использоваться любая другая библиотечная функция работы со шрифтами.

В случае успеха возвращается положительное значение, в противном случае возвращается отрицательное значение.

7.5.2. Установка текущего шрифта. SETFONT

Dummy=setfont(options)

Функция setfont выполняет поиск единственного шрифта из набора зарегистрированных, имеющего характеристики, заданные строкой options.

Параметры функции setfont.

Таблица 7.2.

	Значение
Параметр	Имя шрифта (Одно из следующих: courier, helv, tms, rmn, modern, script, roman).
Ну	Высота символа. При этом у есть число, определяющее значение высоты в элементах изображения (пикселях).
Wx	Ширина символа. Где x - число пикселей.
F	Выбор шрифта с Шрифт с фиксированной разрядкой.
P	Выбор шрифта с пропорциональной разрядкой.
V	Выбор только векторного шрифта.
R	Выбор только битового шрифта.
B	Выбор наиболее подходящего шрифта, который в большей мере соответствует заданным пользователем характеристикам, при этом используется следующая последовательность критериев (в порядке приоритета): 1) высота символов шрифта; 2) начертание шрифта; 3)ширина символов шрифта;4) разрядка шрифта.
Nx	Выбор шрифта с номером x, где x номер зарегистрированного процедурой registerfonts шрифта.

Если шрифт найден, он устанавливается как текущий шрифт.

Текущий шрифт используется во всех последовательных вызовах функции outgtext. *Активным может быть только один шрифт.*

Параметры функции задаются строкой символов, которые определяют желаемые характеристики шрифта. Порядок перечисления символов в строке параметров произвольный, возможны как прописные, так и строчные символы, которые могут разделятся пробелами.

Пользователь может задать любое количество параметров, исключая параметр nx, который следует использовать как единственный параметр.

При задании взаимно исключающих параметров (например f/p или r/v) функция setfont игнорирует их.

Если задан параметр nx, то функция будет игнорировать любые другие задаваемые параметры и поддерживать только шрифт, номер которого соответствует x.

Функции работы со шрифтами оказывают воздействие только на вывод с использованием процедуры outgtext.

При успешном завершении возвращается значение 0, иначе — 1.

7.5.3. Вывод текста с использованием шрифта. OUTGTEXT

Call outgtext(text)

Процедура outgtext отображает строку текста text на экран с использованием текущего шрифта в текущей графической позиции и текущим цветом.

Форматирование текста не выполняется. После вывода текста процедура outgtext изменяет текущую графическую позицию.

Данная процедура выполняется только в графическом режиме. Цвет текста устанавливается функцией setcolor.

7.5.4. Отключение шрифтов. UNREGISTERFONTS

Call unregisterfonts()

Процедура unregisterfonts освобождает память, ранее выделенную с использованием процедуры registerfonts.

Любая попытка использовать функции setfont или outgtext после вызова функции unregisterfonts вызывает ошибку.

Приведем теперь пример использования этих функций.

Пример: Вывести слова Text на экране 6-ю разными шрифтами.

```
      include  'fgraph.fi'
      program ffonts
      include  'fgraph.fd'
      parameter          ( nfonts = 6 )
      RECORD /xycoord/xy
      character*10       options(nfonts)
      character*20       list
      DATA options / "t'courier'", "t'helv'"    , "t'tms rmn'" ,
     +                    "t'modern'" , "t'script'"  , "t'roman'"   /
c     Выполняется регистрация шрифтов  (при этом файл со шрифтом
c      должен находится в одной  директории с программой)
      if( registerfonts( '*.FON' ). lt. 0 )
     +     stop 'Error: cannot find font files'
c      Установка видео режима
      IF( setvideomode( $MAXRESMODE ) .EQ. 0 )
     +    STOP 'Error:  Не могу установить графический режим'
c     Вывод слова Text на экране 6-ю разными шрифтами
      call clearscreen( $GCLEARSCREEN )
      DO ifont = 1, NFONTS
        list = options(ifont) // 'h30w24b'
c     Устанавливается один из 6-ти стандартных шрифтов,
c     заданных в массиве options с высотой символа - 30 пикселей (h30)
c     шириной - 24 (w24), с выбором наиболее подходящего шрифта (пара-
c     метр b)
```

```
      IF( setfont( list ) .GE. 0 ) THEN
        CALL moveto(10, ifont*35, xy )
      CALL outgtext("Text")
      READ (*,*)
      END IF
      END DO
c     Отключение шрифтов и освобождение отведенной под них памяти
      CALL unregisterfonts()
      dummy = setvideomode( $DEFAULTMODE )
      END
```

В цикле устанавливается один из 6-ти стандартных шрифтов, заданных в массиве options, с его использованием пишется слово Text, затем выбирается другой шрифт и т.п.

***Замечание**:* Обратите внимание, что текущая позиция изменяется с помощью функции moveto.

Приведем еще несколько функций для работы со шрифтами:

7.5.5. Получение информации о шрифте. GETFONTINFO

Dummy=getfontinfo(fi)

Функция getfontinfo устанавливает характеристики текущего шрифта в структуре fontinfo, определенной в файле fgraph.fd.

Структура fontinfo имеет вид:

```
STRUCTURE/ fontinfo/
  INTEGER*2 type !Векторный (1) или битовый (0)
  INTEGER*2 ascent !Расстояние от верха до базы
  INTEGER*2 pixwidth !Ширина символа в пикселях
  INTEGER*2 pixheight !Высота символа в пикселях
  INTEGER*2 avgwidth !Средняя ширина символа
  CHARACTER*81 filename !Имя файла шрифта
  CHARACTER*32 facename !Имя шрифта
END STRUCTURE
```

Во многом эта функция похожа на функцию getvideoconfig.

7.5.6. Получение ширины заданного текста. GETGTEXTEXTENT

Dummy=getgtextextent(text)

Функция getgtextextent возвращает значение ширины поля в пикселях, требуемое для вывода строки text текущим шрифтом с использованием процедуры outgtext.

Использование этой функции целесообразно при использовании шрифтов с пропорциональной разрядкой.

Возвращается значение ширины строки текста в пикселях.

Пример: Различными шрифтами печатаются их названия.

Функции getfontinfo и getgtextextent используются для размещения надписи в центре экрана.

```
      include  'fgraph.fi'
```

```
      program ffonts
      include  'fgraph.fd'
      parameter          ( nfonts = 6 )
      integer*2          dummy, x, y
      integer*4          ifont
      character*11       face(nfonts)
      character*10       options(nfonts)
      character*20       list
      RECORD /videoconfig/ vc
      RECORD /xycoord/     xy
      RECORD /fontinfo/    fi
      DATA face    / "Courier"   , "Helvetica"  , "Times Roman",
     +               "Modern"    , "Script"     , "Roman"      /
      DATA options / "t'courier'", "t'helv'"    , "t'tms rmn'" ,
     +               "t'modern'" , "t'script'"  , "t'roman'"   /
      call clearscreen( $GCLEARSCREEN )
      if( registerfonts( '*.FON' ). lt. 0 )
          stop 'Error: cannot find font files'
      IF( setvideomode( $MAXRESMODE ) .EQ. 0 )
        STOP 'Error:  Не могу установить графическую моду'
      CALL getvideoconfig( vc )
      DO ifont = 1, NFONTS
       list = options(ifont) // 'h30w24b'
       CALL clearscreen( $GCLEARSCREEN )
       IF( setfont( list ) .GE. 0 ) THEN
         x = (vc.numxpixels-getgtextextent(face( ifont))) / 2
          IF( getfontinfo( fi ) .NE. 0 ) THEN
            CALL outtext( 'Error:  cannot get font info' )
            READ (*,*)
         END IF
         y = (vc.numypixels - fi.ascent) / 2
         CALL moveto( x, y, xy )
         IF( vc.numcolors .GT. 2 ) dummy = setcolor( ifont )
         CALL outgtext( face(ifont))
       ELSE
         CALL outtext( 'Error:  cannot set font' )
       END IF
       READ (*,*)
      END DO
      CALL unregisterfonts()
      dummy = setvideomode( $DEFAULTMODE )
      END
```

***Замечание*:** Все функции работы со шрифтами оказывают воздействие только на вывод с помощью функции outgtext.

7.5.7. Установка формы текстового курсора

Dummy=settextcursor(attr)

Функция settextcursor устанавливает форму курсора в текстовом режиме в зависимости от значения параметра attr.

Пример. Написать программу, которая в текстовом режиме очищает экран, выводит по главной диагонали букву L и номер строки, затем поменять направление и продолжить вывод на побочной диагонали , а затем поменять вид курсора.

```
      include  'fgraph.fi'
      program ffonts
      include  'fgraph.fd'
      parameter          ( nfonts = 6 )
      integer*2          dummy, x, y
      integer*4          ifont
      character*11         face(nfonts)
      character*10         options(nfonts)
      character*20         list
      RECORD /videoconfig/ vc
      RECORD /xycoord/     xy
      RECORD /fontinfo/    fi
      DATA face    / "Courier"   , "Helvetica"  , "Times Roman",
     +             "Modern"    , "Script"     , "Roman"      /
      DATA options / "t'courier'", "t'helv'"    , "t'tms rmn'" ,
     +              "t'modern'" , "t'script'"  , "t'roman'"   /
      call clearscreen( $GCLEARSCREEN )
      if( registerfonts( '*.FON' ). lt. 0 )
     +     stop 'Error: Не могу найти файлы'
      IF( setvideomode( $MAXRESMODE ) .EQ. 0 )
     +   STOP 'Error:  cannot set graphics mode'
      CALL getvideoconfig( vc )
      DO ifont = 1, NFONTS
       list = options(ifont) // 'h30w24b'
       CALL clearscreen( $GCLEARSCREEN )
       IF( setfont( list ) .GE. 0 ) THEN
         x = (vc.numxpixels-getgtextextent(face( ifont))) / 2
         IF( getfontinfo( fi ) .NE. 0 ) THEN
           CALL outtext( 'Error:  cannot get font info' )
           READ (*,*)
         END IF

      y = (vc.numypixels - fi.ascent) / 2
        CALL moveto( x, y, xy )
```

```
    IF( vc.numcolors .GT. 2 ) dummy = setcolor( ifont )
    CALL outgtext( face(ifont))
  ELSE
    CALL outtext( 'Error:  Не могу найти шрифт' )
  END IF
READ (*,*)
END DO
CALL unregisterfonts()
dummy = setvideomode( $DEFAULTMODE )
END
```

8. Выполнение программ

Для выполнения простых, учебных программ можно воспользоваться командой

FL name.for name1.for … namen.for

где name.for, name1.for, …, namen.for — имя файлов, в которых находится основная программа (name.for) и все подпрограммы пользователя (namei.for i=1, 2, …, n), необходимые для работы данной основной программы. Такая команда производит компиляцию основной программы и подпрограмм, при этом получаются одноименные файлы с расширением obj и один файл name.exe. Это объектные файлы, т.е. программы и подпрограммы, написанные на машинном языке, и выполняемый файл. Для больших программ требуется большое количество файлов, что требует существенного времени для компиляции программ. В этом случае все файлы, необходимые для программы, компилируют отдельно с параметром –c, записывая их объектные модули в библиотеку, или просто в какую либо директорию, а затем при помощи специальной программы LINK создают выполняемую программу с расширением exe. Для включения многих подпрограмм в один файл библиотечного типа используется программа обслуживания библиотек LIB.

В данном разделе рассматриваются опции команды компилятора FL, линковщика LINK и библиотекаря LIB.

8.1. Опции команды FL

Опции команды fl дают пользователю следующие возможности:

— установка модели памяти;

— выбор способа выполнения операций над числами с плавающей точкой;

— управление допустимыми свойствами языка Фортран;

— создание программ, исполняемых в среде операционной среды OS/2;

— оптимизация программ для минимизации размера занимаемой области памяти или максимизации скорости исполнения программ;

— автоматическое ассемблирование и компоновка процедур, написанных на машинном языке;

— создание путей поиска для файлов;

— переключение предупреждающих сообщений об ошибках компилятора;

— выбор формата страниц и заголовков для выходных листингов.

На самом деле для запуска процесса компиляции программы, написанной на Фортране, со стандартным расширением (.for) достаточно в командной строке написать следующее:

Fl name.for

Где name.for — тот файл, который вы хотите откомпилировать.

В итоге вы, если в программе были ошибки или помарки, получите сообщения об ошибках, их характере, номере строки, где они встретились, и т.д. Если же ошибок не найдено — это не значит, что их нет. Логические ошибки, ошибки типа бесконечного цикла и т.п. вполне могут быть. В связи с этим, обращайте внимание на замечания (warning). После компиляции вы получаете выполняемый файл с расширением .exe и объектный файл с расширением .obj.

Если вы компилируете программу, где используются графические функции, то надо в командной строке написать следующее:

Fl name.for graphics.lib

Замечание: Если вы имеете дело с более поздней версией Фортрана, например компилятора Microsoft Workstation, то в командной строке пишется:

Fl32 name.for

Если вы компилируете программу, где используются нестандартные функции, объектные коды которых находятся в какой-то библиотеке (или библиотеках), то надо в командной строке написать следующее:

Fl name.for list_of_librares

Где list_of_librares —список используемых программой библиотек, разделенных пробелами или запятыми.

В ряде случаев все же требуется нечто более сложное. Ниже приведены наиболее важные возможности компиляции.

Команда fl имеет следующий обобщенный формат:

FL[<option>..][<filespec> ..>[<option>..]

[/link[<libfield>][linkoptions]][/MA <options>]

Замечание*:* В квадратных скобках [ ] здесь и далее находятся необязательные аргументы, которые можно опустить.

Замечание*:* Если некоторое синтаксическое предложение не умещается на одной строке, оно продолжается на 2-й, 3-ей и т.д. строках.

Каждый параметр <option> является одним из описываемых ниже параметров; каждое «описание файла» <filespec>именует обрабатываемый файл.

В командной строке можно указать сколько угодно параметров и имен файлов; но при этом длина строки не должна превышать 128 литер.

Можно обрабатывать сразу группу файлов, имеющих требуемое расширение имени, используя для этого символы универсального сопоставления DOS.

Любое «описание файла», задаваемое в командной строке параметром <filespec>, может включать полную или частичную спецификацию пути, представляя возможность обрабатывать файлы из любых каталогов или на различных устройствах.

Каждому параметру предшествует символ слэш(/) за которым следует одна или несколько букв. Вместо символа /, можно использовать дефис (-). Например, /I и –I являются допустимыми формами представления параметра I. В тексте данного руководства при описании параметров используется /, хотя в сообщениях об ошибках применяется дефис.

***Замечание**:* Несмотря на то что имена файлов могут задаваться и строчными и прописными буквами, параметры следует задавать, учитывая регистр букв. Например: запись параметра в виде /Zd является правильной, а запись в виде /ZD или /zd неверна.

Параметры могут появляться в любом месте командной строки FL. В общем случае, параметр относится ко всем файлам, следующим за ним в командной строке, и не влияет на предшествующие файлы (однако данному правилу подчиняются не все параметры).

Для некоторых параметров указываются аргументы, например имена файлов, строки или числа. В некоторых случаях разрешается использование пробелов между опцией и ее аргументами: рекомендуется внимательно рассматривать синтаксис каждой опции во избежание ошибок.

Рассмотрим один наиболее важный параметр.

8.1.1. Получение справочной информации (параметр /help или /HELP)

При указании параметра /HELP возможно получение списка наиболее часто используемых параметров языка Фортран.

Для функционирования данного параметра, файл fl.hlp, содержащий описание всех параметров языка Фортран, должен находиться в текущем каталоге или присутствовать в описании пути в переменной PATH.

Спецификация данного параметра не зависит от регистра букв. При его записи допустима любая комбинация прописных и строчных букв.

При указании параметра /HELP в командной строке fl будет выдан список всех параметров. Никаких других действий выполняться не будет (даже если командная строка содержит какую-либо информацию).

8.1.2. Выполнение операций с плавающей точкой (/FP)

Следующие параметры предназначены для описания того, как программа пользователя осуществляет управление и обработку операций над числами с плавающей точкой.

/FPi— генерация встроенных инструкций и выбор математического пакета эмуляции.

8.1.3. Специфические параметры управления синтаксисом файлов исходных программ (/4Y, /4N)

При использовании этого параметра можно запрещать или разрешать использовать расширения стандарта Фортран 77. Буква Y указывает на разрешение параметра, а буква N на его запрещение. Параметры /4 могут быть составными. Например, параметр /4Ysfb запрещает все расширения языка Фортран фирмы Microsoft, разрешает ввод в свободном формате и разрешает режим расширенного управления ошибками. Следует отметить, что группы параметров Y и N должны указываться как отдельные параметры командной строки; они не могут «смешиваться» в один параметр.

Опишем один из таких параметров, который наиболее часто используется.

Параметр /4[Y,N]f — разрешает (/4Yf) или запрещает (/4Nf) использование свободного формата.

В программах, написанных в свободном формате можно не использовать оператор **$freeform.** При этом в команде для компиляции необходимо указать параметр /4Yf.

8.1.4. Именование объектного файла (параметр /Fo)

/Fo<objfile>

По умолчанию в команде fl каждому объектному файлу назначается то же «базовое» имя, что и имя соответствующего исходного файла, плюс расширение .obj. Параметр /Fo позволяет задавать объектному файлу некоторое другое имя или создавать его в другом каталоге.

Аргумент <objfile> должен следовать сразу за параметром, без пробелов. Этот файл может быть спецификацией файла, именем устройства или спецификацией пути. Если аргумент <objfile> является спецификацией файла, то параметр \Fo относится только к исходному файлу, следующему в командной строке сразу за параметрами. Объектный файл, создаваемый при компиляции данного исходного файла, имеет имя, задаваемое аргументом <objfile>.

Если аргумент <objfile> является именем устройства или спецификацией пути, то команда fl создает объектные файлы в заданном каталоге или устройстве. Для объектных файлов используются имена, принимаемые по умолчанию. Т.е. каждый объектный файл имеет «базовое» имя, соответствующее «базовому» имени исходного файла.

Замечание: Когда пользователь задает только спецификацию пути, тогда аргумент <objfile> должен оканчиваться символом обратного слэша (\).

8.1.5. Компиляция без компоновки (параметр /c)

Параметр /c (иногда он называется “только компиляция”) запрещает выполнение процедуры компоновки, т.е. получения готовой к выполнению программы. Параметр /c относится ко всей командной строке FL, независимо от того, в каком ее месте он указывается.

8.1.6. Создание файла листинга (параметр /Fs)

В команде fl допустимы ряд параметров, используемых для получения файла листинга.

Параметр /Fs[<listfile>] используется для получения листинга исходной программы.

Имя расширения файла, принимаемое по умолчанию, будет .lst.

Имя файла, принимаемое по умолчанию, используется тогда, когда параметр задается без аргументов или при использовании в качестве аргумента имени устройства или спецификации пути. Расширение, принимаемое по умолчанию, используется тогда, когда при задании имени файла расширение имени опущено.

8.2. Создание и модификация библиотек объектных модулей

Пользователь может создавать свои собственные библиотеки объектных модулей и в дальнейшем использовать функции из них, аналогично использованию библиотеки graphics.lib.

Для создания библиотек используется команда lib. Данная команда также используется для удаления модулей из библиотеки, копирования модуля из библиотеки в отдельный файл, добавления нового объектного модуля в библиотеку, замены существующего модуля новым и получения списка модулей, находящихся в библиотеке.

Команда lib имеет следующий синтаксис:

Lib oldlib[/PAGESIZE:number][commands][,[listfile][,newlib]]][;]

Где параметры имеют следующий смысл:

Oldlib — имя библиотеки (существующей или создающейся). Если такой библиотеки нет, то на экране появится надпись:

Library file does not exist. Create?

И необходимо нажать клавишу “У”, если вы хотите создать новую библиотеку, или “n”, чтобы прекратить работу команды lib.

По умолчанию библиотечные файлы имеют расширение .lib.

/P[AGESIZE]**:number**— опция, задающая размер страниц для размещения модулей в библиотеке. Number—целое число, представляющее степень 2 в пределах от 2 до 32768. При меньшем размере страницы библиотека получается более компактной, но при этом размер библиотеки не должен превышать number*65536 байт. По умолчанию number=16. При этом размер библиотеки не должен превышать 1Мб.

Commands—командные символы для манипуляции объектными модулями. Возможны следующие командные символы:

+ — добавление модуля в библиотеку. Также может использоваться для слияния библиотек. В этом случае за командой “+” должно следовать имя библиотеки.

- — удаление указанного модуля из библиотеки. Имя модуля не должно содержать расширений и имени директории.

-+ — замена указанного модуля. Модуль должен находиться в рабочей директории. Расширение модуля указывается (должно быть .obj).

***** — копирование модуля из библиотеки в объектный файл с именем, совпадающим с именем модуля. Файл создается в рабочей директории. Перенос его в другую директорию или на другой носитель, а также изменение его имени возможно только средствами операционной системы.

-* — перемещение модуля из библиотеки в объектный файл. Дает тот же эффект, что и последовательное выполнение команды lib с командными символами «*» и «-».

Замечание: Все эти команды вы сможете увидеть, если в командной строке напишете: lib – без всяких аргументов или lib /h.

Listfile — создает файл, содержащий информацию о библиотеке.

Команда lib не предполагает какого-либо расширения для данного файла, и, если не указано расширение, будет создан файл без расширения.

Newlib — задает имя библиотеки, создаваемой при модификации существующей библиотеки. Если имя не задано, то по умолчанию принимается имя существующей библиотеки. При этом фактически создается новая библиотека, а старая переименовывается, получая расширение .bak.

Приведем простейшие примеры работы с библиотекой.

Пример 1. Необходимо создать новую библиотеку из 4 подпрограмм или подпрограмм-функций пользователя.

Пусть три подпрограммы и одна подпрограмма-функция имеют следующие имена: Sub1, Sub2, Sub3, Fun и находятся в файлах с соответствующими именами с расширением for.

Тогда необходимо подать команды

```
Fl –c Sub1.for Sub2.for Sub3.for Fun.for
Lib Mylib.lib + Sub1 Sub2 Sub3 Fun;
```

Первая команда компилирует три исходных файла (файла на алгоритмическом языке) и получает три объектных файла (файла на машинном языке). Вторая команда создает (если не было) личную библиотеку пользователя с именем Mylib.lib и записывает в нее четыре объектных модуля Sub1.obj Sub2.obj Sub3.obj Fun.obj.

Если какой то модуль необходимо доработать и перезаписать в библиотеке, то подается команда Lib с параметром -+.

Пример 2. Допустим, подпрограмму Sub2 надо перезаписать. Тогда подается команда:

```
Lib Mylib.lib -+ Sub2;
```

Пример 3. Программа Progr1 использует программные модули Sub1 Sub2 Sub3 Fun3 и находится в файле Progr1.for. Необходимо подать команды для создания выполняемого модуля Progr1.exe.

```
Fl Progr1.for Mylib.lib
```

Команда компилирует основную программу Progr1.for и компонует ее в выполняемую программу. При компоновке программы, редактор автоматически выбирает требуемые функции и подпрограммы из библиотеки встроен-

ных функций, а если какие-то модули там не находит, то ищет их в библиотеке Mylib.lib.

Для решения данной учебной задачи можно обойтись без создания библиотеки. Для этого подать команду:

Fl Progr1.for Sub1.for Sub2.for Sub3.for Fun.for

Однако если в программе > 10 модулей, то лучше использовать библиотеки.

8.3. Редактор связей link

Для сборки всей объектных модулей и создания выполняемого файла наряду с командой fl можно применять редактор связей link.

Команда link имеет следующий формат:

Link objfiles,[,[exefile][,[mapfile][,[libfiles]]]][options][;]

где

Objfiles—список объектных файлов, разделенных пробелами или знаком ”+”,

Exefile — имя создаваемого выполняемого файла. Если это имя не указано, то по умолчанию имя файла будет совпадать с именем первого объектного файла и иметь расширение “.exe”.

Mapfile —имя файла для вывода карты памяти, т.е. размещения данных в памяти машины. По умолчанию предполагается пустой файл.

Libfiles— имена библиотек объектных файлов, в который будет производится поиск требуемых модулей. Имена библиотек также разделяются пробелом или знаком “+”. По умолчанию предполагается, что стандартные библиотеки Фортрана всегда добавляются в этот список.

Options— список опций команды link. Мы не будем их перечислять, а ограничимся только несколькими простыми примерами.

Пример 1. Пусть три подпрограммы и одна подпрограмма-функция имеют следующие имена: Sub1, Sub2, Sub3, Fun и находятся в файлах с соответствующими именами с расширением for. И в файле Progr1.for находится основная программа. Тогда можно откомпилировать все модули без компоновки командой

Fl –c Progr1.for Sub1.for Sub2.for Sub3.for Fun.for

Затем для компоновки всей программы подать команду

Link Progr1+Sub1+Sub2+Sub3+Fun;

которая получит выполняемый модуль в файле Progr1.exe.

Если необходимо изменить только один программный модуль, допустим Sub2, то надо его изменить и заново подать команды:

Fl –c Sub2.for

Link Progr1+Sub1+Sub2+Sub3+Fun;

Если программных модулей очень много, то список объектных модулей можно поместить в отдельный файл и при вызове команды Link указать имя файла, поставив при этом впереди имени символ @.

Пример 2. Пример 1 оформить с использованием файла для списка объектных модулей.

Link @Progr1.lnk

При этом в файле и именем Progr1.lnk поместить

Progr1+Sub1+Sub2+Sub3+Fun;

Список литературы

1. Соловьев П.В. Fortran для персонального компьютера. – М.: Арист, 1991 – 233с.
2. Брич З. С. , Гулецкая О. Н., Капелевич Д.В. и др. Фортран 77 ЕС ЭВМ. – М.: Финансы и статистика, 1989.- 351с.
3. Самохин А.Б., Самохина А.С. Фортран и вычислительные методы. – М.: Русина, 1994. - 120с.

ОГЛАВЛЕНИЕ