Министерство образования
Республики Беларусь

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ
ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра «Строительная механика»

В.М. Трепачко

КРАТКИЙ КУРС ПРОГРАММИРОВАНИЯ НА АЛГОРИТМИЧЕСКОМ ЯЗЫКЕ FORTRAN POWER STATION

Методическое пособие по дисциплине «Информатика»

Минск 2006

Министерство образования Республики Беларусь
БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра «Строительная механика»

В.М. Трепачко

КРАТКИЙ КУРС ПРОГРАММИРОВАНИЯ НА АЛГОРИТМИЧЕСКОМ ЯЗЫКЕ FORTRAN POWER STATION

Методическое пособие
по дисциплине «Информатика»
для студентов специальности
1-70 02 01 «Промышленное и гражданское строительство»

Минск 2006

УДК 004.438 (075.8)
~~ББК 22.18~~
Т 66

Рецензенты:
Н.А. Пашина, В.В. Саяпин

Трепачко, В.М.

Т 66 Краткий курс программирования на алгоритмическом языке Fortran Power Station: методическое пособие по дисциплине «Информатика» для студентов специальности 1-70 02 01 «Промышленное и гражданское строительство» / В.М. Трепачко. – Мн.: БНТУ, 2006. – 110 с.

ISBN 985-479-518-7.

Пособие представляет собой справочное руководство по составлению программ на алгоритмическом языке Fortran, которые являются необходимой составной частью контрольных и курсовой работ по дисциплине «Информатика», содержит описание основных средств алгоритмического языка, а также сведения по работе в среде Fortran Power Station. Пособие может также использоваться в качестве справочника по языку Fortran.

УДК 004.438(075.8)
ББК 22.18

ISBN 985-479-518-7

Введение

Это пособие адресовано главным образом пользователям персональных компьютеров, желающим изучить алгоритмический язык Фортран и вычислительные методы для своей работы или учебы. Для понимания материала данного пособия не требуется обладать знаниями в области программирования, а минимальный уровень математической подготовки может быть в пределах средней школы или, в крайнем случае, первого курса технического вуза. В пособии излагаются алгоритмический язык Фортран и особенности его применения для персональных компьютеров. Изложение сопровождается примерами, иллюстрирующими особенности применения конструкций языка. В последнем разделе приводятся некоторые задачи по языку Фортран с решениями, которые дополняют основной текст и помогают в изучении алгоритмического языка, а также могут служить примерами выполнения контрольных работ.

Автор надеется, что после изучения этого пособия читатель сможет самостоятельно составлять программы на Фортране для решения прикладных задач.

Часто приходится слышать от студентов вопрос: «Почему стоит использовать ФОРТРАН?».

Даже при наличии огромного количества программ для персональных компьютеров многим пользователям необходимо самостоятельно составлять программы – для решения инженерных, научно-технических, экономических и других задач. Для этого необходимо использовать языки программирования высокого уровня, поскольку написание программ непосредственно на машинном языке или близких к нему языках чрезвычайно непроизводительно.

Язык программирования Фортран (*FORTRAN – FORmula TRANslator*) появился в числе первых языков программирования в 1956 году. С тех пор он остается основным языком программирования при решении инженерных, научно-технических и других задач. Появлялась и проходила мода на другие языки. За прошедшие годы были разработаны языки программирования, которые, казалось, могли заставить уйти Фортран с арены программирования – в 1960-е годы это был язык Алгол, в 1970-е годы *PL*-1, в 1980-е годы Паскаль и Си. Тем не менее Алгол и *PL*-1 в настоящее время практически не используются, а языки Паскаль и Си не нашли широкого применения при решении инженерных задач. Такое долголетие Фортрана объясняется следующими причинами:

– постоянным развитием, т.е. появлением все более совершенных версий языка, которые включают предыдущие версии как свои подмножества, что делает возможным использовать старые Фортран-программы;

– эффективной трансляцией текста Фортран-программы в машинный язык, что обеспечивает высокую скорость обработки данных, которая имеет большое значение при решении вычислительных задач.

В процессе развития языка Фортран было разработано много различных версий, но наиболее популярным и широко распространенным был и остается *Fortran*-77, разработанный в 1977 году. К настоящему времени сформировался и обретает все большую популярность *Fortran Power Station.*

1. ЭТАПЫ РЕАЛИЗАЦИИ ИНЖЕНЕРНЫХ ЗАДАЧ НА КОМПЬЮТЕРЕ

При решении задачи на компьютере основная роль принадлежит человеку. Ни один опытный программист не станет составлять программу решения инженерной задачи сидя за компьютером. Написанию программы предшествует множество стадий, которые необходимо выполнять, чтобы до конца быть уверенным, что написанная программа верна. Процесс составления программ можно представить в виде последовательности описанных ниже стадий.

1. *Постановка задачи.* Суть этой стадии заключается в содержательной формулировке задачи и определении конечных целей решения.

2. *Построение математической модели.* Здесь предполагается математическая формулировка задачи, которую необходимо решить численно. Данный этап предусматривает глубокое понимание проблемы и знание соответствующих разделов математики. На данном этапе решения задачи необходимо: уяснить и четко изложить условие задачи; выбрать исходные переменные; выбрать переменные, подлежащие определению; записать ограничения переменных; записать связи между переменными в виде уравнений, неравенств, таблиц и т.д.

3. *Выбор метода решения.* На этой стадии выполняются два этапа анализа:

➢ математический анализ. Он включает в себя: подтверждение существования решения; подтверждение единственности решения; определение теоретических методов, которые будут использованы при решении задачи;

➢ численный анализ. При выполнении данного этапа производят: выбор наиболее приемлемого метода численного решения задачи с учетом особенностей математической модели решения задачи; оценку возможности реализации выбранного численного метода на принятом алгоритмическом языке, типе персонального компьютера, операционной системе и т.д.

Если не удается найти хороший метод решения, необходим возврат к предыдущим этапам, т.е. требуется построить новую модель или иначе сформулировать задачу.

4. *Разработка принципиальной схемы алгоритма.* В принципиальной схеме алгоритма, реализующего выбранный численный метод решения задачи, с максимальной лаконичностью должны быть отражены основные (без ненужной деталировки) блоки алгоритма. Они должны быть представлены в той последовательности действий, которая необходима для решения определенной задачи. Алгоритмы описываются разными способами. Удачной формой является словесная запись.

В практике программирования применяется общая методика построения принципиальной схемы алгоритма – метод пошаговой детализации («нисходящее программирование»), заключающийся в том, чтобы решение какой-либо сложной задачи свести к решению некоторого ряда более простых задач. Сначала в решаемой задаче выделяется небольшое число более простых задач (подзадач), а в проектируемой программе намечается соответствующее число блоков (частей программы), каждый из которых предназначен для решения одной из выделенных подзадач, определяются назначение каждого из них, порядок выполнения этих блоков и их связи между собой по обрабатываемым данным. На данном этапе важно определить лишь функциональное назначение каждого блока – что он должен делать, т.е. какие данные являются исходными для блока и что является результатом его работы. Вопрос о механизме реализации, т.е. как будут выполняться возложенные на блок функции, пока можно не рассматривать.

После определения порядка выполнения выделенных блоков и построения общей схемы алгоритма необходимо проверить его правильность. С этой целью программист, прослеживая логику выполнения алгоритма, должен убедиться в том, что к началу выполнения каждого блока все его исходные данные и все результаты на выходе из блока будут действительно определены, а на выходе из программы будут получены требуемые результаты – в предположении, что каждый блок правильно выполняет свои функции.

5. *Построение блок-схемы алгоритма.* На этой стадии сложные первоначально выделенные подзадачи необходимо повторно разбить на более простые блоки (второй шаг детализации). Этот процесс может продолжаться до тех пор, пока реализация блока не вызовет трудностей. Затем блок представляется графически в виде блок-схемы с подробной деталировкой. В блок-схеме все действия алгоритма изображаются геометрическими фигурами – прямоугольниками, трапециями, ромбами и т.д. (основные элементы блок-схем, требования к блок-схемам приведены в прил. 1, 2). Между фигурами восстанавливаются связи, показывающие последовательность выполнения действий. Блок-схема составляется так, чтобы каждому блоку соответствовала часть программы. Работа над блок-схемой позволяет обнаружить ошибки, допущенные при разработке принципиальной схемы алгоритма. Именно на этом этапе определяется сущность окончательного алгоритма, который будет реализован.

6. *Составление программы.* Программа – это последовательность команд, понятных компьютеру. Эта стадия состоит из двух этапов:

1) написание текста программы (конструирование программы);

2) набор текста программы на компьютере.

Конструирование программы производится с помощью алгоритмического языка, а ее перевод на язык компьютера (трансляция) осуществляется самой вычислительной машиной.

При наборе текста программы на компьютере необходимо стремиться к тому, чтобы программа была наглядной и удобочитаемой. Для этого можно: 1) сдвигать внутренние операторы («тело») цикла и условия на 3-4 позиции вправо; 2) давать необходимые комментарии; 3) размещать метки в возрастающем порядке от начала к концу программы; 4) группировать операторы формата. Кроме того, необходимо по возможности избегать операторов перехода и учитывать,

что операторы, имена переменных, констант, подпрограмм и т.д. не должны содержать русских символов.

7. *Отладка программы.* Цель данного этапа – выявление имеющихся в программе ошибок, определение и устранение причин, их вызвавших. При отладке программ разработчик сталкивается с двумя видами ошибок:

- синтаксические ошибки. Это ошибки в записи конструкций языка программирования (чисел, переменных, выражений, меток, операторов, подпрограмм и т.п.). Данные ошибки обнаруживаются на этапе компиляции. Исключение составляют ошибки в операторах формата, которые компилятором не фиксируются, а обнаруживаются только на этапе выполнения программы;
- семантические (смысловые, ошибки выполнения) ошибки. Они связаны с неправильным содержанием действий или использованием недопустимых значений величин. Данные ошибки приводят: к прекращению выполнения программы и выдаче предупреждений; отсутствию признаков выполнения программы из-за ее «зацикливания»; появлению непредусмотренной формы или содержания печати результатов; потере точности вычислений; большому времени выполнения вычислений или других операций.

8. *Решение контрольного (тестового) примера.* На данной стадии, используя принятые математическую модель и численный метод решения задачи, вычисления выполняют вручную по заданным величинам исходных данных, определяя при этом промежуточные и конечные результаты, которые будут контролироваться при расчете по программе. Рекомендуется приготовить несколько контрольных примеров, что позволит с большей достоверностью оценить в дальнейшем эффективность разработанной программы (особенно для алгоритмов разветвленного типа).

9. *Расчет задачи на компьютере.* На данной стадии необходимо: спланировать и подготовить данные для отладки, разработать исходные данные тестов, испытать программу в условиях ее использования, приближенных к реальным (на различных ОС, аппаратных системах и т.д.). Следует протестировать все возможные варианты прохождения программы.

Следует отметить, что при получении нелепых промежуточных либо окончательных результатов возможен возврат к любому предшествующему этапу. Для проверки достоверности результатов программа испытывается при решении контрольных задач.

10. *Разработка пояснительной записки.* Цель стадии – разработка детального описания принципов функционирования, используемых численных методов, ограничений на входные и выходные данные, требуемых затрат времени и других ресурсов на выполнение программы. В пояснительной записке могут указываться: область применения программы, ее возможности, сведения о затратах памяти, требуемые технические средства, сведения о надежности и т.д.

Следует отметить, что при решении какой-то задачи не обязательно будут задействованы все этапы представленной последовательности. В зависимости от сложности конкретной задачи некоторые пункты могут быть не использованы.

2. АЛГОРИТМИЧЕСКИЙ ЯЗЫК ФОРТРАН

2.1. Структура программы

Текст Фортран-программы представляет собой запись на языке программирования алгоритма решения поставленной задачи. При записи текста используются следующие символы:

1. Прописные или строчные буквы латинского алфавита от A до Z;
2. Цифры от 0 до 9;
3. Специальные символы (знаки арифметических операций, операции отношения, знаки пунктуации и др.).

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

Любая программа состоит из операторов (предложений) языка, которые располагаются в строках. Положение символа в строке нумеруется слева направо, начиная с 1-й позиции.

В языке Фортран имеются определённые правила, которых следует придерживаться при наборе программы на компьютере.

Записать программу в *Fortran Power Station* можно двумя способами.

Первый способ – классический (характерен для версии *Fortran*-77). Согласно ему, текст программы записывается с 7-й по 72-ю позиции строки (поле текста программы, рис. 2.1), причем в этих пределах расположение текста произвольное.

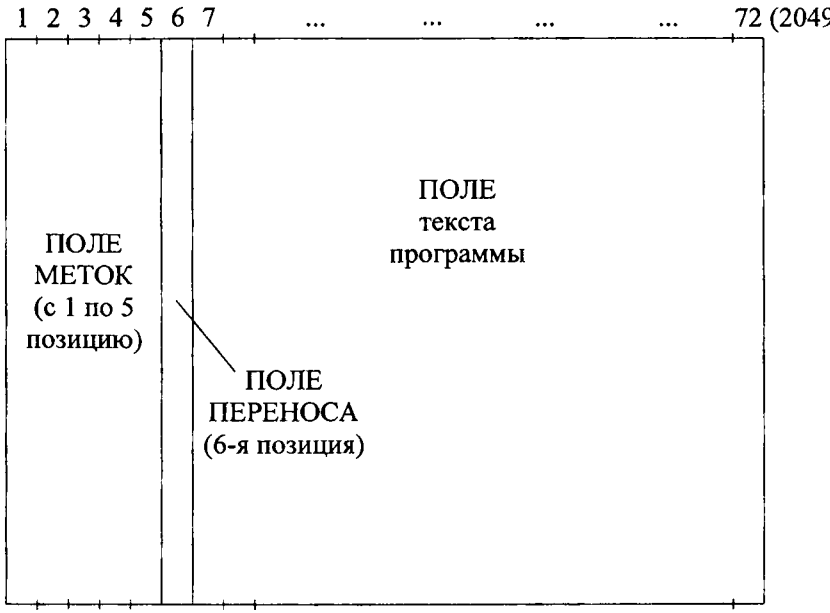


Рис. 2.1. Структура программы

Любой оператор может быть помечен меткой – целым десятичным числом (не более 5 цифр), метка располагается в позициях 1-5 строки (поле меток, см. рис. 2.1). Метки ставят не на все операторы, а только на те, на которые будут ссылки в программе. Назначение меток: метки дают возможность обращаться к нужной строке программы из любого места этой программы.

При использовании меток в программе необходимо учитывать следующее:

- ✓ в одной программе не может быть двух одинаковых меток;
- ✓ номера меток ставятся в произвольном порядке.

В строке программы должен размещаться только один оператор, однако если оператор не помещается в позициях 7-72 или желателен его перенос на следующую строку, то в каждой строке продолжения в 6-й позиции (поле переноса, см. рис. 2.1) печатается символ звездочка «*» либо любой другой символ, отличный от нуля. Если в первой позиции любой строки программы напечатан «!» (восклицательный знак), то такая строка рассматривается как комментарий

текста программы и транслятором игнорируется. Для записи текста комментария могут использоваться любые символы, а сам текст комментария располагается в позициях 2–72 строки. При сохранении программы, написанной таким способом, необходимо присваивать ей расширение FOR.

В качестве поясняющего текста может быть любая информация (она транслятором не воспринимается, а служит для читаемости программы при последующих просмотрах её текста).

Второй способ – универсальный. При реализации программы этим способом можно располагать операторы с 1-й позиции по 2049-ю, не соблюдая поле меток. В случае когда какой-то строке необходимо присвоить метку, ее располагают в самом начале строки, а затем не менее чем через пробел располагают оператор. При сохранении программы, написанной таким образом, можно присваивать ей расширение F90.

2.2. Типы данных

Каждая величина, используемая в тексте программы, может быть либо константой, либо переменной.

Константа – это величина, значение которой задается в тексте программы в явном виде и в дальнейшем не изменяется.

Переменная – это величина, значение которой может изменяться в процессе исполнения программы.

Обращение к константам и переменным осуществляется по имени.

Каждая величина, будь то константа или переменная, должна относиться к одному из типов данных. Для вычислительных задач основными являются следующие типы данных:

– целый;
– вещественный;
– вещественный с двойной точностью;
– комплексный;
– комплексный с двойной точностью;
– логический;
– текстовый.

Целые константы представляют собой последовательность цифр без десятичной точки. Перед отрицательным числом должен стоять знак «минус», а перед положительным знак «плюс» может отсутствовать.

Пример 2.1. Примеры целых констант

12 -1979 318 +5678

В памяти компьютера целая константа занимает 4 байта памяти и может иметь значение от -2147483648 до 2147483647.

Вещественная константа может быть представлена одним из следующих двух способов:

1) вещественная константа с фиксированной точкой. Сначала записывается знак числа «-» или «+» (знак «+» можно опускать), целая часть числа, а затем десятичная точка и дробная часть числа.

Пример 2.2. Примеры вещественных констант с фиксированной точкой

23.45 -654.321 .3 +5.

2) вещественная константа с плавающей точкой, которая записывается в виде

*n*E*m*

Здесь *n* – мантисса (представляется как вещественная константа), *m* порядок (однозначное или двузначное целое число со знаком или без знака).

Пример 2.3. Примеры вещественных констант с плавающей точкой

-0.00001E2 (в математике $-0{,}00001 \cdot 10^2$);

7.342E+11 (в математике $7{,}342 \cdot 10^{11}$);

0.03E-7 (в математике $0{,}03 \cdot 10^{-7}$).

В памяти компьютера вещественная константа занимает 4 байта памяти и имеет внутреннее представление как вещественная константа с порядком. Диапазон изменения – от -8,43E-37 до 3,37E+38, количество цифр в мантиссе – не более семи.

Вещественная константа двойной точности записывается только в виде вещественной константы с порядком. Для указания порядка вместо E используется D. В памяти компьютера эта константа занимает 8 байт памяти. Диапазон изменения – от -10D-64 до +9,99D+62, количество цифр в мантиссе – не более 14.

Пример 2.4. Примеры записи вещественных констант двойной точности

724D2 6.194D-14

Комплексная константа записывается в виде двух вещественных чисел, заключённых в круглые скобки и разделённых запятой, т.е.

$$(X_r , X_i) \quad ,$$

где X_r и X_i – вещественные константы;

X_r – действительная часть комплексного числа;

X_i – мнимая часть.

Пример 2.5. Примеры комплексных констант

(12.35,-1.129) (-3.94,2.0E-13),

которые в обычной математической записи имеют вид

$12{,}35 - 1{,}129i$, $-3{,}94 + 2 \cdot 10^{-13} i$

В памяти компьютера комплексная константа занимает 8 байт памяти.

Комплексная константа двойной точности состоит из пары вещественных констант двойной точности. В памяти компьютера эта константа занимает 16 байт памяти.

Пример 2.6. Примеры комплексных констант двойной точности

```
(2.1D0,-3.12D-2)
```

Логическая константа может принимать только два значения:

TRUE. (истина), .FALSE. (ложь).

Точки в записи являются обязательными. В памяти компьютера логическая константа занимает 4 байта.

Текстовая константа может быть представлена в 2 формах:

1) холлеритовская строка. Представляет собой число выводимых символов *n* (целая беззнаковая константа в диапазоне от 1 до 255), признак константы – буква **H** (латинская) и сами символы:

nH <*набор символов*>

Пример 2.7. Пример холлеритовской строки

```
9Нстроитель
```

2) строка символов, заключённая между двумя апострофами.

Пример 2.8. Пример строки символов

```
' Строительный  факультет '
' Специальность " ПГС " '
```

Примечание: символ «апостроф» внутри текста отображается 2 апострофами, идущими подряд.

Переменные и константы в программе различаются по именам. Имя (идентификатор) может содержать от 1 до 1320 символов. Причём первый символ – буква (строчная или прописная). Тип переменной или константы должен быть определен в программе и не может изменяться в процессе ее исполнения. Если никаких указаний о типе переменной или константы в программе нет, то переменная относится к одному из двух типов (целому или вещественному) по следующему правилу: если переменная начинается с букв *I, J, K, L, M, N*, то она считается переменной целого типа, а если переменная начинается с любой другой буквы латинского алфавита, то она считается переменной вещественного типа. Такое описание типа называется описанием типа по умолчанию. Во всех других случаях типы переменных задаются с помощью операторов описания.

Следует избегать имен (идентификаторов), совпадающих с операторами языка, например REAL, READ, а также с именами встроенных функций.

2.3. Операторы описания типов данных

2.3.1. Оператор описания переменных целого типа

Для описания переменных и массивов целого типа используется оператор INTEGER. Этот оператор также можно использовать для преобразования переменной или массива вещественного типа в переменную или массив целого типа. Будем рассматривать числовые величины только десятичной системы исчисления.

Данные целого типа делятся на две группы:

✓ обычной точности (описываются как INTEGER или INTEGER*2) – числа в диапазоне от -32767 до 32767;

✓ двойной точности (INTEGER*4) – числа от -2147483647 до 2147463647.

Примечание: 1) целые константы не должны выходить из диапазона; 2) десятичная точка недопустима в целой константе.

Пример 2.9. Примеры целых констант

123	+123	0
00000123	32767	-32767

Переменные или массивы, имена которых начинаются с *I, J, K, L, M, N,* по умолчанию относятся к целому типу (т.е. их не обязательно описывать оператором INTEGER).

Пример 2.10. Пример использования оператора описания

```
INTEGER  DELTA, B(10)
DELTA = 4
B(1) = 132
I = 1
B(2) = B(1) + DELTA
```

2.3.2. Оператор описания переменных вещественного типа

Для описания переменных и массивов вещественного типа используется оператор REAL.

Если имеется переменная или массив целого типа (когда имя начинается на любую букву из *I, J, K, L, M, N*), то с помощью оператора REAL можно преобразовать её в переменную вещественного типа.

Основная действительная константа содержит:

1. Необязательный знак («+» или «-»).
2. Целую часть.
3. Десятичную точку (.).
4. Дробную часть.
5. Необязательный показатель экспоненты (*E* или *D*).

Целая и дробная части содержат по одной или больше десятичных цифр, а десятичная точка является разделителем. Как целая часть, так и дробная могут отсутствовать, но не обе.

Пример 2.11. Примеры вещественных констант

-123.456	+123.456	123.456
-123.	+123.	123.
-.456	+.456	.456

Экспоненциальная часть содержит букву «*E*» или «*D*», за которой следует (необязательно) целая константа со знаком из одной или двух цифр. Экспонента показывает, что предшествующую величину нужно умножить на десять в степени целой константы.

Пример 2.12. Примеры простых экспоненциальных частей

E12 E-12 D+12 E0

Действительная константа может быть представлена как с экспоненциальной частью, так и без нее.

Пример 2.13

+1.000E-2	1.E-2	1E-2	Все это – одно и то
+0.01	100.0D-4	0.0001D+2	же число: 0,01.

Если число не содержит экспоненту, то его называют числом с фиксированной точкой, а если содержит – числом с плавающей точкой.

Данные вещественного типа делятся на две группы:

1) обычной точности (описываются как REAL*4 или REAL) – числа в диапазоне от $8{,}43 \cdot 10^{-37}$ до $3{,}37 \cdot 10^{38}$ (положительные числа), от $-3{,}37 \cdot 10^{38}$ до $-8{,}43 \cdot 10^{-37}$ (отрицательные числа), 0 (нуль) – для этих чисел экспонента E;

2) двойной точности (REAL*8 или DOUBLE PRECISION) – в диапазоне от $4{,}19 \cdot 10^{-307}$ до $1{,}67 \cdot 10^{308}$ (положительные числа), от $-1{,}67 \cdot 10^{308}$ до $-4{,}19 \cdot 10^{-307}$ (отрицательные числа), 0 (нуль) – для этих чисел экспонента D.

Пример 2.14. Пример использования оператора описания.

```
REAL  K, LAM (5)
K = 2.5
```

Переменная K и элементы массива LAM могут принимать в программе значения только вещественного типа.

2.3.3. Оператор описания переменных комплексного типа

Комплексная константа в Фортране состоит из необязательного знака, левой скобки, двух целых или действительных чисел, разделенных запятой, и правой скобки.

Для описания переменных комплексного типа используется оператор COMPLEX (для чисел обычной точности – COMPLEX или COMPLEX*8, для чисел двойной точности – COMPLEX*16). Каждая компонента (действительная и мнимая) COMPLEX*8 – это REAL*4. Каждая компонента COMPLEX*16 – это REAL*8. Все переменные комплексного типа обязательно должны быть описаны в программе.

Пример 2.15.

```
COMPLEX Q1, QX2, M12
Q1 = (-10.5, 1.23)
```

2.3.4. Оператор описания переменных символьного типа

Для описания переменных и массивов символьного (текстового) типа используется оператор CHARACTER. Если длина текстовой переменной не превышает 4 байт, то ее можно не описывать. Все остальные данные текстового типа должны быть описаны.

После оператора описания указывается звездочка, а за ней – количество символов, которое может содержаться в переменной.

Пример 2.16.

```
CHARACTER*10  C, C1, D12*15
C = 'Факультет'
```

Данная строчка означает, что в программе переменные *C*, *C*1 и *D*12 будут текстового типа (строковые), причём переменные *C*, *C*1 могут содержать до 10 символов. Переменная *D*12 может содержать до 15 символов текста.

2.3.5. Оператор описания переменных логического типа

Логический тип данных содержит две логических величины: .TRUE. и .FALSE. Описание переменных логического типа осуществляется оператором LOGICAL.

При описании могут применяться следующие формы оператора:

LOGICAL*2 занимает в памяти компьютера два байта (одно слово), причем младший значащий (первый) байт – либо 0 (.FALSE.), либо 1 (.TRUE.); старший значащий байт не определен;

LOGICAL*4 занимает в памяти четыре байта (два слова), младшее значащее (первое) из которых содержит величину LOGICAL*2. Старшее значащее слово не определено.

Пример 2.17.

```
LOGICAL T1,T2
```

2.3.6. Оператор описания массивов

Другой разновидностью переменных являются так называемые *индексированные переменные* или *массивы*.

Массив – это совокупность данных одного типа, объединённых одним именем (массив может состоять из одного элемента). Всякий массив обязательно имеет размерность.

Массивы бывают:

- ✓ одномерными (другое их название – *вектор-столбец*);
- ✓ двумерными (другое название – *матрица*);
- ✓ трехмерными и т.д. Максимальная размерность массива равна 7.

Одномерный массив – это последовательность ячеек, расположенных в одну линию. На рис. 2.2 приведен пример такого массива.

Индекс i	1	2	3	4	5	6	7	8
Значение элемента q_i =	2,3	-6	0	4,4	-3	0	8,2	4,7

Рис. 2.2. Одномерный массив q

Массив имеет имя q. Для того чтобы можно было отличить одну ячейку массива от другой ячейки этого же массива, их нумеруют. Нумерация ячеек обычно начинается с 1. Номер ячейки массива называется его *индексом*, а константа в ячейке – *элементом* массива. Теперь становится возможной работа с отдельной ячейкой такого массива. Например, команда $q_7 = 8{,}2$ означает, что 7-й элемент массива q равен 8,2. Команда $r_{41} = q_2 + q_5$ означает, что 41-му элементу массива r присваивается значение суммы 2-го и 5-го элементов массива q.

Двумерный массив по расположению ячеек напоминает математическую матрицу (рис. 2.3). Элемент такого массива характеризуется двумя индексами: первый показывает строку, второй – столбец. Например, команда $d_{2,5} = 43$ означает, что элемент со значением 43 размещается на пересечении 2-й строки и 5-го столбца двумерного массива d.

	1	2	3	4	**5**	6
1	-2	4	6	-8	-3	1
2	14	112	7	**-10**	**43**	3
3	0	-5	-9	12	6	0
4	16	14	12	11	5	9

Рис. 2.3. Двумерный массив *d*

Аналогично устроена структура массивов трех и большей размерности.

Массивы должны быть описаны в программе одним из операторов описания, приведенных выше (см. п. 2.3.1...2.3.5), в зависимости от типа элементов массива. Кроме того, существует специальный оператор для их описания:

DIMENSION <имя массива> (<размерность>)

Пример 2.18.

```
DIMENSION  A(10),B(8),C(3,4),K(15)
```

Оператор не выполняет никаких действий в программе и служит лишь для отведения места в памяти компьютера на размещение массивов. Изначально всем элементам массива присваиваются значения, равные нулю.

Пример 2.19.

```
DIMENSION A(10), C(4,6)
K=4
M=3
```

```
A(1)=2.5
A(3)=1.7E-7
A(K+M)=D+A(1)
A(4)=A(M)+A(K+1) ! Эта строка аналогична A(4)=A(3)+A(4+1).
```

При выполнении программы транслятор анализирует, не превышает ли значение вызываемого индекса массива его предельного значения, описанного в операторе DIMENSION (т.е. проверяется случай когда вызываемый индекс массива не существует – массив имеет меньшие размеры). Если, например, для массива *C*, приведенного выше, производится обращение к элементу *C*(5,1), то программа выдаст сообщение об ошибке.

Как указано ранее, массивы можно описать ещё и с помощью операторов описания типов переменных.

Пример 2.20.

```
REAL      C(4,6) ! Вещественный массив С из 24 элементов, расположенных в 4 строках и 6 столбцах.
INTEGER K(15) !Целочисленный массив К из 15 элементов, расположенных в одну строку.
```

Примечание: повторное описание массивов не допускается.

2.4. Арифметические выражения

Арифметическое выражение – это запись математической формулы с использованием констант, переменных, функций, знаков арифметических операций и круглых скобок.

Знаки арифметических операций: + (сложение), - (вычитание), * (умножение), / (деление), ** (возведение в степень).

Скобки в арифметических выражениях имеют обычный математический смысл.

При написании арифметических выражений следует соблюдать определенные правила. Так, два знака операции не могут стоять рядом, а если это необходимо, то они должны быть разделены скобками. Например, запись вида X/-Y неверна, а запись X/(-Y) верна.

Если в арифметическом выражении отсутствуют круглые скобки, то порядок выполнения операций будет следующим:

а) вычисление функций;
б) возведение в степень;
в) умножение и деление;
г) сложение и вычитание.

Операции одного ранга выполняются последовательно слева направо, за исключением операции возведения в степень, которая выполняется справа налево. Например: A=B**C**2 будет выполняться как A=B**(C**2).

При написании арифметических выражений во избежание ошибок, а также для задания очерёдности выполнения арифметических операций, целесообразно использовать скобки, причем количество открытых скобок должно быть равно количеству закрытых скобок.

Пример 2.21. Записать на алгоритмическом языке Фортран арифметическое выражение

$$\frac{x \cdot y}{x^2 + 7{,}5} + \cos x .$$

Решение: X * Y / (X ** 2 + 7.5) + COS (X)

В приведенном примере происходит обращение к библиотечной функции Фортрана COS, полный список которых содержится в прил. 3. При обращении к библиотечным функциям необходимо указать имя функции, а после имени в скобках – аргументы. При этом аргументы могут быть арифметическими выражениями.

При использовании библиотечных функций необходимо учитывать следующее:

- отрицательное число не может быть возведено в вещественную степень;
- для тригонометрических функций аргумент указывается в радианах.

2.5. Учет типов величин при записи арифметических выражений

Величины разных типов (целые, вещественные и т.д.) в памяти компьютера кодируются по-разному, и, следовательно, арифметические операции над ними выполняются, вообще говоря, также по-разному.

Для целых величин определены все операции, однако результат не должен выходить из множества целых чисел. Поэтому результатом деления двух целых величин будет целая часть полученного частного. Например, если J = 9, I = 4, то J / I = 2.

Любая вещественная величина может быть возведена в целую степень. Операция возведения вещественных величин в вещественную степень вида A**B выполняется только в том случае, если A>0.

Для комплексных величин определены операции сложения, вычитания, умножения и деления. Операция возведения в степень для комплексной величины выполняется только в том случае, если показатель степени – целая величина.

Когда в арифметическом выражении все операнды (переменные и константы) одного типа, то величина, являющаяся результатом выполнения этого выражения, того же типа. Когда операнды имеют разный тип, то тип результата, как правило, определяется типом операнда максимального ранга.

Ранг операнда зависит от его типа (табл. 2.1).

Таблица 2.1

Ранг операнда	Тип параметра
1	INTEGER
2	REAL
3	REAL*8
4	COMPLEX
5	COMPLEX*16

Например, результатом арифметического выражения с операндами INTEGER (ранг = 1) и REAL (ранг = 2) будет величина, относящаяся к типу REAL.

Особый случай: операции над операндами REAL*8 и COMPLEX породят величину COMPLEX*16, а не COMPLEX.

В любой Фортран-программе должен присутствовать по крайней мере один выполняемый оператор. Простейшим выполняемым оператором является арифметический оператор присваивания, который имеет следующий вид:

V = A.

Здесь V – имя переменной, а A – арифметическое выражение.

Правило выполнения: переменной с именем V присваивается значение арифметического выражения A.

Если переменная V – целая, а тип A – вещественный, то V будет присвоено значение целой части результата A, а при комплексном типе A переменной V будет присвоено значение целой части от действительной части результата A.

Если переменная V – вещественная, а тип A – комплексный, то V будет присвоено значение действительной части результата A.

Пример 2.22. Примеры результатов арифметических операций над различными типами величин (табл. 2.2).

Таблица 2.2

Запись операции на языке Фортран	Значение переменной		Примечание
	при математических вычислениях	в результате выполнения операции	
A = 1/3	0,33333(3)	A = 0	Результат арифметической операции над целыми числами – целое число, над вещественными – вещественное
A = 1./3.	0,33333(3)	A = 0.33333(3)	
I = 199/100	1,99	I = 1	
J = 1./3	0,33333(3)	J = 0	
Y = X**(1/3)	$X^{0.33333(3)}$	$Y = X^0 = 1$	
Y = X**(1./3.)	$X^{0.33333(3)}$	$Y = X^{0.33333(3)}$	

Замечания:

1. Следует строго отслеживать запись арифметических выражений и анализировать тип и величину результата. Наибольшее количество ошибок возникает при операциях с различными типами параметров.

2. Все переменные, находящиеся справа от знака равенства в операторе присваивания, должны иметь конкретные значения, которые задаются в предыдущей части программы либо с помощью других операторов присваивания, либо при задании исходных данных с помощью операторов ввода. Если значение переменной ранее в программе не задано, то ее значение равно нулю.

2.6. Последовательность создания Фортран-программ

После написания текста программы на языке Фортран возникает необходимость ее запуска для проверки ее работоспособности. Как и на других языках высокого уровня, на Фортране возможно выполнение файла, имеющего расширение .EXE. Для запуска программы необходимо выполнить следующую последовательность действий:

1. Сохранение исходного текста программы.

Исходный текст программы создается любым текстовым редактором или непосредственно в оболочке *Fortran PS* и сохраняется с именем (желательно не более 8 символов) и расширением .FOR или .F90 (см. п. 2.1). Например, TEXT1.FOR. Для сохранения файла в *Fortran PS* необходимо нажать клавишу F12.

2. Компиляция программы (исправление ошибок).

Составленная программа на Фортране может содержать ошибки. Существуют два основных вида ошибок – синтаксические и семантические.

Синтаксические ошибки обусловлены неправильным использованием конструкции алгоритмического языка (ошибки в названиях операторов, пропуск знаков и пр.). Эти ошибки указываются после трансляции программы. Исправление этих ошибок, как правило, не вызывает трудностей даже для начинающих пользователей.

Семантические ошибки обусловлены неверным алгоритмом ре-

шения или неправильным размещением обрабатываемых данных в памяти. Эти ошибки (деление на ноль, использование неопределенных величин, недопустимое значение индекса массива) проявляются в том, что при запуске программы на выполнение на экране компьютера может появиться сообщение о переполнении разрядной сетки в какой-то ячейке памяти.

Для компиляции программы в *Fortran PS* необходимо нажать сочетание клавиш CTRL+F8. Затем в нижней части экрана будет выведено количество ошибок (error) и, если они присутствуют, в скобках будут указаны номера строк, в которых они содержатся. Для быстрого перехода к строке с ошибкой, необходимо дважды нажать левой кнопкой мыши на запись, указывающую на ошибку. После исправления ошибки программу снова запускают на компиляцию. Только после устранения всех ошибок переходят к следующему этапу.

3. Трансляция программы (создание файла для запуска или EXE-файла).

Для трансляции программы в *Fortran PS* необходимо нажать сочетание клавиш SHIFT+F8. Затем в нижней части экрана будет выведено сообщение о наличии ошибок. Как правило, ошибки на этом этапе связаны с переполнением ячеек памяти и неверным использование внешних процедур. При отсутствии ошибок выполняется следующая операция.

4. Запуск программы на выполнение.

Для запуска готовой программы в *Fortran PS* необходимо нажать сочетание клавиш CTRL+F5. В появившемся окне следует ввести требуемые данные и просмотреть полученные результаты.

Иногда возникает необходимость экстренного прерывания выполнения программы. Для этого необходимо нажать сочетание клавиш CTRL+Break или создания искусственной сбойной ситуации, например, путем ввода текстовой информации вместо числовой.

2.7. Операторы ввода/вывода

2.7.1. Оператор ввода DATA

Оператор DATA служит для задания значений переменным или массивам на этапе трансляции программы. Общая форма записи оператора

DATA *<список переменных>* / *<список значений>*/

В списке имён указываются имена переменных, имена и элементы массивов, которым перед началом выполнения программы должны быть присвоены некоторые значения.

Примечание: список переменных должен соответствовать списку значений по очередности следования и по типу данных; количество переменных не должно превышать количество значений.

Пример 2.23.

```
DIMENSION C(10)
DATA A , B , K , N / 2.3 , -1.6E-4 , 13 , 9 /
DATA C / 4.1 , 5.5 , 6*0. , -12.7 , 14. /
```

В результате выполнения этой части программы переменные получат следующие значения: $A = 2{,}3$, $B = -1{,}6 \cdot 10^{-4}$, $K = 13$, $N = 9$.

Заполнение массива C, состоящего из 10 элементов, происходит следующим образом: $C(1) = 4{,}1$; $C(2) = 5{,}5$; затем следует групповой множитель 6*0., который означает – шесть раз повторить число 0., т.е. $C(3) = C(4) = C(5) = C(6) = C(7) = C(8) = 0$; далее $C(9) = -12{,}7$; $C(10) = 14$.

Пример 2.24.

```
DATA C(1) , C(2) , C(8) / 3*7. /
```

Элементам массива *C* с индексами 1, 2 и 8 будет присвоено значение 7.

В операторе DATA могут присутствовать данные как целого и вещественного типа, так и комплексного, текстового и логического типов.

Пример 2.25.

```
COMPLEX  D1, D2
CHARACTER*6  E1, E2, E3*12
DATA  D1, E1 / (-3.6 , 15.7E3) , 'ОСЕНЬ␣' /
```

Примечания:

1) оператор DATA действует в программе один раз во время трансляции программы и не может изменить значения переменных при повторном его выполнении;

2) операторов DATA в программе может быть сколько угодно;

3) операторы DATA могут располагаться в любом месте программы.

2.7.2. Оператор ввода READ

Этот оператор позволяет осуществлять ввод данных с клавиатуры или другого устройства (например, внешнего файла) в процессе выполнения программы. Общий вид оператора

READ (*n* , *m*) *<список ввода>*,

где *n* и *m* – натуральные числа;

n – номер устройства, с которого производится считывание информации (если вместо *n* указывается * или цифра 5, то считывание данных производится с клавиатуры);

m – номер метки, после которой следует оператор формата записи считываемых величин (если вместо *m* указывается *, то ввод значений производится в произвольной форме – бесформатный ввод).

Пример 2.26.

```
READ (*,*) A
READ (5,*) B
READ (1,*) C
```

Считывание значений переменных *A* и *B* производится с клавиатуры, а переменной *C* – с внешнего устройства 1. Значения этих переменных могут задаваться в произвольной форме (бесформатный ввод).

Пример ввода данных по формату будет приведен ниже (см. п. 2.8).

2.7.3. Оператор вывода WRITE

Этот оператор позволяет осуществлять вывод данных на экран или другое устройство (например, внешний файл) в процессе выполнения программы. Общий вид оператора

WRITE (*n* , *m*) *<список вывода>*,

где *n* – номер устройства, на которое производится вывод информации (*n* – целое число; если вместо *n* указывается * или цифра 5, то вывод производится на экран);

m – номер метки, после которой следует оператор формата.

Пример 2.27.

```
WRITE (*,*) D
WRITE (5,*) E
WRITE (2,*) F
WRITE (*,*) D+E
```

Вывод значений переменных *D* и *E* производится на экран, а переменной *F* – на внешнее устройство 2. Результат суммы переменных *D* и *E* также будет выведен на экран. Значения всех переменных будут выводиться в произвольной форме (бесформатный вывод).

Примеры вывода данных по формату будут приведены ниже (см. п. 2.8).

2.7.4. Оператор вывода на экран PRINT

Этот оператор позволяет осуществлять вывод данных в процессе выполнения программы только на экран. Общий вид оператора

PRINT *m* , *<список вывода>*,

где *m* – номер метки, после которой следует оператор формата (если вместо *m* указывается *, то вывод значений производится в произвольной форме).

Пример 2.28.

```
PRINT *, K
```

Вывод значения переменной *K* производится на экран (бесформатный вывод). Пример вывода данных по формату будет приведен ниже (п. 2.8).

2.8. Оператор задания формата ввода – вывода (FORMAT)

Оператор FORMAT является невыполняемым оператором. Он всегда имеет метку, но на эту метку нельзя передавать управление. Он может находиться в любом месте программы и используется для ввода и вывода информации в сочетании с операторами ввода – вывода READ и WRITE.

В общем виде оператор формата можно записать

m FORMAT (<*список спецификаций*>).

Здесь *m* – номер метки.

Пример 2.29.

```
      READ ( 5 , 3 ) I , K
3     FORMAT ( I4 , I5 )
```

Оператор ввода READ означает: читать с устройства 5 по формату 3 значения переменных *I* и *K*.

Оператор FORMAT указывает, каким образом будут считываться данные с носителя информации (строка экрана, строка текстового файла или устройство передачи данных) или выводиться на носитель информации.

В списке спецификаций указывается, по какой форме будет произведено считывание значений для переменных. Спецификации бывают следующих типов: *I, F, E, G, X, T, A*.

2.8.1. Спецификации X, T

Спецификация X служит для отступа (пропуска) указанного числа пробелов. В общем виде записывается – nX. При вводе информации обозначает «пропустить, не читая, n позиций». При выводе информации обозначает «пропустить *n* позиций (вывести n пробелов)».

Для вывода информации в конкретное место строки используется спецификация Tn, где n – номер позиции, начиная с которой будет вводиться или выводиться информация.

Спецификации X и T по своей сути одинаковы. Отличие заключается в том, что при использовании спецификации T на экране будет отступ на n-1 позицию, а для X – n позиций.

2.8.2. Спецификация I

В общем виде Iw, где w – количество позиций для записи числа.

Спецификация I используется для ввода – вывода информации только целого типа. Поэтому переменная, которой присваивается считанная величина, обязательно должна быть целого типа (начинаться с букв *I, J, K, L, M, N* или должна быть описана при помощи оператора INTEGER). В противном случае при трансляции будет обнаружена ошибка.

Пример. 2.30. Пример ввода данных по спецификации I.

```
      READ ( 5, 3 ) I1, K
3     FORMAT ( I4, I5 )
```

Если ввести подряд 9 цифр, то переменной I1 присвоятся числа из первых 4 позиций, а переменной *K* присвоятся числа из следующих 5 позиций. Если при вводе цифр будет больше чем 9, то лишние символы будут проигнорированы.

Пример. 2.31. Пример вывода по спецификации I.

```
      WRITE ( 5, 4 ) I1, K
4     FORMAT (1X , 'Переменная I1=' , I4 , 2X , 'Переменная
K=' , I5 )
```

При выполнении данного блока в строке отступается один пробел (спецификация 1X), выводится на экран текст: «Переменная I1=», следом числовое значение переменной I1 будет выведено в четырех позициях, через два пробела после этого выводится текст: «Переменная K=», и в следующих пяти позициях выводится числовое значение переменной *K*.

Если I1=123, *K*=10005, то на экране будет представлена следующая информация:

␣Переменная I1=␣123␣␣Переменная K=10005

Примечания:

1) при форматном выводе в начале каждой новой строки целесообразно отступить хотя бы на один пробел, т.к. первая позиция строки служит для управления режимом вывода информации;

2) если количество символов в переменной меньше, чем количество позиций для записи числа, то сначала отступается несколько пробелов, а затем выводится само число. Если же количество символов больше количества позиций, то на экран выводятся символы «» (звездочка).*

Пример 2.32. Примеры вывода чисел по спецификации I

Тип переменной	Значение переменной	Вывод переменной по спецификации		
		I2	I3	I4
Integer	23	23	␣23	␣␣23
Integer	127	**	127	␣127
Real	10.25	**	***	****

Примечание: число 10,25 является вещественным и по формату I выведено не будет.

2.8.3. Разделители

Разделителем оператора FORMAT может быть запятая («,») или символ «слэш» («/»). Символ «слэш», расположенный в тексте оператора FORMAT, означает переход на новую строку или новую запись. Для продолжения вывода информации в текущей строке используется символ «обратный слэш» («\»).

Первая позиция каждой строки при выводе информации служит для управления печатью. Если в первой позиции стоит символ ␣ (пробел) или текстовый символ, то вывод осуществляется в текущей строке. При этом информация, попадающая в первую позицию, теряется. Если в первую позицию заносится «0» (ноль), то информация выводится через одну строку. Одна страница при выводе на печатающее устройство содержит 63 строки, а при выводе на экран – 25.

Пример 2.33.

```
      I1=123
      K=10005
      PRINT  6, I1, K
6     FORMAT ( 2X , I3 / 4X , I5 )
```

После выполнения оператора FORMAT на одной строке экрана будет выведен 1 пробел (1 пробел теряется) и значение $I1$, на следующей строке – 3 пробела (1 пробел теряется) и значение K, т.е. на экране будет выведено

```
␣123
␣␣␣10005
```

Пример 2.34

```
      WRITE (5,10)
10    FORMAT (1X, ' Введите значение x= ' \)
      READ (5,*) x
```

Данный блок сработает следующим образом: появится надпись «Введите значение x=». Вводимое число будет отображаться сразу после знака «равно» (из-за знака «обратный слэш»). Если не использовать «обратный слэш», то вводимое число будет отображаться на следующей строке, что менее наглядно.

2.8.4. Спецификация F

Используется для ввода-вывода данных только вещественного типа. Позволяет вводить и выводить информацию с фиксированной десятичной точкой.

Общая запись спецификации F

Fw.d

где w – количество позиций, отведенных под число, включая знак и десятичную точку;

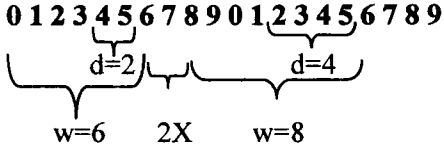
d – количество позиций из w, в которых размещается дробная часть числа.

Пример 2.35. Пример ввода информации по спецификации F.

```
      READ (5,11) A,B
11    FORMAT ( F6.2 , 2X , F8.4 )
```

Информация для ввода по спецификации F может быть представлена в 2 формах: без десятичной точки и с десятичной точкой.

Пример 2.36. Пример ввода информации без десятичной точки. Предположим, что на экране дисплея мы набрали следующую информацию:



Ввод значения переменной *A* производится по формату F6.2 (см. пример 2.35). Это значит, что первое число будет читаться из первых 6 ($w = 6$) позиций, из которых 2 последние позиции ($d = 2$) будут считаться дробной частью вводимого числа. Переменной *A* присвоится значение 123,45. Затем будут пропущены два символа 6 и 7 (так как используется пропуск символов по формату 2X) и произведётся счи-

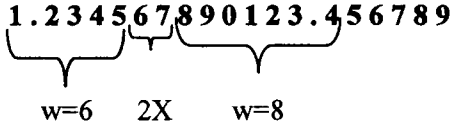
тывание значения переменной *B* по формату F8.4, т.е. её значение составит 8901,234000. Последние цифры 5, 6, 7, 8, 9 не входят в перечень описанных в операторе FORMAT полей и считываться и обрабатываться не будут.

Пример 2.37. Пример ввода информации с десятичной точкой. В этом случае параметр *d* не играет никакой роли, а число читается из отведенного для числа количества позиций *w*.

Для спецификации F6.2 можно задавать численные данные

–123.5
99999.
.10005

При вводе данных с десятичной точкой в виде



получим значения переменных *A* = 1,2345, *B* = 890123,4. Цифры 6 и 7 будут пропущены по спецификации 2X. Последние цифры (5, 6, 7, 8, 9) не прочитаются, так как выходят за пределы обрабатываемых полей.

Пример 2.38. Пример вывода информации по спецификации F

Тип переменной	Значение переменной	Вывод переменной по спецификации		
		F6.2 ␣␣␣.␣␣	F5.2 ␣␣.␣␣	F7.3 ␣␣␣.␣␣␣
Real	12,3456	␣12.35	12.35	␣12.346
Real	-98,76	-98.76	*****	-98.760
Real	-0,100056	␣␣-.10	␣-.10	␣␣-.100
Integer	123	******	*****	*******

Примечание: число 123 является целым и по формату F выведено не будет.

При выводе по формату F5.2 максимальное число, которое можно вывести, – числа из интервала [-9,99 ; 99.994(9)]. Для примера: A = 99,989 – выведет по этому формату; A = 99,999 – не выведет по этому формату, так как d = 2, то оставшуюся дробную часть округлит и все число округлится до 100, а число 100 вывести по этому формату нельзя).

Примечание: если выводимое число не может быть выведено по указанному формату (например, число 100 по F5.2), то вместо числа в отведенных позициях напечатаются символы «»" (для числа 100 выведет ******).*

2.8.5. Использование повторителей в операторе FORMAT

Повторители рекомендуется использовать, когда по формату вводится/выводится несколько переменных. С их использованием строка с оператором FORMAT получается короче и нагляднее. Работу с использованием повторителей рассмотрим на примере.

Пример 2.39.

```
      READ (5,8) A, B, X, V
8     FORMAT ( F4.2, 3F5.3 )
```

В этом примере переменная A будет читаться по формату F4.2, а переменные B, X, V будут считываться по формату F5.3, который из-за повторителя (цифра 3) сработает, как сработала бы следующая строка:

```
8     FORMAT ( F4.2, F5.3, F5.3, F5.3)
```

То есть в данном примере повторитель (цифра 3) заменил три F5.3. Кроме того, повторители могут использоваться для группы форматов:

```
      READ (5,9) X, B, C, K, D1, D2, M
9     FORMAT ( F6.2, 2 ( 2F4.2, I3 ) )
```

2 – повторитель, который обозначает, что группа форматов повторяется два раза.

Порядок считывания по форматам: X – по F6.2; B и C – по F4.2; K – по I3; D1 и D2 – по F4.2; M – по I3.

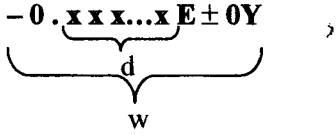
Примечание: если группа форматов заключена в скобки и число повторителей перед скобкой не указано, то группа форматов будет повторяться бесконечное число раз до тех пор, пока не будет исчерпан список оператора ввода/вывода.

Все данные должны быть расположены в одной строке. Для перехода на другую строку при вводе (выводе) информации в операторе FORMAT в качестве разделителя используется символ «/» – «слэш».

2.8.6. Спецификация E

Спецификация E используется для ввода – вывода данных вещественного типа и позволяет вводить и выводить информацию с фиксированной десятичной точкой.

Общая запись спецификации E



где w – количество позиций, отведенных под число, включая знак, десятичную точку, показатель экспоненты и степень;

d – количество позиций из w, в которых размещается дробная часть числа.

При выводе по спецификации E обязательно должно выполняться условие

$$w-d \geq 7.$$

Это происходит из-за того, что 3 позиции в начале выводимой строки и 4 позиции в конце строки отводятся под вывод служебной информации. Если на компьютере установлено гашение незначащего нуля, то $w-d \geq 6$.

Необходимо предусматривать достаточное количество цифр после запятой (параметр *d*), иначе при выводе будет происходить округление отбрасываемой части и результат может сильно измениться.

Данные по спецификации E12.5 выводятся в не совсем обычном виде:

0	выводится как	0.00000E+00
1	выводится как	0.10000E+01
101	выводится как	0.10100E+03
0.0012	выводится как	0.12000E–02

Пример 2.40. Примеры вывода числа 1,4878960 по различным форматам E

Формат вывода	Результат	Погрешность
E9.1	0.1E+01	49 %
E9.2	0.15E+01	1 %
E10.3	0.149E+01	0,01 %

Примечание: при бесформатном выводе данные автоматически выводятся по спецификации E15.7.

2.8.7. Спецификация Gw.d

Спецификация G является универсальной и может применяться как для ввода и вывода переменных целого типа, так и вещественного.

Для переменных целого типа формат Gw.d соответствует и работает как формат Iw.

Для переменных вещественного типа при вводе информации она может быть представлена как по формату Fw.d, так и по формату Ew.d. То есть если информацию вводить без десятичной или с десятичной точкой, то спецификация G будет полностью работать как формат F. Если информацию вводить с плавающей запятой, то спецификация G будет работать как формат E.

Пример 2.41. Пример вывода различных значений переменной *Z* по формату G8.2

```
      WRITE (*,10) Z
10    FORMAT ( 1X, G8.2 )
```

Тип переменной	Значение переменной *Z*	G8.2 ␣␣␣␣␣␣␣␣	Примечание
INTEGER	1235	␣␣␣␣1235	аналогично формату I8
REAL	-98,762	␣␣-98.76	аналогично формату F8.2
То же	-12345,67890	-.12E+05	аналогично формату E8.2 (если установлено гашение нуля)
	-12345,67890	********	не установлено гашение нуля

Предыдущий пример может быть переписан проще и короче без использования оператора FORMAT:

```
      WRITE ( *, ' ( 1X , G8.2 ) ' ) M
```

или

```
      PRINT ' ( 1X , G8.2 ) ' , M
```

Запись операторов ввода – вывода с оператором FORMAT длиннее, но позволяет несколько раз использовать один и тот же оператор FORMAT, обращаясь к нему из разных мест программы.

2.9. Операторы условия

Оператор условия служит для изменения порядка выполнения операторов в зависимости от какого-либо условия. Этот оператор используется для организации циклов, выбора и разветвлений. В Фортране существуют два вида операторов условия: логический и арифметический.

2.9.1. Логический оператор условия

Общая форма логического оператора условия имеет следующий вид:

IF (<*условие*>) THEN

<*операторы 1*>

ELSE

<*операторы 2*>

END IF

Действие оператора условия заключается в следующем: ЕСЛИ условие удовлетворяется, ТО действие программы передается группе операторов, следующих после THEN, в противном случае (ИНАЧЕ) выполняются операторы, следующие за ELSE. Условие всегда заключается в скобки.

В программах может использоваться сокращенная конструкция (без использования блока ELSE, также может отсутствовать THEN). Если в сокращенной конструкции условие не удовлетворяется, то действие программы передается оператору, следующему за блоком оператора условия, т.е. все операторы, следующие после THEN, игнорируются.

В условиях, следующих за оператором IF, происходит сравнение значений двух выражений или переменных. Эти выражения разделяются операцией отношения (табл. 2.3).

Таблица 2.3

Математическая операция	Запись на Фортране
$>$	.GT.
$\geq$	.GE.
$=$	.EQ.
$\neq$	.NE.
$\leq$	.LE.
$<$	.LT.

Пример 2.42.

```
IF ( A .GT. B ) Y = SIN(X)
```

В этом примере используется сокращенная конструкция (без использования THEN). Если A больше B, то Y присваивается значение sin(x). Затем после этого оператора будет выполняться следующий за ним. Если A меньше или равно B, то оператор присваивания Y= игнорируется и выполняется оператор, следующий после IF.

Пример 2.43.

```
IF ( z .EQ. 0. ) THEN
  PRINT *,' Z равно 0 '
ELSE
  PRINT *,' Z не равно 0 '.
END IF
```

В этой записи при $z = 0$ на экран будет выведено сообщение 'Z равно 0', во всех остальных случаях – сообщение 'Z не равно 0'.

Кроме операций отношения используют и логические операторы, с помощью которых можно организовать более сложные условия (табл. 2.4).

Таблица 2.4

Логическая операция	Запись на Фортране
'И'	.AND.
'ИЛИ'	.OR.
'НЕТ'	.NOT.

Пример 2.44.

```
IF ( x .GE. 1.0 .AND. x .LE. 3.0) THEN
  PRINT *,' X принадлежит отрезку '
ELSE
  PRINT *,' X не принадлежит отрезку '
END IF
```

В этой записи при одновременном выполнении условий $x \geq 1$ и $x \leq 3$ (т.е. $x \in [1,3]$) на экран будет выведено сообщение 'X принадлежит отрезку', во всех остальных случаях – сообщение 'X не принадлежит отрезку'.

Пример 2.45. Показать решение примера 2.44, используя сокращенную форму оператора условия.

Решение.

```
IF (x.GE.1.0 .AND. x.LE.3.0) PRINT *,'X принадлежит отрезку'
IF (x.LT.1.0 .OR. x.GT.3.0) PRINT *,'X не принадлежит отрезку'
```

Пример 2.46. Записать условие, согласно которому если $a+b \leq c < 5\cos(x)$, то на экран выводятся значения *a*, *b*, *x*.

```
IF (A+B .LE. C .AND. C .LT. 5.*COS(X))  PRINT *,  A, B, X
```

В этом примере сначала определяется истинность логического выражения, стоящего в скобках ($a+b \leq c < 5\cos(x)$). Если условия в скобках соблюдаются, то на экран выводятся значения *a*, *b*, *x*. Если условия в скобках не соблюдаются, то действие передается на оператор, следующий за оператором условия.

Пример 2.47. Составить программу для вычисления площади треугольника по формуле Герона $S = \sqrt{p \cdot (p-a) \cdot (p-b) \cdot (p-c)}$, где $p = \dfrac{a+b+c}{2}$.

Алгоритм

При выполнении программы предусмотрим:

1. Ввод исходных данных (значения сторон треугольника *a*, *b*, *c*).
2. Вычисление полупериметра p и площади S
3. Вывод значения площади на экран.

Программа на языке Фортран *(с пояснениями)*

```
!    Программа вычисления площади треугольника
```

```
REAL  a, b, c, p, S
```

В этом месте программы типы переменных определяются как вещественные.

```
WRITE (5,*)' Введите стороны треугольника A, B, C '
```

На экран (цифра 5) выводится текст-подсказка ***'Введите стороны треугольника A, B, C'****. Символ * означает, что вывод бесформатный (произвольный).*

```
READ (5,*) a, b, c
```

*Данная строка расшифровывается так: ввести с клавиатуры числовые данные для переменных a, b, c. Символ * – бесформатный упрощенный ввод.*

Вычисляется значение полупериметра:

```
p=(a+b+c)/2
```

Вычисляется значение площади треугольника:

```
S=SQRT (p*(p-a)*(p-b)*(p-c))
WRITE (5,1) a, b, c, S
```

На экран (цифра 5) по формату (оператор формата находится после метки 1) выводится информация, находящаяся в списке вывода за скобками оператора WRITE.

```
1 FORMAT (1x, ' Площадь треугольника со сторонами ', 3F5.2, ' равна', F6.3)
```

При использовании этого оператора на экране появится текст Площадь треугольника со сторонами, *затем – численные значения переменных a, b, c, значения которых выводятся по формату* F5.2, *затем текст* равна, *и численное значение переменной S выводится по формату* F6.3.

```
END
```

2.9.2. Арифметический оператор условия

Общая форма арифметического оператора условия имеет следующий вид:

IF (<*арифметическое выражение*>) m_1, m_2, m_3 ,

где m_1, m_2, m_3 – метки операторов, на которые будет передаваться управление программой.

Арифметический оператор условия работает следующим образом:

1) вычисляется арифметическое выражение в скобках;

2) вычисленное значение сравнивается с нулём: если оно меньше нуля, то управление передаётся на метку m_1, если значение равно нулю, то управление передаётся на метку m_2, если значение больше нуля – на метку m_3.

С помощью IF арифметического, к примеру, можно проверять подкорневое выражение, и если оно окажется отрицательным (по законам математики нельзя извлечь корень квадратный из отрицательного числа), то управление передастся на нужную метку и прерывания выполнения программы при попытке вычисления корня из отрицательного числа не произойдёт.

Примечания:

1. В арифметическом операторе условия всегда должны быть три метки, две из них могут быть одинаковыми.

2. Метки могут быть расположены в любом месте программы (выше или ниже оператора IF).

3. Рекомендуется ставить метку перед оператором, следующим после IF арифметического.

4. IF арифметический не может быть последним оператором в цикле DO.

5. Если в качестве условия имеется неравенство, то его необходимо привести к виду, при котором происходит сравнивание выражения с нулём. Например, выражение $a^2 + b^2 > r$ *следует привести к виду* $a^2 + b^2 - r > 0$.

Пример 2.48.

```
      IF ( z ) 10, 20, 10
10    WRITE (*,*)  ' Z не равно 0 '
      ...    ...
      GOTO  30
20    WRITE (*,*)  ' Z равно 0 '.
      ...    ...
30    ...
```

В этой записи при $z = 0$ программа перейдет к метке 20 и на экран будет выведено сообщение 'Z равно 0', во всех остальных случаях ($z < 0$ или $z > 0$) программа перейдет к метке 10 и выведет сообщение 'Z не равно 0'. Следует обратить внимание на оператор GOTO (особенности его работы рассмотрены в п. 2.11.1). В этом фрагменте программы он используется для пропуска группы операторов, стоящих после метки 20. Если не использовать этот оператор, то на экране будут выведены два сообщения одновременно.

Пример.2.49. Доработаем программу примера 2.47, предусмотрев проверку исходных данных на корректность (если значения длин сторон треугольника меньше или равны нулю или если самая длинная сторона больше, чем полупериметр, то треугольник не существует) при помощи арифметического оператора условия.

В п. 1 алгоритма примера 2.47 после ввода исходных данных предусмотрим их проверку на корректность.

Программа на языке Фортран *(с пояснениями)*

```
!     Программа вычисления площади треугольника
      REAL  a, b, c, p, S
!     Ввод исходных данных
10    WRITE (*,*)' Введите стороны треугольника a, b, c '
      READ (*,*) a, b, c
!     Проверка исходных данных на корректность
      IF (a) 10, 10, 11
```

Проверка введенных исходных данных: сторона треугольника не может быть отрицательной или равна нулю.

```
11    IF (b) 10, 10 ,12
12    IF (c) 10, 10, 13
```

Вместо этих трех операторов можно использовать один:
IF (AMIN1(a,b,c)) 10, 10 ,13

```
!     Вычисление полупериметра
13    p=(a+b+c)/2
!     Проверка условия существования треугольника, вычисление площади и вывод результата
      IF ( AMAX1 (a,b,c)–p ) 6, 8, 9
```

Встроенные функции AMIN1, AMAX1 из списка переменных, перечисленных в скобках, выбирают минимальное (максимальное) число. Последний оператор обозначает: если максимальная из сторон равна полупериметру, то управление передается на метку 8. Если максимальная из сторон меньше полупериметра, то идем на метку 6 и вычисляем площадь, иначе – идем на метку 9.

```
6     S=SQRT (p*(p-a)*(p-b)*(p-c))
      WRITE (*,7) S
7     FORMAT (1x, ' Площадь треугольника равна', F6.3)
      GO TO 14
8     WRITE (*,*) ' Площадь треугольника равна нулю '
      GO TO 14
9     WRITE (*,*) ' Такой треугольник не существует '
14    END
```

2.10. Операторы цикла

Цикл является типичной структурой алгоритмов, реализуемых на компьютере. Для организации циклов в алгоритмических языках предусмотрены специальные операторы.

2.10.1. Оператор цикла DO

Оператор цикла DO может быть представлен в одной из следующих форм:

1-я форма

DO <*переменная*>=<*начальное значение*>,<*конечное значение*>,<*шаг*>
<*операторы*>
END DO

2-я форма

DO <*метка*> <*переменная*>=<*начальное значение*>, <*конечное значение*>, <*шаг*>
<*операторы*>
<*метка*> CONTINUE или <*оператор*>

Если шаг равен 1, то <*шаг*> можно опустить. Операторы, расположенные между операторами DO и END DO, образуют тело цикла и выполняются многократно. DO является оператором начала цикла, END DO (CONTINUE) – оператор конца цикла.

Выполнение цикла, образованного оператором DO, заключается в следующем: переменной присваивается начальное значение, и она сравнивается с конечным значением. Если значение переменной при положительном шаге не больше (или при отрицательном шаге не меньше) конечного значения, то выполняются операторы тела цикла и по последнему оператору цикла (например, END DO, CONTINUE) осуществляется возврат к началу цикла. К текущему значению переменной прибавляется шаг (со своим знаком), и снова проверяется условие. Если условие удовлетворяется, то тело цикла выполняется повторно. В противном случае происходит выход из цикла и переход к оператору, следующему за последним оператором цикла. Так как первая проверка условия выхода из цикла осуществляется до первого выполнения тела цикла, то возможна ситуация, когда тело цикла не будет выполнено ни разу.

Примечания:

1) после окончания цикла переменная цикла всегда сохраняет значение большее, чем конечное значение цикла;

2) в случае выхода из цикла до его завершения переменная сохранит свое текущее значение.

Пример 2.50.

```
DO J = 2 , 20 , 2
  PRINT  *, J, J*J
END DO
```

В приведенном примере переменная J принимает значения 2, 4, 6, 8, ..., 20. В результате выполнения цикла на экран будут выведены значения J и J^2.

Используя вторую форму, этот цикл можно записать еще двумя способами:

```
      DO 10 J = 2 , 20 , 2
        PRINT  *, J, J*J
10    CONTINUE
```

```
      DO 10 J = 2 , 20 , 2
10    PRINT  *, J, J*J
```

Цикл можно организовать также с помощью оператора условия (см. пример 2.55).

2.10.2. Оператор цикла DO WHILE

Наряду с циклом DO может использоваться оператор цикла DO WHILE. Он также может быть записан двумя способами:

1-й способ

```
DO WHILE (<условия>)
<операторы>
END  DO
```

2-й способ

```
DO <метка> WHILE (<условия>)
<операторы>
<метка> <оператор>
```

Условия в операторе DO WHILE записываются так же, как и в логическом операторе условия IF.

Следует обратить внимание на то, что при написании программ следует избегать зацикливания (бесконечного цикла). Для этого необходимо организовывать цикл так, чтобы переменная, входящая в условие цикла, изменяла в нем свое значение.

Пример 2.51. Организовать цикл, приведенный в примере 2.50, с помощью оператора DO WHILE.

1-й способ

```
J=2
DO WHILE ( J .LE. 20)
  WRITE (*,*) J, J*J
  J=J+2
END DO
```

2-й способ

```
J=2
DO 20 WHILE ( J .LE. 20)
  WRITE (*,*) J, J*J
20      J=J+2
```

2.10.3. Оператор выхода из цикла

Для прекращения выполнения некоторого блока цикла программы или экстренного выхода из него служит оператор EXIT.

Пример 2.52.

```
INTEGER i
REAL  A(100), pr
... ... ... ...
pr = 1.0
DO i = 1, 100
  IF ( A(i) .EQ. 0.0 )  EXIT
  pr = pr * A(i)
END DO
```

В приведенном фрагменте программы вычисляется произведение элементов массива (ввод элементов массива опущен). Если хотя бы один элемент равен нулю, то происходит выход из цикла.

2.11. Операторы перехода

2.11.1. Оператор безусловного перехода GOTO

Операторы безусловного перевода используются для быстрого перехода к другим операторам программы. Общий вид оператора

GOTO < *метка* >

Этот оператор передает управление первому оператору после указанной метки.

Пример 2.53.

```
2     READ (*,*) x
      PRINT *, x, SIN(x)
      GOTO 2
```

В результате выполнения программа сначала затребует ввод значения x, затем выведет значения x и $\sin(x)$ на экран и после этого вновь вернется к запросу ввода значения x.

2.11.2. Вычисляемый оператор перехода GOTO

Общий вид вычисляемого оператора GOTO

GOTO (m_1, m_2, m_3,..., m_n) *<арифметическое выражение>* ,

где m_1, m_2, m_3,..., m_n – метки операторов, на которые будет передаваться управление программой; значение арифметического выражения – целое число.

При выполнении оператора сначала вычисляется значение арифметического выражения. Если полученное значение равно 1, осуществляется переход к *метке* m_1; если 2 – то к *метке* m_2 и т. д. Если значение выражения меньше 1 или больше числа указанных меток, то управление передается оператору, следующему за оператором GOTO.

Пример. 2.54.

```
      GOTO (4,5,6) K
7     PRINT K
...   ...
5     PRINT K**2
```

Если значение K равно 2, то управление будет передано метке 5 (будет выведено значение K^2). Если значение выражения меньше 1 или больше 3, то следующим будет выполняться оператор после метки 7 (будет выведено значение K).

2.11.3. Оператор условного перехода

Для передачи в зависимости от условия используется оператор перехода IF–GOTO, который образуется комбинацией двух операторов – оператора условия IF и оператора безусловного перехода GOTO. Общая форма записи этого оператора

IF (<*условие*>) GOTO < *метка* >

Действие оператора условия заключается в следующем: если условие удовлетворяется, то действие программы передается группе операторов, стоящих после указанной метки.

Пример.2.55. Цикл, приведенный в примере 2.50, эквивалентен следующему:

```
      J = 2
30    PRINT  *, J, J*J
      J=J+2
      IF ( J <= 20 ) GOTO 30
```

2.12. Работа с массивами

Перед обработкой массивов их следует описать (см. п. 2.3.6).

2.12.1. Ввод массивов

Предположим, что в программе описаны массивы

```
DIMENSION A(10), B(8), C(5,6), D(4,3)
```

Существуют 4 способа ввода массивов.

Первый способ ввода массивов

```
READ (*,*) A
```

означает «ввести все числовые значения массива *A* в том количестве, сколько их описано в операторе DIMENSION ».

Недостатки	Достоинства
необходимо вводить числа до тех пор, пока массив не заполнится	самый простой способ записи ввода массивов
если вводится несколько массивов, то сначала полностью заполнится первый массив, затем без предупреждения будут заполняться остальные	числа с клавиатуры можно вводить: ✓ в строчку – через пробел или запятую; ✓ нажимая «Enter» после каждого введённого числа
программа не начнёт выполняться, пока не будут введены все элементы массивов	

Если имеется массив из 10 элементов, а при вводе в строке будет введено больше 10 элементов, то компьютер проигнорирует все лишние числа. Если же, наоборот, будет введено меньше 10 элементов, то он будет ожидать ввода остальных чисел.

Второй способ ввода массивов

В этом способе можно использовать внешний цикл DO:

для одномерных массивов

```
      DO 18 I=1,10
18    READ (*,*) A(I)
```

для двумерных массивов

```
      DO 18 I=1,5
      DO 18 J=1,6
18    READ (*,*) C(I,J)
```

или неявный цикл DO, где I изменяется от 1 до 10 с шагом 1:

для одномерных массивов

```
READ (*,*) ( A(I), I=1,10 )
```

для двумерных массивов

```
READ (*,*) ( ( C(I,J), J=1,6), I=1,5)
```

В последней записи ввод элементов будет осуществляться по строкам. Для ввода элементов массива по столбцам можно поменять циклы для переменных, т.е. записать так:

```
READ (*,*) ( ( C(I,J), I=1,5), J=1,6)
```

Недостаток	Достоинство
нет возможности задавать количество вводимых элементов (т.к. это количество указывается в программе как постоянная величина), и для того,чтобы можно было ввести любое их количество, необходимо делать изменения в программе	таким способом можно ввести весь массив целиком или часть этого массива, что весьма удобно использовать в программе, когда массив необходимо заполнить не полностью, а частично

Третий способ ввода массивов

для одномерных массивов

```
READ (*,*) N
READ (*,*) ( A(I), I=1,N )
```

или аналогично

```
READ (*,*) N, ( A(I), I=1,N )
```

Сначала вводится размер массива N, а затем вводятся N элементов массива.

Для двумерных массивов

```
READ (*,*) M, N
READ (*,*) ( ( C(I,J), J=1,N), I=1,M)
```

Здесь сначала вводится количество строк M массива, затем количество столбцов N, и, наконец, вводятся $M \times N$ элементов массива.

Достоинства:

✓ используя такой ввод массивов, можно создать программы не для конкретных случаев, а для разных, когда количество данных и сами данные задаёт пользователь;

✓ данный способ позволяет ввести только необходимое количество элементов при любых размерах массивов.

2.12.2. Вывод массивов

При выводе массивов можно использовать 4 способа

Первый способ вывода массива

```
WRITE (*,*) B
```

При бесформатном выводе массива *B* на экран в строку будут выведены значения всех элементов массива, количество которых было описано в операторе DIMENSION. В каждой строке печатаются по 4 числа вида (при соответствующей настройке – 5 чисел)

```
-0.1000567E-01
```

т.е. с точностью до 7 знаков после запятой.

Недостаток этого способа заключается в плохой наглядности.

Второй способ вывода массива

Для одномерных массивов с использованием внешнего цикла DO:

```
      DO 11 I=1,N
11    WRITE (*,*) B(I)
```

При такой записи все числа будут выведены в столбец (по 1 числу в строке).

или неявного цикла DO

```
WRITE (*,*) ( B(I), I=1,N )
```

Для этой записи все числа будут выведены в строку (по 4 числа в строке).

Для двумерных массивов с использованием неявного цикла

```
WRITE (*,*) ( ( D(I,J), J=1,N), I=1,M)
```

Здесь также все числа будут выведены в строку (по 4 числа в строке).

Достоинства:

- ✓ позволяет вывести только часть массива;
- ✓ количество выводимых элементов задаётся пользователем.

Недостаток – элементы выводятся на экран по 5 чисел в каждую строку.

Третий способ вывода массива

```
WRITE (*,*) ( ' B( ', I ,' )= ', B(I), I=1,N )
```

Здесь используется неявный цикл типа DO, по которому происходит вывод имени массива с помощью текстовой константы, а за ним в скобках указывается номер выведенного элемента и через знак '=' само численное значение элемента массива:

A(1) = число A(2) = число и т.д.

Недостаток – весь массив будет разбросан по экрану дисплея.

Достоинство – имеется возможность вывода каких-либо пояснений.

Четвертый способ вывода массива

Самый удобный способ вывода массива – использовать вывод по формату с повторителем:

```
      WRITE (*,12) ( A(I), I=1,10 )
12    FORMAT (10 (2x, F5.2) )
```

В результате выполнения этой записи на экран будут выведены 10 чисел по формату F5.2 через 2 пробела.

```
      WRITE (*,13) ( (D(I,J), J=1,3), I=1,4)
13    FORMAT ( 3 (2x, F5.2) )
```

В результате выполнения этой записи на экран будет выведен массив, причем в 4 строках по формату F5.2, будет выведено по 3 числа,расположенных через 2 пробела.

2.12.3. Примеры обработки массивов

Пример 2.56. Составить программу вывода значений массивов *A* и *B* размерностью 10 и массива *C*, каждый элемент которого является результатом сложения элементов массивов *A* и *B*, в виде таблицы

```
==================================
| N | Массив A | Массив B | Массив C |
```

Программа на языке Фортран *(с пояснениями)*

```
!       Программа вывода массивов в виде таблицы
        DIMENSION  A(10), B(10), C(10)
        READ(*,*) A
        DO  1  I=1, 10
        WRITE (*,2) I
2       FORMAT ( 1X , ' Введите элемент B( ' , I2 , ' )= ' \ )
```

Символ «\» (обратный слэш) отменяет переход на новую строку.

```
1       READ (*,*) B(I)
        DO  3  I=1, 10
3       C(I)=A(I)+B(I)
!       Вывод шапки таблицы
        WRITE(*,4)
4       FORMAT (1X , 34 (' = ') )
        WRITE(*,5)
5       FORMAT(1X, '| N | МАССИВ A | МАССИВ B | МАССИВ
C |')
        DO  6  I=1, 10
6       WRITE(*,7) I,A(I),B(I),C(I)
7       FORMAT (1X, 38 ('-') / 1X , ' | ', I2 ,' | ', 3 ( F9.3 ,' | ') )
```

Выводится линия из символов «-», затем осуществляется переход на другую строку, печатается номер и по одному значению элементов трех массивов A, B, C.

```
WRITE (*,4)
```

Снизу таблицу подчеркнули двойной чертой

```
END
```

Пример 2.57. Составить программу вычисления произведения элементов массива *D*(20), стоящих на нечётных местах.

Программа на языке Фортран

```
        REAL D(20)
        READ (*,*) D
        p=1.
        DO  8  I=1, 20, 2
```

```
8       p=p*D(I)
        WRITE (*,9) p
9       FORMAT (2x, 'Произведение элементов равно', F7.3)
        END
```

Программа будет по циклу брать только нечётные элементы (1, 3, 5...) и перемножать их друг с другом. Вычисление произведётся до 19 элемента и остановится, т.к. 19 – последний нечётный элемент этого массива. При $I = 19$ завершится последний цикл. Но затем к переменной I будет добавлена величина шага, равная 2, и новое значение $I = 21$ будет сравниваться с конечным значением 20. Поскольку I будет больше конечного значения, то осуществляется выход из цикла и управление будет передано оператору, следующему после последнего в области цикла, т.е. оператору WRITE. Следует обратить внимание, что после окончания цикла переменная цикла имеет значение большее, чем конечное значение в операторе цикла DO.

Пример. 2.58. Составить программу нахождения наибольшего элемента массива $B(10,20)$ и его индексов.

Программа на языке Фортран

```
 REAL  B (10,20), maxb
 READ (*,*) ( ( B(I,J), J=1,20), I=1,10)
 maxb=B(1,1)
 m=1
 n=1
 DO  I=1,10
   DO  J=1,20
     IF(B(I,J).GT.maxb) THEN
       maxb = B(I,J)
       m = i
       n = j
     END  IF
   END  DO
 END  DO
 WRITE (*,13) maxb, m, n
13 FORMAT (2x, 'Наибольший  элемент массива  равен', F7.3/
 'его индексы (' , I2 , ' , ' , I2 , ' ) ' )
  END
```

2.13. Подпрограммы

При написании программы возникают случаи, когда некоторая одинаковая совокупность действий должна выполняться в нескольких различных местах программы. Повторение каждый раз группы операторов, реализующих эти действия, приводит к увеличению объема программы. Чтобы избежать повторений и уменьшить объем программы, указанную группу операторов можно записать в программе один раз и обращаться к ней, когда в этом возникает необходимость.

Обособленную группу операторов, которую можно выполнять многократно, обращаясь к ней из различных мест программы, называют подпрограммой. Чтобы подпрограмма при обращении к ней выполнялась каждый раз с новыми данными, ее нужно составить в общем виде, а исходные данные для работы передавать в переменные подпрограммы (они называются формальными переменными) перед обращением к ней.

Если, например, в программе требуется решить три квадратных уравнения с различными коэффициентами, то алгоритм нахождения корней квадратного уравнения целесообразно оформить в виде подпрограммы, используя для обозначения коэффициентов переменные. Перед каждым обращением к подпрограмме нужно задать этим переменным числовые значения, соответствующие коэффициентам решаемых уравнений. Таким образом, использование подпрограмм приводит к уменьшению общего количества операторов в программе, и, следовательно, для размещения программы требуется меньше памяти.

Подпрограммы обладают также некоторыми другими преимуществами. Использование подпрограмм оптимизирует структуру программы. Кроме того, облегчается отладка программы, так как работа каждой подпрограммы может быть проверена по отдельности. Многие подпрограммы имеют дополнительную ценность, поскольку ими может воспользоваться не только тот, кто написал подпрограмму, но и другие лица.

В Фортране подпрограмма оформляется как группа операторов, которая должна выполняться при обращении к ней, и может записываться несколькими способами:

- ✓ непосредственно в основной (головной) программе;
- ✓ в виде отдельного блока после головной программы;
- ✓ в виде отдельного файла.

Общая структура программы может быть представлена, как на рис. 2.4.

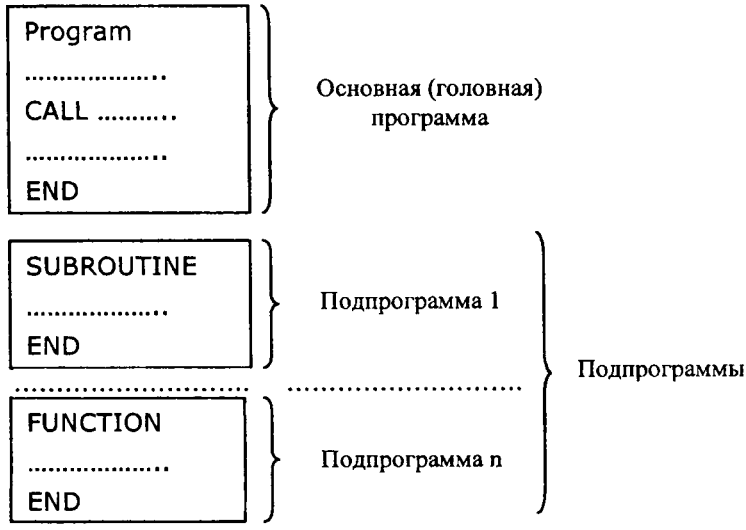


Рис. 2.4. Структура программы

В Фортране к подпрограммам можно отнести:

✓ оператор-функцию, задаваемую пользователем (результатом ее выполнения является имя функции);

✓ подпрограмму-функцию FUNCTION (результатом ее выполнения является имя подпрограммы);

✓ подпрограмму-процедуру SUBROUTINE (результатом ее выполнения являются некоторые переменные, определяемые пользователем).

В результате выполнения подпрограмма-процедура может возвращать от 0 до сколь угодно большого количества значений, а подпрограмма-функция – только одно.

2.13.1. Оператор-функция

Помимо стандартных функций (см. прил. 3) в программе можно определить и далее использовать другие (нестандартные) функции

(оператор-функции). Они используются в тех случаях, когда по одной и той же формуле необходимо производить большое количество вычислений с различными данными. Оператор-функция записывается в самом начале программы до первого выполняемого оператора. Общий вид записи оператор-функции

<имя> (<список параметров>) = <арифметическое выражение>

Арифметическое выражение в правой части должно обязательно содержать хотя бы один из формальных параметров, указанных в скобках, но может содержать также и другие переменные, общие для всей программы. Значения этих переменных будут определяться при выполнении программы.

Пример 2.59.

```
Z (X,Y) = SQRT(X+Y) + ABS(Y)*2.*D
```

Задавая различные значения X и Y, эта оператор-функция всегда будет вычислять арифметическое выражение по заданной формуле.

Обращение к оператор-функции осуществляется путём указания имени функции и записи в скобках фактических параметров.

Пример. 2.60. Вычислить с использованием оператор-функции арифметическое выражение

```
Q=2.*SIN(X)+Z(A,B)+Z(3.2, COS(C))**2
```

При обращении к оператор-функции *Z* фактические параметры *A* и *B* будут подставляться вместо формальных параметров *X* и *Y*, затем будет осуществляться вычисление арифметического выражения оператор-функции. Результатом вычисления будет являться одно число, и это одно число возвращается на то место в программе, откуда осуществляется обращение к оператор-функции.

Затем идёт повторное обращение к оператор-функции *Z*. Во втором случае фактическими параметрами являются константа (3,2) и выражение (COS(*C*)). Константа подставится вместо *X*, а вычисленное значение выражения будет подставляться вместо *Y*. Произво-

дится новое вычисление арифметического выражения оператор-функции. В результате получают новое число, которое возвращается в программу на то место, откуда оператор-функция вызывалась второй раз.

Переменная *Z* вычислится как 2*SIN(*X*) + результат *Z*1+ результат *Z*2.

В одной программной единице может быть большое количество оператор-функций. Формальными параметрами могут быть только переменные. Фактическими параметрами могут быть переменные, константы, выражения и элементы массивов.

Примечание: список формальных параметров оператор-функции должен соответствовать списку фактических параметров при обращении к ней по очерёдности следования и типу данных.

Имя оператор-функции должно соответствовать правилам формирования имён языка Фортран, то есть все оператор-функции считаются вещественного типа, за исключением тех, имя которых начинается на буквы *I, J, K, L, M, N.*

Пример 2.61. Составить программу для вычисления следующих выражений:

$$d = \sqrt{a^2 + b^2},$$

$$e = \sqrt{a^2 + d^2} + 2\sqrt{d^2 + b^2},$$

$$f = \sqrt{a^2 + (3{,}5\sqrt{e^2 + d^2})^2},$$

$$q = \sqrt{f^2 + \sin^2 x + \cos^2 y}.$$

Как можно заметить, во всех выражениях можно выделить общую часть, которую можно записать в виде $\sqrt{z^2 + t^2}$. Эту общую часть можно записать в виде оператор-функции, к которой часто обращаются при вычислениях. Это избавляет нас от необходимости громоздкой записи для вычисления каждой формулы.

Программа на языке Фортран *(с пояснениями)*

```
!       Программа № 3 – Вычисление выражений
        REAL  a, b, d, x, y, z, t, e, f, q
!       Описываем оператор-функцию
        FUN(z,t) = SQRT( z*z+t*t )
!       Ввод исходных данных
          RITE (*,*) ' Введите исходные данные '
          EAD (*,*) a, b, x, y
          d = FUN(a,b)
```

Обращаемся к оператор-функции FUN. Фактические параметры A и B при обращении подставляются вместо формальных Z и T, и по ним производится вычисление. Результат вычислений в виде вещественного числа возвращается на место обращения к оператор-функции и затем присваивается переменной D.

```
          e = FUN(a,d) + FUN(d,b)*2.
          f = FUN(a,3.5*FUN(e,d))
          q = FUN(f,FUN(SIN(x),COS(y)))
```

При первом обращении к оператор-функции фактическими параметрами являются выражения SIN(X) и COS(Y). При втором обращении фактическими параметрами будут F и результат первого обращения к оператор-функции.

```
          WRITE (*,1) d, e, f, q
1         FORMAT(2X, 'D=', F6.3,' E=', F6.3,' F=', F6.3,' Q=', F6.3)
          END
```

2.13.2. Подпрограмма-функция FUNCTION

В общем виде подпрограмма-функция записывается

```
<тип> FUNCTION <имя> (<список формальных параметров>)
<операторы>
<имя> = <выражение>
RETURN
END
```

В качестве типа функции могут использоваться REAL, INTEGER, COMPLEX, CHARACTER.

Список формальных параметров может состоять из имён переменных, имён массивов и символов «*» (для передачи метки в вызываемую подпрограмму-функцию). Внутри подпрограммы-функции могут быть любые операторы языка Фортран (операторы описания типов, ввода-вывода данных, циклов, условия и др.). Подпрограмма-функция может содержать свои операторные функции или обращаться к другим подпрограммам.

Примечание: особенностью работы подпрограммы-функции является то, что результатом вычисления является одно число, и этот результат должен быть присвоен имени функции.

Оператор RETURN служит для выхода из подпрограммы FUNCTION и передачи результата вычисления в головную программу на то место, откуда произошло обращение к подпрограмме-функции.

Чтобы обратиться к подпрограмме-функции, необходимо (как и в случае с оператор-функцией) указать имя подпрограммы-функции и список фактических параметров. Список фактических параметров должен соответствовать списку формальных параметров по количеству, очерёдности следования, типу данных и размерности массивов (размерность массива в подпрограмме-функции не должна превышать размерности соответствующего массива головной программы). Результатом работы подпрограммы-функции является одно число, и это число возвращается в программу на то место, откуда осуществляется вызов.

Пример 2.62. Составить программу для вычисления выражений примера 2.61, используя подпрограмму-функцию.

Программа на языке Фортран *(с пояснениями)*

```
!       Программа № 3 – Вычисление выражений
        REAL  a, b, d, x, y, z, t, e, f, q
!       Ввод исходных данных
        WRITE (*,*) ' Введите исходные данные '
        READ (*,*) a, b, x, y
          d = FUN(a,b)
```

```
      e = FUN(a,d) + FUN(d,b)*2.
      f = FUN(a,3.5*FUN(e,d))
      q = FUN(f,FUN(SIN(x),COS(y)))
      WRITE (*,1) d, e, f, q
1     FORMAT(2X, 'D=',F6.3,' E=',F6.3,' F=',F6.3,' Q=',F6.3)
    END

!     Подпрограмма-функция
      REAL FUNCTION FUN (Z,T)
      REAL  z, t
        FUN = SQRT( Z*Z+T*T )
      RETURN
      END
```

2.13.3. Подпрограмма-процедура SUBROUTINE

Подпрограмма-процедура – основной объект программирования на языке Фортран. Любая большая задача разбивается на подпрограммы, эти программы отлаживаются и тестируются по отдельности. Головная программа служит лишь для ввода исходных данных, поочерёдного обращения к различным подпрограммам, записи и обработки полученных результатов. В программировании признан рациональным модульный принцип построения программ (разбиением на подпрограммы).

Общий вид записи подпрограммы-процедуры:

SUBROUTINE <*имя*> (<*список формальных параметров*>)
<*операторы*>
RETURN
END

Обращение к подпрограммам осуществляется с помощью оператора

CALL <*имя подпрограммы*> (<*список фактических параметров*>)

Оператор CALL вызывает только одну подпрограмму. Обращение с помощью оператора CALL сразу к нескольким подпрограм-

мам недопустимо. При выполнении этого оператора управление из головной программы передаётся в подпрограмму, где происходят все вычислительные действия. При достижении оператора RETURN осуществляется возврат в то место основной программы, из которого произошло обращение к подпрограмме, а именно к оператору, следующему за оператором CALL. Подпрограммы могут содержать обращения к другим подпрограммам.

Отличия подпрограммы-процедуры от подпрограммы-функции:

✓ имя подпрограммы-процедуры не имеет значения типа;

✓ результатом работы подпрограммы-процедуры могут быть одно или несколько чисел, массив, несколько массивов. Эти результаты могут передаваться или не передаваться в головную программу;

✓ список формальных параметров в подпрограмме-процедуре может отсутствовать.

Правила соответствия формальных и фактических параметров те же, что и для подпрограмм-функций.

Примечания:

✓ *списки имён переменных и имена меток каждой подпрограммы независимы друг от друга и могут повторяться;*

✓ *вычислительный процесс каждой подпрограммы также независим от головной программы;*

✓ *одна подпрограмма может обращаться к нескольким другим подпрограммам и может содержать оператор-функции;*

✓ *стыковка между подпрограммами осуществляется только через список формальных – фактических параметров или посредством оператора COMMON.*

В предлагаемом пособии способ передачи данных в подпрограмму с помощью оператора COMMON не рассматривается.

2.14. Работа с внешними файлами

При работе с программой возникает необходимость многократно использовать одни и те же исходные данные. Для того чтобы данные каждый раз не вводить с клавиатуры, гораздо удобнее получать их из внешних файлов. Кроме того, во внешних файлах очень удобно сохранять промежуточные результаты для их последующего использования.

Имя файла данных складывается из названия (не более 8 симво-

лов) и расширения (не более трех), разделенных точкой. Как правило, название файла и его расширение выбираются таким образом, чтобы они свидетельствовали о хранящейся информации. Файлам исходных данных обычно присваивают расширения TXT и DAT, а файлам результатов – TXT и REZ. При формировании файлов допускается не указывать расширение, тогда имя файла состоит из названия.

Последовательность работы с внешними файлами такова:

1. Открыть внешний файл.
2. Прочитать данные из файла или записать данные в файл.
3. Закрыть внешний файл.

Пример. 2.63. Примеры имен файлов исходных данных

DATA.TXT *MASSIV.DAT* *INDAT*

Пример 2.64. Примеры имен файлов результатов

REZULT.TXT *MASSIV.REZ* *ITOG*

2.14.1. Оператор открытия файла OPEN

Открытие файла данных осуществляется при помощи оператора

$$\text{OPEN} \quad (n, \quad \text{FILE='<}\textit{имя файла}\text{>'}, \quad \text{ACCESS=}\left\{\begin{matrix}\textit{Direct}\\ \\ \textit{Sequential}\end{matrix}\right\},$$

$$\text{ACTION=}\left\{\begin{matrix}\textit{Read}\\ \\ \textit{Write}\end{matrix}\right\}),$$

где n – номер канала связи (целое число большее 0);

ACCESS – вид доступа к файлу (Direct указывается для файлов прямого доступа, Sequential (указывается по умолчанию) – для файлов последовательного доступа);

ACTION – вид действия над файлом (READ – файл для считывания информации, WRITE – для записи информации, READWRITE

(указывается по умолчанию) – для считывания и для записи).

В операторе открытия файла могут использоваться и другие опции (BLANK, BLOCKSIZE, CARRIAGECONTROL, DELIM, ERR, FORM, IOFOCUS, IOSTAT, PAD, POSITION, RECL, SHARE, STATUS). В связи с простотой рассматриваемых задач эти опции в пособии не описываются.

Например, запись OPEN (1, File=' DATA.TXT') означает, что файл DATA.TXT открыт как файл последовательного доступа (не указана опция ACCESS) и может использоваться как для считывания, так и для записи (не указана опция ACTION).

Примечание: канал связи может быть открыт только для одного файла.

2.14.2. Оператор закрытия файла CLOSE

Закрытие файла данных должно осуществляться после окончания работы с ним при помощи оператора

CLOSE (*n*) ,

где *n* – номер канала связи, использованный для обмена информацией (тот же, что и в операторе OPEN).

Например, запись CLOSE (1) означает, что канал номер 1 закрывается для передачи данных (т.е. доступ к открытому внешнему файлу закрывается).

2.14.3. Примеры работы с файлами

В Фортране имеется возможность работать с файлами данных последовательного и прямого доступа.

Файл прямого доступа характеризуется тем, что доступ к нужной информации можно получить непосредственно, зная только ее расположение в файле. Это делает файлы прямого доступа сложными для использования.

Файл последовательного доступа характеризуется тем, что порядок следования данных в нем определяется последовательностью, в которой данные записываются на диск. Считывание из файла последовательного доступа возможно только в том же порядке, в ко-

тором данные хранятся в этом файле. Пересылка данных в файл последовательного доступа осуществляется при помощи оператора WRITE, считывание из файла – при помощи READ.

Данные, передаваемые в файл при помощи одного оператора WRITE, называются записью. Таким образом, сформированный файл состоит из последовательности записей. Считывание из файла также осуществляется записями, т.е. при пересылке данных в следующий элемент списка оператора READ выбирается следующая запись. Это необходимо учитывать при формировании файла.

После окончания пересылки данных в файл последовательного доступа и выполнения оператора CLOSE на диске формируется признак конца файла.

Пример 2.65. Составить программу для формирования файла последовательного доступа с именем *DATA*. В файл *DATA* должны быть переданы десять чисел, вводимых с клавиатуры. При формировании файла будем нумеровать каждую строку.

```
Real x
OPEN (2, File= 'DATA', ACTION='Write')
DO  I=1, 10
  PRINT *, ' введите ' , I , '-е число'
  READ (*,*)  x
  WRITE (2,*) x
END DO
CLOSE (2)
END
```

Пояснения к примеру. Оператором OPEN открывается файл *DATA* для записи и определяется канал с номером 2 для передачи данных. Далее в цикле по I последовательно вводятся числа и присваиваются переменной *x*. Каждое число сразу после ввода передается в файл *DATA* по каналу с номером 2. Оператор CLOSE закрывает файл.

Пример 2.66. Составить программу, которая формирует массив B из данных, содержащихся в файле *DATA*.

```
DIMENSION B(10)
OPEN (4, File= 'DATA', ACTION='Read')
DO  I=1, 10
READ (4,*) B(I)
PRINT *, B(I)
END DO
CLOSE (4)
END
```

Пояснения к примеру. В 1-й строке описывается массив *B*, состоящий из 10 элементов. Оператор OPEN открывает файл *DATA* для чтения и определяет канал с номером 4 для передачи данных. Далее в цикле для каждого значения *I* последовательно считываются числа и присваиваются соответствующим элементам массива *B*(*I*). Каждое число сразу после считывания выводится в столбец на экран. Оператор CLOSE закрывает файл.

3. ПРИМЕРЫ ВЫПОЛНЕНИЯ КОНТРОЛЬНЫХ ЗАДАНИЙ

Задача 1

Уравнение деформированной оси гибкой нити на интервале (*a, b*) описывается уравнением

$$q(x) = 5{,}91\sin(1{,}2 - x) + 2{,}69\cos(x + 1{,}1).$$

Требуется разработать программу вычисления координат точек нити в *m* промежуточных равноотстоящих точках интервала (*a, b*).

При работе над задачей необходимо выполнить следующие этапы:

1. Описать математическую модель задачи и привести ее графическую интерпретацию.

2. Начертить блок-схему алгоритма задачи.

3. Выбрать форму и формат печати исходных данных и результатов выполнения программы на экран. Предусмотреть при этом вывод фамилии, имени, отчества студента, номера группы и шифра задания.

4. Написать текст программы на алгоритмическом языке Фортран.

5. Выполнить тестовый расчет по исходным данным.

При решении задачи примем следующие значения исходных данных: интервал (4,0; 9,0), количество промежуточных точек $m = 4$.

Решение

1. Математическая модель и графическая интерпретация задачи

Изобразим графическую интерпретацию задачи. Для этого проведем оси координат $x0y$, нанесем точку A с координатами $(a,0)$, точку B $(b,0)$ и изобразим абстрактно функцию $y = q(x)$ (рис. 3.1).

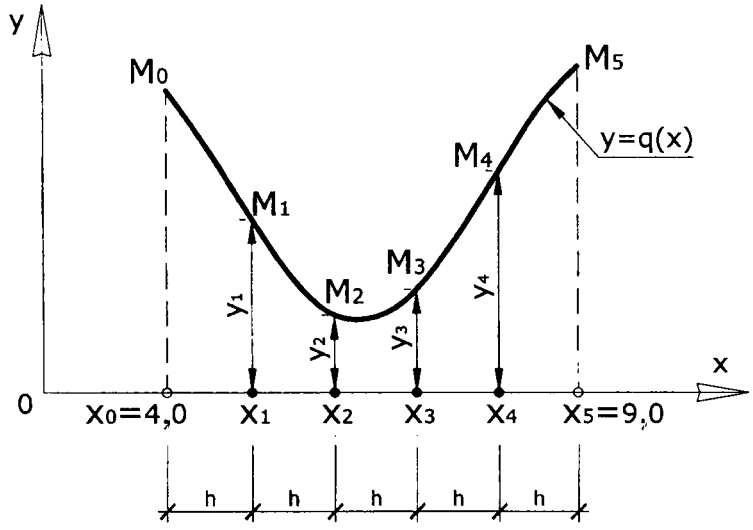


Рис. 3.1. Графическая интерпретация задачи 1

Учитывая, что значения функции будут вычисляться в 4 промежуточных равноотстоящих точках интервала, вычислим расстояния между точками (шаг разбиения интервала):

$$h = \frac{x_b - x_a}{m+1} = \frac{9{,}0 - 4{,}0}{5} = 1{,}0. \qquad (3.1)$$

Разбив интервал (4,0; 9,0) на равные отрезки длиной h, находим абсциссы точек x_1, ... , x_4, в которых мы должны найти ординаты провисания. Графически легко определить, что координаты всех точек $x_0 = 4{,}0$, $x_1 = x_0 + h = 5{,}0$, $x_2 = x_0 + 2h = 6{,}0$, ... , $x_5 = x_0 + 5h = 9{,}0$, т.е., зная величину шага и координату начальной точки x_0, в общем виде их можно записать

$$x_i = x_0 + i \cdot h \text{, где } i = \overline{1,4}. \qquad (3.2)$$

Вычисляя значения функции $y = q(x)$ во всех промежуточных точках x_i (формула (3.2)), получим координаты соответствующих им точек M_i ($i = \overline{1,4}$).

Учитывая вышеизложенное, запишем *математическую модель* задачи в следующем виде:

Вычислить $q(x_i) = 5{,}91\sin(1{,}2 - x_i) + 2{,}69\cos(x_i + 1{,}1)$ для всех $x_i = x_a + i \cdot h = 4{,}0 + i$, где $x_a = x_0 = 4{,}0$ (по условию), $h = 1{,}0$ (формула (3.1)), $i = \overline{1,4}$.

2. Блок-схема алгоритма задачи

Предусмотрим в программе:

– ввод исходных данных;

– вычисление шага разбиения h;

– вывод исходных данных и значение шага для контроля на печать;

– вычисление абсцисс x промежуточных точек и значений функции $y = q(x)$;

– вывод результатов расчета на экран.

Блок-схема алгоритма представлена на рис. 3.2.

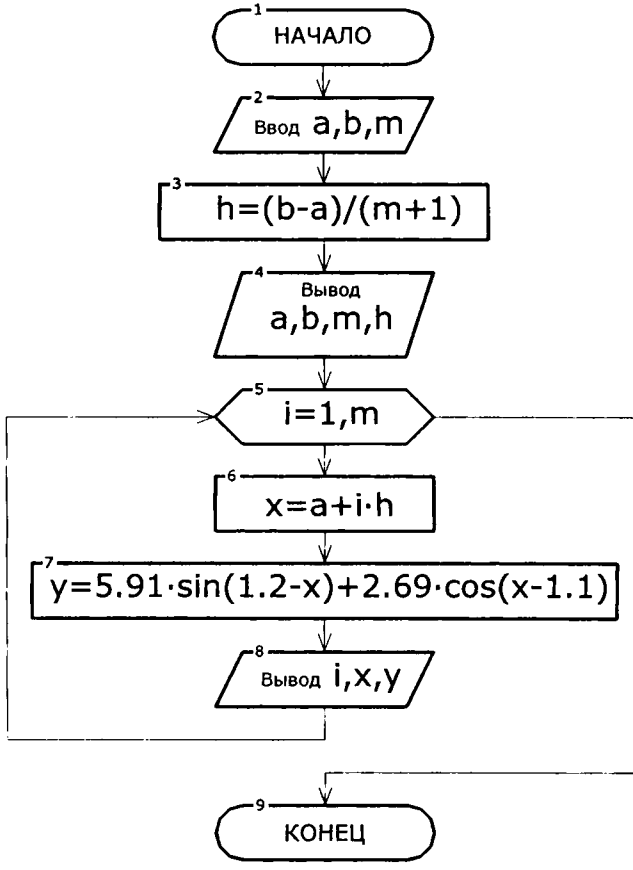


Рис. 3.2. Блок-схема алгоритма задачи 1

Приведем краткое описание блок-схемы. В блоке 2 головной программы производится ввод исходных данных (концов интервала a и b, числа промежуточных точек m). В блоке 3 вычисляется шаг разбиения. В блоке 4 исходные данные выводятся на печать для контроля. Затем в цикле, образованном блоками 5-8, где его счетчик i будет изменяться от 1 до m с шагом 1, последовательно производится вычисление абсциссы x и соответствующей ей ординаты y, выводится на экран номер точки, ее абсцисса и ордината. В блоке 9 программа завершает свою работу.

3. Форма и формат печати исходных данных и результатов выполнения программы

Левый край бумаги

Контрольная работа по информатике

Задача 1

студент группы 312514

Сергеев С.С. шифр 000

Исходные данные:

интервал (04.0, 09.0)

количество точек m=04

шаг разбиения h=1.00

Результаты:

№	X	У
01	04.00	11.111

и т.д.

4. Программа на алгоритмическом языке Фортран

```
      REAL  a, b, h, x, y
      PRINT *, 'Введите интервал (a, b) '
       READ (*,*)  a, b
      PRINT  *, 'Введите количество промежуточных точек  m'
       READ (*,*)  m
       h = ( b - a) / ( m + 1)
      WRITE ( * , 5 ) 'Контрольная работа по информатике'
5       FORMAT ( 5x , a33 )
      WRITE ( * , 6 ) 'Задача 1'
6       FORMAT ( 10x , a11)
      WRITE ( * , 7 ) 'студент группы 312514'
7       FORMAT ( 10x , a21)
      WRITE ( * , 8 ) 'Сергеев С.С.' , 'шифр 000'
8       FORMAT ( 5x , a12 , 6x , a8 )
      WRITE ( * , 9 ) 'Исходные данные:'
```

```
9       FORMAT ( 5x , a16 )
   WRITE ( * , 10 ) 'интервал', a , b
10      FORMAT ( 1x , a8 , 4x , '(' , f4.1 , ';' , f4.1 , ')')
   WRITE ( * , 11 ) 'количество точек', m
11      FORMAT ( 1x , a30 , 1x , 'm = ' , i2)
   WRITE ( * , 12 ) 'Шаг разбиения', h
12      FORMAT ( 1x , a16 , 1x , 'h = ' , f4.2 )
   WRITE ( * , 6 ) 'Результаты:'
   WRITE ( * , 13 )
13      FORMAT ( 7x , '№' , 6x , 'x' , 6x, 'y' )
     DO  i = 1 , m
       x = a + i * h
       y = 5.91 * sin ( 1.2-x ) + 2.69 * cos ( x+1.1 )
       WRITE ( * , 14 ) i , x , y
     END  DO
14      FORMAT ( 6x , I2 , 4x , F5.2 , 4x, F6.3 )
   END
```

5. Тестовый расчет

Сведем вычисления координат точек нити в табл. 3.1. Значения $y_i^{\text{р}}$ будем округлять до трех цифр после десятичной точки.

Таблица 3.1

№ точек (i)	Результаты ручного счета		Результаты расчета на компьютере $y_i^{\text{к}}$	Погрешность, %
	x_i	$y_i^{\text{р}}$		
1	5,0	6,261	Выполняется на лабораторных занятиях	Определяется на лабораторных занятиях
2	6,0	7,729		
3	7,0	2,091		
4	8,0	-5,470		

Примечание: погрешность определяется по формуле

$$\varepsilon_i = \left| \frac{y_i^{\text{к}} - y_i^{\text{р}}}{y_i^{\text{к}}} \right| \cdot 100\ \%.$$

Задача 2

Составить программу вычисления значения функции

$$y = \begin{cases} \dfrac{1}{x}, & x \le 5, \\ \sqrt{18 - x}, & x > 5. \end{cases} \qquad (3.3)$$

При работе над задачей выполнить следующие этапы:

1. Описать математическую модель задачи.
2. Начертить блок-схему алгоритма задачи.
3. Выбрать форму и формат печати исходных данных и результатов вывода программы на экран. Предусмотреть при этом вывод фамилии, имени, отчества студента, номера группы и шифра задания.
4. Написать текст программы на алгоритмическом языке Фортран.
5. Выполнить тестовый расчет по исходным данным.

Решение

1. Математическая модель задачи

Фактически математическая модель задачи описывается формулой (3.3). Проведем исследование области определения функции.

При $x \le 5$ функция $\dfrac{1}{x}$ определена для всех x, кроме $x = 0$. Поэтому при $x \le 5$ и $x = 0$ функция y не имеет значений, а при $x \le 5$ и $x \ne 0$ принимает значение $y = \dfrac{1}{x}$.

При $x > 5$ функция $\sqrt{18-x}$ определена для всех $x \le 18$. Поэтому при $x > 5$ и $x > 18$ функция y не имеет значений, а при $5 < x \le 18$ – $y = \sqrt{18-x}$.

Таким образом, при составлении программы будем учитывать область определения функции.

2. Блок-схема алгоритма задачи

Предусмотрим в программе:

– ввод значения x;

– вычисление значения y (с учетом области определения);

– вывод результатов расчета на экран.

Одна из возможных блок-схем алгоритма изображена на рис. 3.3.

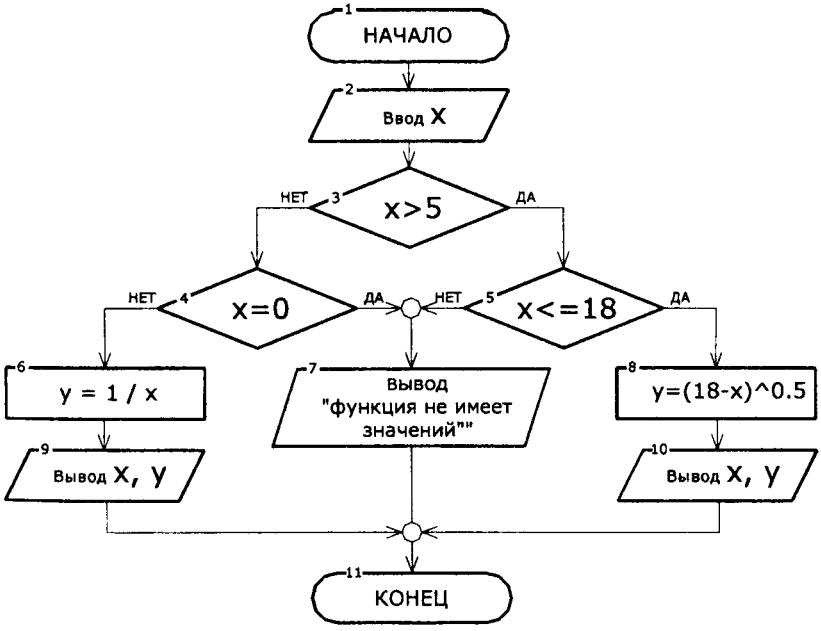


Рис. 3.3. Блок-схема алгоритма задачи 2

Приведем краткое описание блок-схемы. В блоке 2 головной программы производится ввод значения x. В блоке 3 определяется, по какой из функций производить расчет. Если $x \le 5$, то по функции $y = \frac{1}{x}$ и переходим к блоку 4, в противном случае ($x > 5$) – по функции $y = \sqrt{18 - x}$ и переходим к блоку 5. Блоки 4 и 5 проверяют, удовлетворяет ли заданное значение x области определения. Если удовлетворяет, то вычисляется и выводится на экран значение функции (блоки 9-10), иначе выдается сообщение «функция не имеет значений (блок 7)». В блоке 11 программа завершает свою работу.

3. Форма и формат печати исходных данных и результатов выполнения программы

Левый край бумаги

Контрольная работа по информатике

Задача 2

студент группы 312514

Сергеев С.С. *шифр 000*

Результат :

При x = 11.00 значение функции = 11.200 или
При x = 11.00 функция не имеет значений

4. Программа на алгоритмическом языке Фортран

```
      REAL  x, y
      PRINT *, 'Введите значение x '
       READ ( * , * )  x
      WRITE ( * , 5 ) 'Контрольная работа по информатике'
5         FORMAT ( 5x , a33 )
      WRITE ( * , 6 ) 'Задача 2'
6         FORMAT ( 10x , a11)
      WRITE ( * , 7 ) 'студент группы 312514'
7         FORMAT ( 10x , a21)
```

```
8       FORMAT ( 5x , a12 , 6x , a8 )
    WRITE ( * , 6 ) 'Результат :'
    IF  ( x .GT. 5.) THEN
     IF ( x .LE. 18.) THEN
          y = SQRT ( 18 – x )
          WRITE ( * , 9 )  x, y
     ELSE
          WRITE ( * , 10 )  x
     END  IF
    ELSE
     IF ( x .NE. 0.) THEN
          Y = 1 / x
          WRITE ( * , 9 )  x, y
     ELSE
          WRITE ( * , 10 )  x
     END  IF
    END  IF
9       FORMAT ( 1x, 'При x =' , F5.2, ' значение функции =', F6.3 )
10      FORMAT ( 1x, 'При x =' , F5.2, ' функция не имеет значений' )
        END
```

5. Тестовый расчет

Выполним тестовые расчеты для нескольких значений x и занесем результаты расчета в табл. 3.2.

Таблица 3.2

Значение x	Результаты ручного счета y	Результаты расчета на компьютере y
-10	-0,1	Выполняется на лабораторных работах
5	0,2	
0	*Функция не имеет значений*	
14	2,0	
20	*Функция не имеет значений*	

Задача 3

Составить программу вычисления произведения всех положительных элементов 3-й строки массива A (m, n).

Работа над задачей выполнить следующие этапы:

1. Описать математическую модель задачи.
2. Начертить блок-схему алгоритма задачи.
3. Выбрать форму и формат печати исходных данных и результатов выполнения программы на экран. Предусмотреть при этом вывод фамилии, имени, отчества студента, номера группы и шифра задания.
4. Написать текст программы на алгоритмическом языке Фортран.
5. Выполнить тестовый расчет для массива A (4, 5).

Решение

1. Математическая модель и графическая интерпретация задачи

В соответствии с условием задан массив A, состоящий из m строк и n столбцов. Общий элемент массива обозначим через a_{ij} (т.е. элемент, стоящий на пересечении i-й строки и j-го столбца), тогда массив A предстанет в следующем виде:

$$A_{m\times n}=\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}.$$

Произведение элементов 3-й строки можно записать как

$$a_{31}\cdot a_{32}\cdot a_{33}\cdot\cdots\cdot a_{3n}=\prod_{j=1}^{n} a_{3j}.$$

С учетом того, что по условию необходимо вычислить произведение положительных элементов, перед перемножением следует определить, является ли элемент положительным. Таким образом, математическая модель задачи запишется так:

в двумерном массиве $A_{m\times n}$ вычислить $\prod_{j=1}^{n} a_{3j}$ для всех $a_{3j} > 0$.

2. Блок-схема алгоритма задачи

Предусмотрим в программе:

– ввод элементов массива A и вывод их для контроля на печать;

– вычисление произведения положительных элементов третьей строки;

– вывод массива A и результата перемножения на печать.

При составлении блок-схемы (рис. 3.4) ввод и вывод значений массива A предусмотрен без детализации.

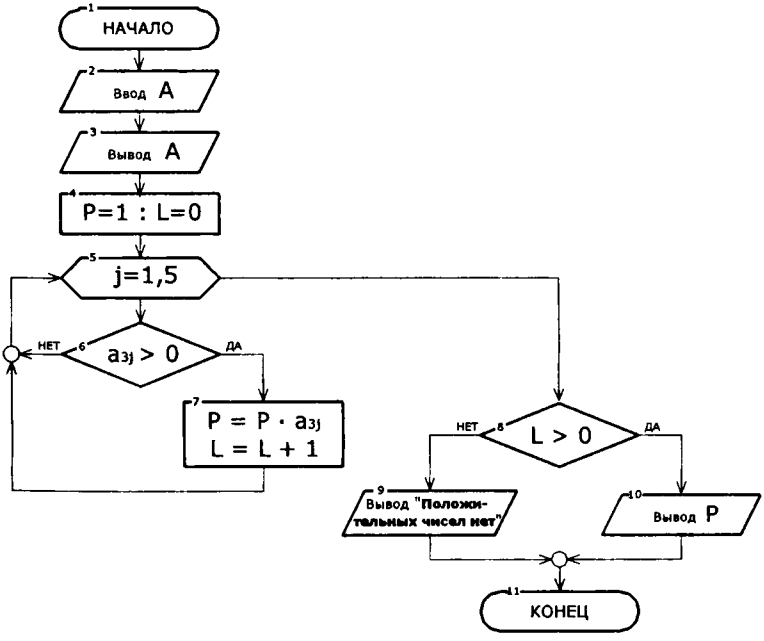


Рис. 3.4. Блок-схема алгоритма задачи 3

Приведем краткое описание блок-схемы. В блоке 2 головной программы производится ввод исходных данных (значений элементов массива *A*). В блоке 3 осуществляется печать элементов массива *A* на печать. В блоке 4 счетчику произведения *P* присваивается значение 1, а счетчик положительных чисел *L* обнуляется. Затем действие передается циклу, образованному блоками 5...7, где его счетчик *j* будет изменяться от 1 до 5 (количество столбцов массива). В блоке 6 производится сравнение элемента третьей строки массива с нулем. Если элемент больше нуля, то в блоке 7 счетчик произведения *P* умножается на этот элемент, величина *L* увеличивается на 1 и затем происходит возврат к блоку 5 (если элемент меньше или равен нулю, то программа минует блок 6 и переходит к блоку 5), где номер столбца увеличивается на 1. После выполнения внутреннего цикла в блоке 8 определяется, есть ли положительные элементы в строке, и если есть ($L > 0$), то печатается результат произведения положительных элементов 3-й строки и программа завершает свою работу. Если положительных элементов нет ($L = 0$), то программа выдает соответствующее сообщение и завершает свою работу.

3. Форма и формат печати исходных данных и результатов выполнения программы

Левый край бумаги

```
    Контрольная работа по информатике
          Задача 3
    студент группы 312514
    Сергеев С.С.    шифр 000
    Исходные данные:  МАССИВ  A
  -5.00   4.50   -6.00   6.50   -4.00
и т.д.
    Результат :
  Произведение равно 1111.11
  Количество равно 11
или
Положительных чисел нет
```

```
      DIMENSION  A (4,5)
      REAL  P
!  ввод элементов массива
      DO  i = 1 , 4
       DO  j = 1 , 5
        PRINT 1 , i , j
        READ ( * , * )  A ( i , j )
       END  DO
      END  DO
1         FORMAT ( 2x, 'Введите значение a (' , I2 , ' ; ' , I2 , ')=' \ )
      PRINT 5 , 'Контрольная работа по информатике'
5         FORMAT ( 5x , a33 )
      PRINT 6 , 'Задача 3'
6         FORMAT ( 10x , a11)
      PRINT 7 , 'студент группы 312514'
7         FORMAT ( 10x , a16)
      PRINT 8 , 'Сергеев С.С.' , 'шифр 000'
8         FORMAT ( 5x , a12 , 6x , a8 )
      PRINT * , ' Исходные данные:  МАССИВ   A'
!  вывод элементов массива
      PRINT 2 , ( ( A(I,J), J=1,5), I=1,4)
2       FORMAT ( 5 (2x, F5.2) )
      PRINT 6 , 'Результат :'
       P = 1.
       L = 0
       DO  j = 1 , 5
        IF ( A ( 3 , j ) .GT. 0.) THEN
          P = P * A ( 3 , j )
          L = L + 1
        END  IF
       END  DO
        IF ( L .GT. 0) THEN
          PRINT 3 , P , L
        ELSE
          PRINT * , ' Положительных чисел нет '
        END  IF
```

```
3     FORMAT ( 3x, ' Произведение равно ' , F7.2 , / 3x, 'Количество равно', I2 )
      END
```

5. Тестовый расчет

Примем для тестового расчета следующий массив:

$$A_{4\times5}=\begin{bmatrix} -1{,}2 & 1{,}4 & 5{,}6 & -7{,}8 & 9{,}1 \\ 2{,}3 & 1{,}6 & 4{,}1 & -1{,}7 & 4{,}3 \\ -1{,}0 & 2{,}0 & 3{,}2 & -1{,}3 & 1{,}5 \\ 0{,}3 & 2{,}1 & 0{,}8 & 3{,}4 & -1{,}3 \end{bmatrix}.$$

Произведение положительных элементов третьей строки равно $2{,}0\cdot3{,}2\cdot1{,}5=9{,}6$.

Задача 4

На отрезке $[a,b]$ задана функция $f(x)=\ln(3x+4)+\sin(3-x)$. Требуется разработать программу вычисления

$$S=h\cdot\sum_{i=0}^{n-1}\left|f(x_i)+\sqrt{f(x_i^2)}\right|$$

при разбиении отрезка на n равных частей.

Исходные данные: $n=36$, $[4,10]$.

При работе над задачей необходимо выполнить следующие этапы:

1. Описать математическую модель задачи и привести ее графическую интерпретацию.
2. Начертить блок-схему алгоритма задачи.
3. Выбрать форму и формат печати исходных данных и результатов выполнения программы на экран (предусмотреть вывод фамилии, инициалов студента, номера группы и шифра задания).
4. Написать текст программы на алгоритмическом языке Фортран (предусмотреть ввод данных как из внешнего файла, так и с клавиатуры; вывод результата по формату как на экран, так и во внешний файл).
5. Выполнить тестовый расчет по исходным данным (при $n=6$).

Решение

1. Математическая модель и графическая интерпретация задачи

Изобразим графическую интерпретацию задачи (рис. 3.5).

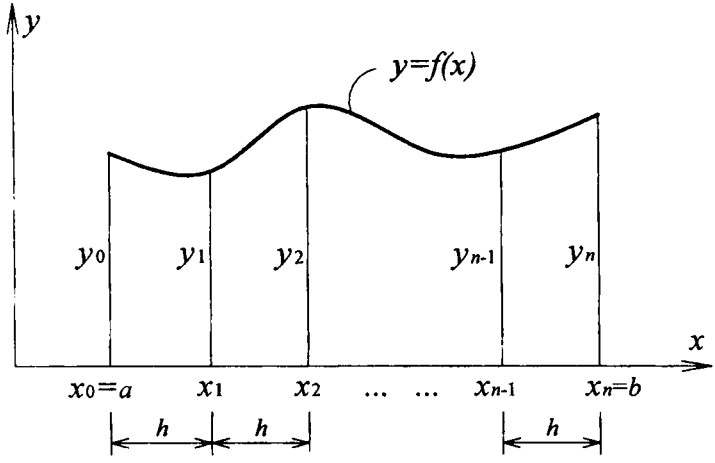


Рис. 3.5. Графическая интерпретация задачи 4

Шаг разбиения отрезка вычисляется по формуле

$$h = \frac{b-a}{n}.$$

Разбив отрезок $[a,b]$ на равные отрезки длиной h, находим абсциссы точек, в которых мы должны найти ординаты функции. В общем виде их можно записать

$$x_i = a + i \cdot h \ , \quad \text{где } i = \overline{0,n}.$$

Учитывая вышеизложенное, запишем *математическую модель* задачи в следующем виде:

для заданного отрезка $[a,b]$, количества частей его разбиения n

вычислить $S = h \cdot \sum_{i=0}^{n-1} \left| f(x_i) + \sqrt{f(x_i^2)} \right|$,

где $f(x_i) = \ln(3x_i + 4) + \sin(3 - x_i)$, $x_i = a + i \cdot h$, $h = \frac{b-a}{n}$,

$i = \overline{0, n-1}$.

2. Блок-схема алгоритма задачи

Предусмотрим в программе:

– ввод исходных данных (a, b, n);

– вычисление длины каждого участка h;

– вывод исходных данных для контроля на печать;

– вычисление значений S_i для каждого значения x_i и их суммирование;

– вывод результатов расчета на экран.

Блок-схема алгоритма представлена на рис. 3.6.

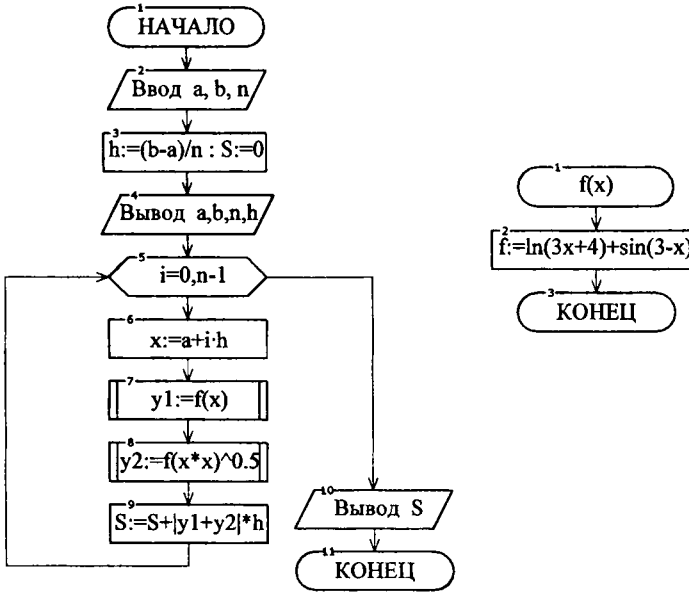


Рис. 3.6. Блок-схема алгоритма задачи 4

3. Форма и формат печати исходных данных и результатов выполнения программы

Левый край бумаги

____Контрольная работа по информатике
_____Задача 4
_____студент группы 312514
____Сергеев С.С. _____шифр 000
____Исходные данные:
отрезок ____[04.0, 09.0]
количество участков n=36
длина участка h=1.00
Результат вычислений ___111.111

4. Программа на алгоритмическом языке FORTRAN

```
PROGRAM Integral
REAL a , b , h
F ( x ) = LOG (3*x + 4) + SIN (3 - x)
WRITE (*,*) 'ввод данных из файла ? [да-1, нет-др. цифра]'
READ ( * , * ) k
IF (k.EQ.1) THEN
   OPEN ( 1 , file = 'data1.txt' )
     READ ( 1 , * ) a
     READ ( 1 , * ) b
     READ ( 1 , * ) n
   CLOSE ( 1 )
ELSE
   WRITE ( * , * ) 'введите отрезок [a,b]'
   READ ( * , * ) a , b
   WRITE ( * , * ) 'введите количество участков'
   READ ( * , * ) n
END IF
h = ( b – a ) / n
DO i = 0 , n – 1
```

```
      x = a + i * h
      s = s + h * ABS ( F(x) + SQRT ( F(X**2) ) )
    END DO
    OPEN ( 2 , file = 'rezult1' )
    WRITE ( 2 , 5 ) 'Контрольная работа по информатике'
    WRITE ( * , 5 ) 'Контрольная работа по информатике'
5     FORMAT ( 5x , a33 )
    WRITE ( 2 , 6 ) 'Задача 4'
    WRITE ( * , 6 ) 'Задача 4'
6     FORMAT ( 10x , a11)
    WRITE ( 2 , 7 ) 'студент группы 312514'
    WRITE ( * , 7 ) 'студент группы 312514'
7     FORMAT ( 10x , a16)
    WRITE ( 2 , 8 ) 'Сергеев С.С.' , 'шифр 000'
    WRITE ( * , 8 ) 'Сергеев С.С.' , 'шифр 000'
8     FORMAT ( 5x , a12 , 6x , a8 )
    WRITE ( 2 , 9 ) 'Исходные данные:'
    WRITE ( * , 9 ) 'Исходные данные:'
9     FORMAT ( 5x , a16 )
    WRITE ( 2 , 10 ) 'отрезок', a , b
    WRITE ( * , 10 ) 'отрезок', a , b
10    FORMAT ( 1x , a7 , 4x , '[' , f4.1 , ';' , f4.1 , ']')
    WRITE ( 2 , 11 ) 'количество участков', n
    WRITE ( * , 11 ) 'количество участков', n
11    FORMAT ( 1x , a19 , 1x , 'n = ' , i2)
    WRITE ( 2 , 12 ) 'длина участка', h
    WRITE ( * , 12 ) 'длина участка', h
12    FORMAT ( 1x , a13 , 3x , 'h = ' , f4.2 )
    WRITE ( 2 , 13 ) 'Результат вычислений', s
    WRITE ( * , 13 ) 'Результат вычислений', s
13    FORMAT ( 1x , a20 , 3x , f7.3)
    CLOSE ( 2 )
    END
```

5. *Тестовый расчет*

Выполним ручной счет при n = 6. Отрезок интегрирования $[4;10]$. Длина каждого участка интегрирования равна $h = \frac{10-4}{6} = 1{,}0$. Вычислим значение функции $f(x)$ в каждой точке отрезка. Расчет сведем в (табл. 3.3).

Таблица 3.3

x_i	$f(x_i)$	$\sqrt{f(x_i^2)}$
4,0	1,931	1,879
5,0	2,035	2,092
6,0	2,950	1,928
7,0	3,976	2,029
8,0	4,291	2,499
9,0	3,713	2,235
$\sum$	18,896	12,662

$$S = h \cdot \sum_{i=0}^{n-1} \left| f(x_i) + \sqrt{f(x_i^2)} \right| = 1 \cdot (18{,}896 + 12{,}662) = 31{,}558.$$

Таблица 3.4

Результат ручного счета при n = 6	Результаты расчета на компьютере при n = 6	Результаты расчета на компьютере при n = 36	Погрешность расчета при различных значениях n, %
31,558	*Выполняется на лабораторных занятиях*		

Задача 5

Отделить корень уравнения $g(x)=3-x^2-e^x=0$ и разработать программу его уточнения методом перебора с точностью 0,001.

При работе над задачей необходимо выполнить следующие этапы:

1. Описать математическую модель задачи и привести ее графическую интерпретацию.
2. Начертить блок-схему алгоритма задачи.
3. Выбрать форму и формат печати исходных данных и результатов вывода программы на экран (предусмотреть вывод фамилии, инициалов студента, номера группы и шифра задания).
4. Написать текст программы на алгоритмическом языке Фортран (предусмотреть: ввод данных как из внешнего файла, так и с клавиатуры; вывод результата по формату как на экран, так и во внешний файл).
5. Выполнить тестовый расчет по исходным данным (для первых четырех шагов).

Решение

1. ***Математическая модель и графическая интерпретация***

Вычисление корней алгебраических и трансцендентных нелинейных уравнений вида $g(x)=0$ с заданной наперед точностью обычно осуществляют путем применения какой-либо итерационной процедуры, состоящей в построении числовых последовательностей x_n, сходящихся к искомому корню $\tilde{x}$ уравнения. Перед применением процедуры отделяют отрезок $[a,b]$, на котором находится корень уравнения.

При решении нелинейного уравнения методом перебора задаются начальное значение аргумента $x=x_0=a$ и шаг $h=\varepsilon$, который при этом определяет и точность нахождения корня. Пока выполняется условие $g(x)\cdot g(x+h)>0$, аргумент x увеличиваем на шаг h ($x = x + h$) (рис. 3.7). Если произведение $g(x)\cdot g(x+h)$ становится отрицательным, то на отрезке [x, x+h] существует решение уравнения. За корень уравнения принимаем при этом $\tilde{x}=x+\frac{h}{2}$.

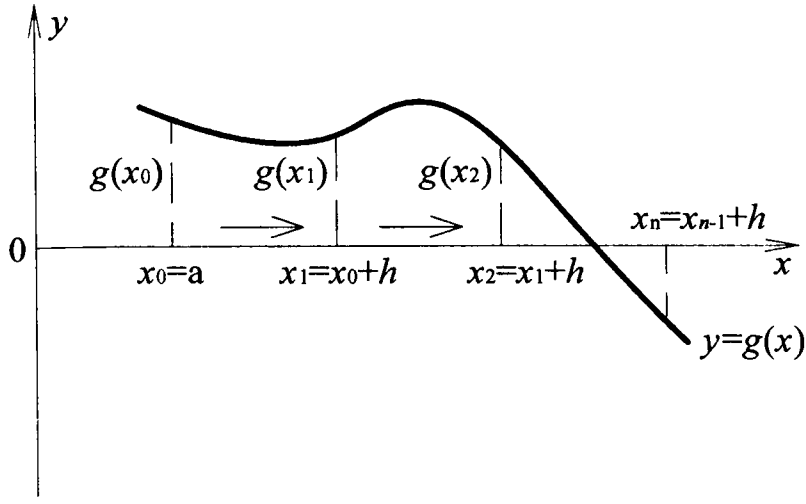


Рис. 3.7. Графическая интерпретация задачи 5

2. Блок-схема алгоритма задачи

Предусмотрим в программе:

– ввод отрезка $[a,b]$, на котором находится корень уравнения, и точности ε;

– выбор начального значения аргумента x_0;

– уточнение корня методом перебора;

– вывод результата на экран и в файл.

Блок-схема алгоритма представлена на рис. 3.8.

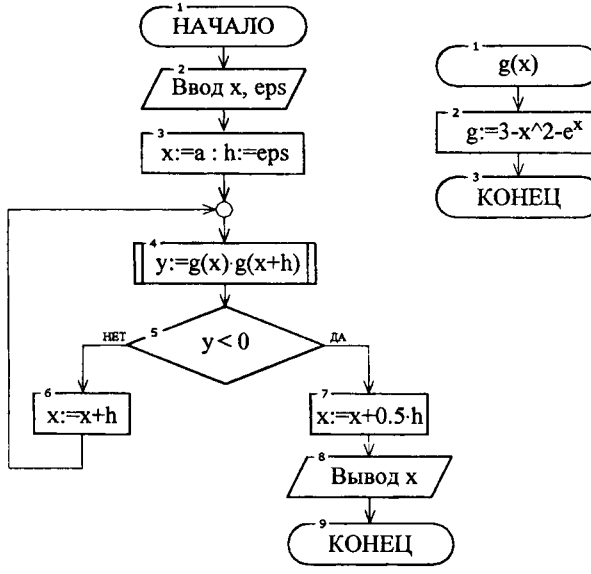


Рис. 3.8. Блок-схема алгоритма задачи 5

3. Форма и формат печати исходных данных и результатов выполнения программы

Левый край бумаги

____Контрольная работа по информатике
_____ ___Задача 5
_____студент группы 312514
____Сергеев С.С. ___ ___шифр 000
____Исходные данные:
отрезок ____[4.0, 9.0]
точность e=0.001
на 111 шаге значение x= 11.1111
_____Результат :
значение корня равно 11.1111

4. Программа на алгоритмическом языке Фортран

```
      PROGRAM Uravnenie
      REAL a , b , eps , h, x , y , z
      INTEGER  i , k
      G ( z ) = 3 - z**2 – EXP ( z )
      WRITE (*,*) 'ввод данных из файла ? [да-1, нет-др. цифра]'
      READ ( * , * ) k
      IF (k.EQ.1) THEN
        OPEN ( 3 , file = 'data2.txt' )
          READ ( 3 , * ) a
          READ ( 3 , * ) b
          READ ( 3 , * ) eps
        CLOSE ( 3 )
      ELSE
        WRITE ( * , * ) 'введите отрезок [a,b]'
        READ ( * , * ) a , b
        WRITE ( * , * ) 'введите точность вычислений e'
        READ ( * , * ) eps
      END IF
      x = a
      h = eps
      OPEN ( 4 , file = 'rezult2' )
      WRITE ( 4 , 5 ) 'Контрольная работа по информатике'
      WRITE ( * , 5 ) 'Контрольная работа по информатике'
5     FORMAT ( 5x , a33 )
      WRITE ( 4 , 6 ) 'Задача 5'
      WRITE ( * , 6 ) 'Задача 5'
6     FORMAT ( 10x , a11)
      WRITE ( 4 , 7 ) 'студент группы 312514'
      WRITE ( * , 7 ) 'студент группы 312514'
7       FORMAT ( 10x , a21)
      WRITE ( 4 , 8 ) 'Сергеев С.С.' , 'шифр 000'
      WRITE ( * , 8 ) 'Сергеев С.С.' , 'шифр 000'
8     FORMAT ( 5x , a12 , 6x , a8 )
      WRITE ( 4 , 9 ) 'Исходные данные:'
      WRITE ( * , 9 ) 'Исходные данные:'
9     FORMAT ( 5x , a16 )
```

```
    WRITE ( 4 , 10 ) 'отрезок', a , b
    WRITE ( * , 10 ) 'отрезок', a , b
10  FORMAT ( 1x , a7 , 4x , '[' , f4.1 , ';' , f4.1 , ']')
    WRITE ( 4 , 11 ) 'точность', eps
    WRITE ( * , 11 ) 'точность', eps
11  FORMAT ( 1x , a8 , 1x , 'e = ' , f5.3)
14  y = G (x) * G (x+h)
    IF (y.GT.0) THEN
      x = x + h
      i = i +1
      WRITE ( 4 , 12 )  i , x
      WRITE ( * , 12 )  i , x
12    FORMAT ( 1x , ' на ', i3, ' шаге значение x= ' , f6.3)
      GOTO 14
    ELSE
      x = x + 0.5*h
    END IF
    WRITE ( 4 , 6 ) 'Результат:'
    WRITE ( * , 6 ) 'Результат:'
    WRITE ( 4 , 13 ) 'Значение корня равно', x
    WRITE ( * , 13 ) 'Значение корня равно', x
13  FORMAT ( 1x , a24 , 3x , f6.3)
    CLOSE ( 4 )
    END
```

5. Тестовый расчет

Отделим корень графически. Для этого в одной системе координат построим графики функций $y = 3 - x^2$ и $y = e^x$ (рис. 3.9).

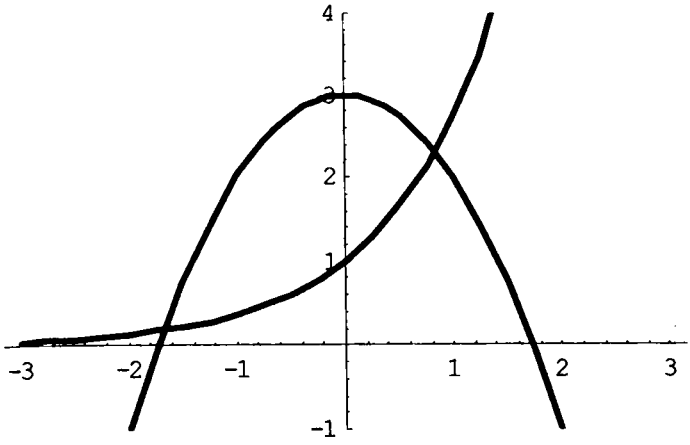


Рис. 3.9. Графическое отделение корня уравнения

Примем отрезок $[0{,}7;\ 1]$ за отрезок, на котором находится корень уравнения. Уточним его методом перебора, выполнив 4 шага расчета. Сведем вычисления в таблицу (табл. 3.5).

Таблица 3.5

Шаг	Значение x_{i-1}	Значение $x_i = x_{i-1} + h$	$g(x_{i-1}) \cdot g(x_i)$	Результаты расчета на компьютере x_i
1	0,7	0,701	0,245	*Выполняется на лабораторных занятиях*
2	0,701	0,702	0,241	
3	0,702	0,703	0,238	
4	0,703	0.704	0,235	

В результате расчета на компьютере (*выполняется на лабораторных занятиях*) корень уравнения $x = 0{,}834$.

Задача 6

Составить программу вычисления значения β по выражению

$$\beta = \sin\frac{\sum_{i=1}^{8} A_i^2}{A_{\max} - A_{\min}} + \cos^2\frac{\sum_{i=1}^{12} B_i^2}{B_{\max} + B_{\min}} - \frac{1}{8}\sqrt{\frac{\sum_{i=1}^{10} C_i^2}{C_{\max}}}.$$

При составлении программы требуется:

✓ разработать подпрограмму *VVOD* ввода элементов одномерного массива с клавиатуры;

✓ разработать подпрограмму *VYVOD* вывода одномерного массива по формату во внешний файл;

✓ разработать подпрограмму *MAXMIN* нахождения наибольшего и наименьшего элементов массива;

✓ разработать подпрограмму *SUMMA* вычисления суммы квадратов элементов массива;

✓ с помощью разработанных подпрограмм написать головную программу вычисления значения β.

При работе над задачей необходимо выполнить следующие этапы:

1. Описать математическую модель задачи.
2. Описать подпрограммы.
3. Начертить блок-схему алгоритма задачи.
4. Выбрать форму и формат печати исходных данных и результатов выполнения программы на экран (предусмотреть вывод фамилии, инициалов студента, номера группы и шифра задания).
5. Написать текст программы на алгоритмическом языке Фортран (предусмотреть ввод данных с клавиатуры и вывод результата по формату во внешний файл).
6. Выполнить тестовый расчет по принятым исходным данным.

Решение

1. Математическая модель задачи

Для заданных по условию одномерных массивов *A* (состоит из 8 элементов), *B* (состоит из 12 элементов) и *C* (состоит из 10 элементов) требуется вычислить значение

$$\beta = \sin\frac{\sum_{i=1}^{8} A_i^2}{A_{\max} - A_{\min}} + \cos^2\frac{\sum_{i=1}^{12} B_i^2}{B_{\max} + B_{\min}} - \frac{1}{8}\sqrt{\frac{\sum_{i=1}^{10} C_i^2}{C_{\max}}},$$

где $A_{\max}$, $B_{\max}$, $C_{\max}$ – наибольшие элементы массивов A, B и C соответственно;

$A_{\min}$, $B_{\min}$ – наименьшие элементы массивов A и B соответственно.

2. Описание подпрограмм

Подпрограмма *VVOD* (*m* , *D*) ввода элементов одномерного массива с клавиатуры. Исходные данные: *m* – количество элементов массива. Результат выполнения подпрограммы – массив *D*, состоящий из *m* элементов.

Подпрограмма *VYVOD* (*m* , *D*) вывода элементов одномерного массива по формату во внешний файл. Исходные данные: *m* – количество элементов массива; массив *D*, состоящий из *m* элементов.

Подпрограмма *MAXMIN* (*m* , *D* , *maxd* , *mind*) нахождения наибольшего и наименьшего элементов массива. Исходные данные: *m* – количество элементов массива; массив *D*, состоящий из *m* элементов. Результаты выполнения подпрограммы: *maxd* – наибольший элемент массива, *mind* – наименьший элемент.

Подпрограмма *SUMMA* (*m* , *D* , *s*) вычисления суммы квадратов элементов массива. Исходные данные: *m* – количество элементов массива; массив *D*, состоящий из *m* элементов. Результат выполнения подпрограммы – сумма квадратов элементов массива *s*.

3. Блок-схема алгоритма задачи

Предусмотрим в программе:

– ввод элементов массивов *A*, *B* и *C* с помощью подпрограммы *VVOD*;

– вычисление сумм квадратов (*sa*, *sb*, *sc*) элементов массивов с помощью подпрограммы *SUMMA*;

– нахождение наибольших (*maxa*, *maxb*, *maxc*) и наименьших (*mina*, *minb*, *minc*) элементов массивов с помощью подпрограммы *MAXMIN*;

– вычисление значения β;

– вывод массивов *A*, *B* и *C* для контроля с помощью подпрограммы *VYVOD*;

– вывод результата расчета (β) во внешний файл.

Блок-схема головной программы представлена на рис. 3.10.

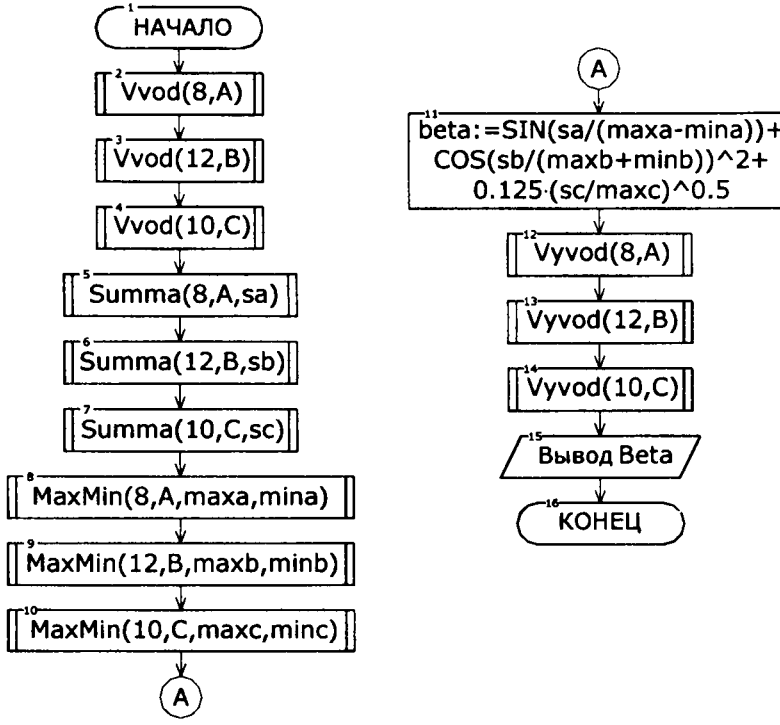


Рис. 3.10. Блок-схема головной программы

Блок-схемы подпрограмм представлены на рис. 3.11.

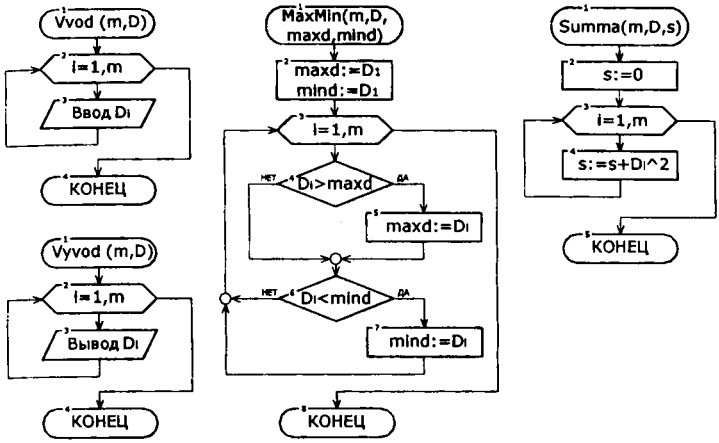


Рис. 3.11. Блок-схемы подпрограмм

4. Форма и формат печати исходных данных и результатов выполнения программы

Левый край бумаги

Контрольная работа по информатике

Задача 6

Студент группы 312514

Сергеев С.С. шифр 000

Исходные данные:

МАССИВ A

11.11 11.11 11.11 11.11 11.11 и т.д.

МАССИВ B

11.11 11.11 11.11 11.11 11.11 и т.д.

МАССИВ C

11.11 11.11 11.11 11.11 11.11 и т.д.

Результат:

Значение beta равно 11.111

5. Программа на алгоритмическом языке Фортран

```
!  Головная программа
   DIMENSION A(8), B(12), C(10)
   REAL  sa, sb, sc, maxa, maxb, maxc, mina, minb, minc, beta
!  Ввод элементов массивов
   WRITE ( * , * ) 'Ввод массива A'
   CALL  VVOD (8, A)
```

Здесь и далее в подпрограмму VVOD посылаем в качестве фактических параметров имя массива – A; количество элементов этого массива – 8; обратно получим заполненный массив.

```
   WRITE ( * , * ) 'Ввод массива B'
   CALL  VVOD (12, B)
   WRITE ( * , * ) 'Ввод массива C'
   CALL  VVOD (10, C)
!  Вычисление сумм квадратов элементов массивов
   CALL  SUMMA (8, A, sa)
```

Здесь и далее в подпрограмму SUMMA посылаем в качестве фактических параметров массив – A; количество элементов этого массива – 8; обратно получим сумму квадратов элементов засланного массива – sa.

```
   CALL  SUMMA (12, B, sb)
   CALL  SUMMA (10, C, sc)
!  Нахождение наибольших и наименьших элементов массивов
   CALL  MAXMIN (8, A, maxa, mina)
```

Здесь и далее в подпрограмму MAXMIN посылаем в качестве фактических параметров массив – A; количество элементов этого массива – 8; обратно получим наибольший элемент массива – maxa*; наименьший элемент массива –* mina*.*

```
   CALL  MAXMIN (12, B, maxb, minb)
   CALL  MAXMIN (10, C, maxc, minc)
!  Вычисление значения BETA
```

```
      beta = SIN ( sa/(maxa–mina) ) + COS ( sb/(maxb+minb) )**2
      beta = beta – (1/8.)*SQRT ( sc/maxc )
!     Вывод исходных данных и результата расчета
       OPEN ( 1 , file = 'rezult6' )
      WRITE ( 1 , 5 ) 'Контрольная работа по информатике'
5      FORMAT ( 5x , a33 )
      WRITE ( 1 , 6 ) 'Задача 3'
6      FORMAT ( 10x , a11)
      WRITE ( 1 , 7 ) 'студент группы 312514'
7      FORMAT ( 10x , a21)
      WRITE ( 1 , 8 ) 'Сергеев С.С.' , 'шифр 000'
8      FORMAT ( 5x , a12 , 6x , a8 )
      WRITE ( 1 , 7 ) ' Исходные данные: '
       WRITE ( 1 , 6 ) ' Массив  A '
      CALL  VYVOD (8, A)
       WRITE ( 1 , 6 ) ' Массив  B '
      CALL  VYVOD (12, B)
       WRITE ( 1 , 6 ) ' Массив  C '
      CALL  VYVOD (10, C)
       WRITE ( 1 , 6 ) ' Результат : '
      WRITE ( 1 , 9 ) beta
9      FORMAT ( 3x , 'Значение beta равно ' , F6.3 )
      CLOSE (1)
      STOP
      END

!     Подпрограмма ввода массива
      SUBROUTINE  VVOD ( m , D )
       DIMENSION  D (*)
       INTEGER  m, i
        DO  i = 1 , m
          READ ( * ,* ) D ( i )
        END DO
       RETURN
      END
```

```
!  Подпрограмма вывода массива
   SUBROUTINE  VYVOD ( m , D )
    DIMENSION  D (*)
    INTEGER  m, i
      WRITE ( 1 , 2 ) ( D(I), I=1,m)
2      FORMAT ( 12 (2x, F5.2) )
    RETURN
   END

   !  Подпрограмма вычисления суммы квадратов элементов
   SUBROUTINE  SUMMA ( m, D, s )
    DIMENSION  D (*)
    INTEGER  m, i
    REAL  s
    S=0
     DO  i = 1 , m
       s = s + D ( i )**2
     END DO
    RETURN
   END

!  Подпрограмма нахождения наибольшего
!  и наименьшего элементов массива
   SUBROUTINE  MAXMIN (m, D, maxd, mind)
    DIMENSION  D (*)
    INTEGER  m, i
    REAL  maxd, mind
    maxd = D (1)
    mind = D (1)
    DO  i = 2, m
     IF ( D (i) .GT. maxd) maxd = D (i)
     mind = AMIN1( D (i), mind)
    END  DO
    RETURN
   END
```

6. Тестовый расчет

Примем в качестве исходных данных массивы:

$$A = (0{,}8 \quad 0{,}6 \quad 0{,}4 \quad 0{,}6 \quad 0{,}5 \quad 0{,}7 \quad 0{,}8 \quad 0{,}6);$$

$$B = (1{,}1 \quad 1{,}2 \quad 1{,}0 \quad 1{,}4 \quad 1{,}3 \quad 1{,}2 \quad 1{,}1 \quad 1{,}3 \quad 1{,}1 \quad 1{,}5 \quad 1{,}4 \quad 1{,}3);$$

$$C = (3{,}6 \quad 3{,}2 \quad 3{,}4 \quad 3{,}3 \quad 3{,}6 \quad 3{,}5 \quad 3{,}7 \quad 3{,}8 \quad 3{,}5 \quad 3{,}4).$$

Определим для каждого массива суммы квадратов их элементов, наибольшие и наименьшие элементы. Для удобства результаты представим в (табл. 3.6).

Таблица 3.6

Массив	Сумма квадратов элементов массива	Наибольший элемент массива	Наименьший элемент массива
A	3,26	0,8	0,4
B	18,75	1,5	1,0
C	122,80	3,8	3,2

Вычисляем значение β:

$$\beta = \sin\frac{3{,}26}{0{,}8-0{,}4} + \cos^2\frac{18{,}75}{1+1{,}5} - \frac{1}{8}\sqrt{\frac{122{,}80}{3{,}8}} =$$

$$= 0{,}957 + 0{,}120 - 0{,}711 = 0{,}366 \ .$$

В результате расчета на компьютере (*выполняется на лабораторных занятиях*), значение $\beta = 0{,}366$ (погрешность – 0 %).

ЛИТЕРАТУРА

1. ГОСТ 19.701-90. ЕСПД. Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения. – Введ. 1992–01–01. – М.: Изд-во стандартов, 1991. – 26 с.

2. Белецки, Я. Фортран-77 / Пер. с пол. – М.: Высшая школа, 1991. – 207 с.

3. Боглаев, Ю.П. Вычислительная техника и программирование. – М.: Высшая школа, 1990.– 543 с.

4. Фурунжиев, Р.И. Вычислительная техника: практикум. – Мн.: Вышэйшая школа, 1985. – 254 с.

5. Программирование на Фортране 77 / Дж. Ашкрофт. [и др.]. – М.: Радио и связь, 1990. – 272 с.

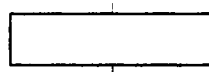
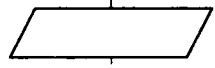
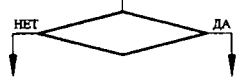
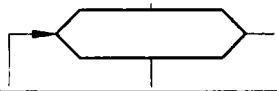
6. Бартеньев, О.В. Фортран для студентов. – М.: Диалог-МИФИ, 1999. –397 с.

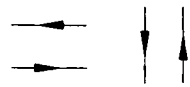
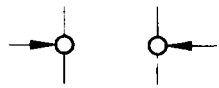
7. Рыжиков, Ю.И. Программирование на Fortran Power Station для инженеров: практическое рук. – СПб.: Корона принт, 1999. – 256 с.

8. Соловьев, П.В. FORTRAN для персонального компьютера: справ. пособие. – М.: Arist, 1991. – 223 с.

Блок-схема алгоритма и ее элементы

Блок-схема – это последовательность блоков, предписывающих выполнение определенных операций и связей между этими блоками. Внутри блоков указывается информация об операциях, подлежащих выполнению. Конфигурация и размеры блоков, а также порядок графического оформления блок-схем регламентированы [1]. В таблице приведены наиболее часто используемые блоки, изображены элементы связей между ними и дано краткое пояснение к ним. Блоки и элементы связей называют *элементами* блок-схем. Представленных в таблице элементов вполне достаточно для изображения алгоритмов, которые необходимы при выполнении студенческих работ.

Название	Элемент	Комментарий
Начало, Конец		Начало, конец, пуск, остановка, вход и выход во вспомогательных алгоритмах
Процесс		Вычислительное действие или последовательность вычислительных действий
Данные		Ввод/Вывод данных
Решение		Проверка условия
Модификация		Заголовок цикла
Предопределенный процесс		Определение вычислений по подпрограмме или стандартной программе. Обычно обозначается вызов подпрограммы

Название	Элемент	Комментарий
Комментарий		Пояснения к схеме алгоритма, формулы
Соединитель		Разрыв линий потока, связывающих блоки алгоритма
Горизонтальные и вертикальные потоки		Линии связей между блоками, направление потоков
Слияние		Слияние линий потоков
Межстраничный соединитель		Используется при переносе части схемы на другую страницу. Если блок стоит в конце обрываемой схемы, то *m* – страница, на которой продолжится прерванная схема, *n* – номер следующего блока. Если соединитель стоит в начале прерванной схемы, то *m* – номер страницы, на которой схема прервана, а *n* – номер предыдущего блока

При соединении блоков следует использовать только *вертикальные* и *горизонтальные* линии потоков. Горизонтальные потоки, имеющие направление справа налево, и вертикальные потоки, имеющие направление снизу вверх, должны быть обязательно помечены стрелками. Прочие потоки могут быть помечены или оставлены непомеченными. Линии потоков должны быть параллельны линиям внешней рамки или границам листа.

ПРИЛОЖЕНИЕ 2

Некоторые требования к изображению элементов блок-схем

Расстояние между параллельными линиями потоков должно быть не менее 3 мм, между остальными элементами схемы – не менее 5 мм.

Горизонтальный и вертикальный размеры блока должны быть кратны 5 мм (делиться на 5 нацело).

Отношение горизонтального и вертикального размеров блока, равное 1,5, является основным. При ручном выполнении блока допустимо отношение, равное 2.

Блоки «Начало», «Конец» и «Соединитель», как правило, имеют высоту вдвое меньшую основной высоты блоков.

Для размещения блоков поле листа рекомендуется разбивать на горизонтальные и вертикальные (для разветвляющихся схем) зоны.

Для удобства описания блок-схемы каждый ее блок следует пронумеровать. Удобно использовать сквозную нумерацию блоков. Номер блока располагают в левой части блока либо в разрыве в левой верхней части рамки блока.

Список некоторых библиотечных функций Фортрана

Математическая запись	Описание	Запись на языке Фортран	Тип параметров	Тип функции
1	2	3	4	5
Тригонометрические функции				
arccos x	арккосинус	ACOS (x)	real	real
		DACOS (x)	real*8	real*8
arcsin x	арксинус	ASIN (x)	real	real
		DASIN (x)	real*8	real*8
arctg x	арктангенс	ATAN (x)	real	real
		DATAN (x)	real*8	real*8
ch x	гиперболический косинус	COSH (x)	real	real
		DCOSH (x)	real*8	real*8
sh x	гиперболический синус	DSINH (x)	real*8	real*8
		SINH (x)	real	real
th x	гиперболический тангенс	DTANH (x)	real*8	real*8
		TANH (x)	real	real
cos x	косинус	COS (x)	real	real
		DCOS (x)	real*8	real*8
ctg x	котангенс	COTAN (x)	real	real
		DCOTAN (x)	real*8	real*8
sin x	синус	DSIN (x)	real*8	real*8
		SIN (x)	real	real
tg x	тангенс	DTAN (x)	real*8	real*8
		TAN (x)	real	real
		DABS (x)	real*8	real*8
		IABS (x)	integer	integer
		ABS (x)	real	real

1	2	3	4	5
Алгебраические функции				
$\sqrt{x}$	квадратный корень	DSQRT (x)	real*8	real*8
		SQRT (x)	real	real
$\ln x$	логарифм натуральный	ALOG (x)	real	real
		DLOG (x)	real*8	real*8
		LOG (x)	real	real
$\lg x$	логарифм десятичный	DLOG10 (x)	real*8	real*8
		LOG10 (x)	real	real
		ALOG10 (x)	real	real
$\max\{a,b,...\}$	максимум	DMAX1 (A, B, ...)	real*8	real*8
		MAX(A, B, ...)	integer	integer
		AMAX0 (A, B, ...)	integer	real
		AMAX1 (A, B, ...)	real	real
$\min\{a,b,...\}$	минимум	AMIN0 (A, B, ...)	integer	real
		AMIN1 (A, B, ...)	real	real
		DMIN1 (A, B, ...)	real*8	real*8
		MIN(A, B, ...)	integer	integer
	остаток от деления a на b	MOD (A, B)	integer	integer
		DMOD (A, B)	real*8	real*8
		AMOD (A, B)	real	real
$[x]$	целая часть числа	INT (x)	real, integer	integer
e^x	экспонента	DEXP (x)	real*8	real*8
		EXP (x)	real	real

Оглавление

Учебное издание

ТРЕПАЧКО Виктор Михайлович

КРАТКИЙ КУРС ПРОГРАММИРОВАНИЯ НА АЛГОРИТМИЧЕСКОМ ЯЗЫКЕ FORTRAN POWER STATION

Методическое пособие
по дисциплине «Информатика»
для студентов специальности
1-70 02 01 «Промышленное и гражданское строительство»

Редактор Т.Н. Микулик
Компьютерная верстка А.Г. Гармазы

Подписано в печать 24.10.2006.
Формат 60x84 1/16. Бумага офсетная.
Отпечатано на ризографе. Гарнитура Таймс.
Усл. печ. л. 6,4. Уч.-изд. л. 5,0.Тираж 300. Заказ 681.

Издатель и полиграфическое исполнение:
Белорусский национальный технический университет.
ЛИ № 02330/0131627 от 01.04.2004.
220013, Минск, проспект Независимости, 65.