БИБЛИОТЕЧКА ПРОГРАММИСТА

Г. К. БОРОВИН
М. М. КОМАРОВ
В. С. ЯРОШЕВСКИЙ

Ошибки-ловушки при программировании на фортране

БИБЛИОТЕЧКА
ПРОГРАММИСТА

Г. К. БОРОВИН, М. М. КОМАРОВ,
В. С. ЯРОШЕВСКИЙ

ОШИБКИ-ЛОВУШКИ ПРИ ПРОГРАММИРОВАНИИ НА ФОРТРАНЕ

Под редакцией
Ю. М. БАЯКОВСКОГО

МОСКВА «НАУКА»
ГЛАВНАЯ РЕДАКЦИЯ
ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ
1987

ПРЕДИСЛОВИЕ ТИТУЛЬНОГО РЕДАКТОРА

У книги, с которой вы начинаете знакомиться, не совсем обычное название. Необычен и ее жанр. Это — не монография, не учебник, не задачник. Можно, наверное, назвать такую книгу хрестоматией, в которой собраны примеры, позволяющие совершенствовать искусство программирования на фортране.

Конечно, если приходится от случая к случаю писать небольшие программы, можно обойтись весьма поверхностными представлениями о языке. Зачастую и программировать-то нет нужды — в библиотеке отыщется готовая, ранее уже кем-то написанная программа.

Однако при решении больших и сложных задач (а именно для решения таких задач создаются и совершенствуются вычислительные машины) прикладной программист вынужден провести иногда довольно глубокое экспериментальное исследование транслятора. Инструментом нужно владеть в совершенстве и пользоваться им разумно и эффективно. К сожалению, в документации, которой располагает программист, содержится, как правило, лишь самая общая информация о языке и не сбсуждаются особенности реализации тех или иных его конструкций, а реализованы они могут быть по-разному. Как-то проводилось исследование шести различных трансляторов языка фортран, и оно показало, что только одна конструкция (а именно — CONTINUE) во всех трансляторах интерпретируется одинаково.

При составлении больших программ необходимо учитывать множество порой противоречивых факторов. Важны и микросекунды, которые затрачиваются на выполнение оператора, и килобайты, которые займет программа в памяти. К тому же полезно обеспечить мобильность программы — возможность решения задачи на разных машинах.

Использование тонких, нетривиальных особенностей языка (а тем более транслятора) неизмеримо повышает риск появления ошибок. Считают, что количество ошибок растет экспоненциально с увеличением длины программы. В «программистском фольклоре» бытует утверждение, согласно которому «в любой работающей программе есть хотя бы одна ошибка». Стадия отладки и тестирования программ чрезвычайно трудоемка. По некоторым оценкам на нее приходится 40—50 % времени, затрачиваемого на производство программы. Принято даже считать, что «хороший программист не тот, который не допускает ошибок, а тот, который их быстро находит». Новичка ошибки обескураживают, приводя его в полнейшее уныние. Стоит ли удивляться: в институтах технике отладки не учат, а формальных правил отладки вообще не существует. В этих условиях настоящая книга может служить существенным подспорьем. Основываясь на своем богатом опыте и опыте своих коллег, авторы собрали, можно сказать, хрестоматийные примеры ошибок, имеющих самую разную природу.

Легкость отладки в значительной мере зависит и от стиля программирования. Если программисту удалось выразить свою мысль на языке программирования просто и ясно и если программа хорошо структурирована, то ее отладка пройдет споро и безболезненно. Путаная и вычурная программа обязательно отнимет при отладке много времени и сил. Говоря о стиле программирования, стоит привести классический пример. Когда вы прочтете приведенные ниже три строки на фортране, остановитесь на мгновение и подумайте: каков будет результат выполнения этого фрагмента? Заметьте (по часам), как долго продолжалось это «мгновение»? Итак, три строки:

```
      DO 1 I=1,N
      DO 1 J=1,N
1 X(I,J)=(I/J)*(J/I)
```

Надо полагать, вы поняли смысл этих трех строк, и поэтому, не прибегая к словесным пояснениям, приведем эквивалентный фрагмент, который, несомненно, понятнее и к тому же, реализованный транслятором, будет существенно эффективнее:

```
      DO 20 I=1,N
         DO 10 J=1,N
10          X(I,J)=0.0
20       X(I,I)=1.0
```

Стилю программирования в последние годы уделяется много внимания. В книге имеется небольшая глава, посвященная этой теме. Дополнительные сведения можно почерпнуть из других источников, упомянутых в списке литературы. В книге есть приложения, которые полезны читателям, работающим на ЭВМ БЭСМ-6.

Насколько мне известно, ни в отечественной, ни в зарубежной литературе по программированию нет аналогов этой книге. Поэтому не сомневаюсь, что она с интересом будет воспринята и теми, для кого программирование — профессия, и теми, для кого оно всего лишь хобби.

Ю. М. Баяковский

ПРЕДИСЛОВИЕ

Годами накопленный опыт может поспорить со знанием.

Р. Акофф

На ошибках учатся — гласит народная мудрость, но лучше учиться на чужих ошибках, чем на собственных. Эта мысль и побудила авторов написать книгу, которую читатель держит в руках. Книга преследует несколько целей. Во-первых, обратить внимание программиста на те «подводные камни», которые могут встретиться на его пути при создании программы, научить его вдумчивому обращению с таким мощным инструментом, как ЭВМ. Во-вторых, предостеречь его от многих ошибок, которые допускает каждый программист. В-третьих, обратить внимание программиста на ту операционную среду, в которой происходит выполнение программы, влияние этой среды на исполнение программы и ее реакцию на ошибки в программе. Программное обеспечение, с которым имеет дело программист, создается человеком (не застрахованным от ошибок). Поэтому ошибки программиста могут выявить ошибки или недоработки системных программ.

Долгое время работа на вычислительной машине была уделом немногих — профессиональных программистов. С появлением микро-ЭВМ и персональных ЭВМ число людей, общающихся с вычислительной техникой, увеличивается. Это связано еще и с введением курса по информатике и вычислительной технике в средней школе. Конечно, это следствие широкого применения ЭВМ в нашей жизни — от прежнего использования ЭВМ как вычислителя до управляющих микро-ЭВМ в станках и автомобилях, от информационных ЭВМ до сетей вычислительных машин коллективного пользования.

В условиях широкого внедрения вычислительной техники во все сферы жизни постоянно повышается роль человека, работающего на ЭВМ, возрастает уровень требований к его профессиональной подготовке и психологическим качествам. Человек создает алгоритм решения задачи, переводит его на язык ЭВМ, тестирует алгоритм и программу и проводит вычисления. Иногда решение задачи разделяется между несколькими специалистами различного профиля: проблемным экспертом, алгоритмистом, программистом.

Книга обобщает опыт работы авторов по разработке программных комплексов (как системного, так и прикладного назначения). Работа по созданию программных комплексов предполагает умение составлять программы на нескольких языках программирования, а также хорошее

знание прикладной области, для которой это программное обеспечение создается.

Создание программ — сложный процесс, требующий больших затрат энергии (как умственной, так и физической). Даже при разработке программ средней сложности, как показывает практика, программисты высокой квалификации допускают ошибки. Характер и количество ошибок определяются в первую очередь квалификацией программиста. При этом не бывает ошибок незначительных. Ведь даже пропуск запятой, случайная замена ее точкой или отсутствие пробела между символами часто делает программу неработоспособной и может привести к тяжелым последствиям. Так, в [28] приведен пример того, как в программе на фортране, управлявшей первым полетом американского космического корабля на Венеру, оператор цикла был записан следующим образом:

```
DO 3 I=1.3
```

Транслятор воспринял его как оператор присваивания, так как вместо запятой, разделяющей параметры цикла, стоит точка. Это и привело к неудаче в запуске. К сожалению, практически невозможно написать программу без ошибок. Это означает, что каждый программист неизбежно сталкивается с необходимостью отлаживать свои программы.

Языки программирования, как и естественные языки, имеют две составляющие: синтаксис и семантику. *Синтаксис* языка определяет правила записи отдельных конструкций, а *семантика* выражает значение (смысл) объектов языка и отражает законы композиции объектов. Программа, правильно написанная с точки зрения языка программирования, должна удовлетворять как синтаксическим, так и семантическим правилам языка. Одна из задач транслятора с любого языка программирования заключается в обнаружении (по возможности) всех ошибок (нарушений синтаксических или семантических правил языка) в программе. Транслятор, как правило, легко выявляет синтаксические ошибки и гораздо труднее семантические. Это приводит к тому, что получаемые в результате трансляции программы могут содержать ошибки, которые проявляются только в процессе исполнения.

Практика программирования показывает, что, несмотря на большое число программистов и количество создаваемых ими программ, различных типов ошибок не слишком много. Под ошибками понимаются такие некорректности в программе, не связанные с алгоритмом задачи, которые не замечает транслятор и компоновщик-загрузчик и которые проявляются в процессе работы программы.

По-видимому, один из лучших способов научить программиста не делать ошибок — показать ему, как они могут появиться. Если программист знает, с какими ошибками он может столкнуться, ему легче их не допустить или исправить. Именно, если программист будет знать, как можно вызвать ошибочные ситуации преднамеренно, то с большей вероятностью они не возникнут у него случайным образом.

Программист с достаточным опытом накопил свой собственный набор ситуаций, в которых могут быть допущены ошибки. Такой набор позволяет достаточно быстро и эффективно находить ошибки и отлаживать программы.

Книга посвящена вопросам возникновения, обнаружения и предотвращения семантических ошибок в программах, написанных на языке фортран.

Книга содержит пять глав, три приложения и список литературы. В гл. 1 кратко рассмотрены основные этапы разработки программ для ЭВМ и связанные с этим проблемы. Глава 2 содержит условия задач. Приведенные задачи делятся на две категории. В задачах первой кате-

гории требуется найти ошибку в приведенной программе исходя из печатаемых результатов. Эти задачи по своему характеру непосредственно соответствуют проблемам, с которыми сталкиваются программисты при отладке программ. Ко второй категории относятся задачи на конструирование «аварийной» ситуации. В этих задачах требуется самому написать программу, выполняющую действия, иногда не совсем обычные для фортрана.

Последовательность, в какой представлены задачи в книге, носит случайный характер. Авторы сознательно отказались от какой-либо классификации условий задач с целью максимально приблизить ситуацию к реальной обстановке, лишив тем самым читателя подсказки. К тому же однозначную классификацию провести сложно. В этом читатель сможет убедиться в процессе ознакомления с задачами. Подавляющая часть задач оригинальна, идеи некоторых задач взяты из литературных источников либо из бесед с коллегами. Программы, приведенные в условиях задач и их решениях, были оттранслированы: на БЭСМ-6 — транслятором фортран-Дубна, на ЕС ЭВМ (ЕС-1045) — фортран IV, на СМ-2 — транслятором фортран II. В тех случаях, когда были использованы трансляторы для других типов ЭВМ, сделаны специальные оговорки.

Глава 3 не содержит в явном виде указаний к решению задач. Приводится систематическое рассмотрение основных источников ошибок в программах на фортране и формы их проявления. Начинающий программист, при вдумчивом прочтении, найдет здесь ключ к решению задач, вызвавших у него затруднения. Более опытному программисту представится целостная картина «опасных» мест при написании программ на фортране.

Ответы на задачи и некоторые комментарии к ним приведены в гл. 4.

Глава 5 посвящена программированию как искусству писать правильные программы. В ней приведен набор полезных рекомендаций для начинающих программистов.

В приложениях даны: реализация операторов управления языка фортран-Дубна на БЭСМ-6, особенности выполнения арифметических операций на БЭСМ-6 и методика поиска ошибок по информации, предоставляемой операционной системой Диспак.

Некоторые задачи могут показаться читателю странными, а программы, приведенные в них, примитивными. Авторы понимают, что никто специально таких программ писать не будет, однако программисты могут допустить ошибки, описанные в приведенных в книге задачах. Реальные программы обычно велики и довольно громоздки, поэтому для краткости и большей ясности из приводимых в задачах программ убрано все лишнее, не относящееся к ошибке, которой посвящена задача. Получившиеся при этом программы становятся короткими и схематичными. Эти программы следует рассматривать как краткие описания идей ошибок. Несмотря на сделанные упрощения, ошибки в программах не стали очевидными и их обнаружение в большинстве случаев представляет непростую задачу. Проверяйте и пытайтесь разобраться. Если вы научитесь вниманию и осторожности на ошибках в предложенных программах, то вам будет гораздо проще проверить свои собственные программы, найти и исправить в них ошибки.

Несмотря на то что книга написана для языка фортран, многое из приведенного в ней в той или иной форме может быть перенесено и на другие языки программирования. Читателю, интересующемуся возможными ошибками-ловушками в программах, написанных на других языках программирования, необходимо тщательно разобраться в

идеях, изложенных в гл. 3 и 4, и построить набор задач, аналогичных предложенным в гл. 2. Основные идеи неизменны, а реализации различны даже для трансляторов с рассмотренного языка программирования фортран.

Эта книга адресована в первую очередь начинающим программистам, которым она поможет более глубоко узнать используемые ими языковые средства и лучше понять сложный процесс программирования и отладки программ, а также избавит их от неприятных встреч со многими ошибками. Более опытным программистам книга даст возможность проверить уровень своей практической подготовки. Преподавателям программирования она поможет акцентировать внимание на некоторых не всегда понятных учащимся местах.

Приведенные задачи требуют от читателя умения логически и нешаблонно мыслить (подходя к проблеме с разных точек зрения), проведения скрупулезного анализа (не пропуская при этом кажущихся мелочей), выдвижения смелых гипотез, объясняющих явление, и доказательства либо опровержения этих гипотез.

Книгу практически можно читать в произвольном порядке. Читатель, имеющий некоторый опыт программирования на языках высокого уровня, может сразу начать с задач, обращаясь при возникновении затруднений к гл. 3. Новичку вначале лучше ознакомиться с гл. 1 и 3 и лишь затем перейти к решению задач. Решения приводятся в гл. 4. В гл. 5 приведены некоторые рекомендации, позволяющие избежать ошибок при программировании.

Рекомендуем читателю постоянно иметь под рукой какое-либо достаточно полное пособие по программированию на фортране (см., например, [8, 11, 22, 27, 41, 42]), так как при чтении и особенно при решении задач могут возникать различные вопросы относительно синтаксиса и употребления операторов языка фортран, организации программ и т. п.

Авторы выражают благодарность чл.-кор. АН СССР Д. Е. Охоцимскому за внимание и поддержку в работе, чл.-кор. АН СССР Л. Н. Королеву, который своими пожеланиями и советами стимулировал работу над книгой, титульному редактору Ю. М. Баяковскому за полезные замечания.

Кроме того, авторы выражают свою благодарность Е. И. Кугушеву, предложившему идеи некоторых задач и высказавшему ряд полезных советов и замечаний по рукописи книги.

Авторы будут признательны всем читателям, которые выскажут свои замечания и пожелания, а также сообщат обо всех обнаруженных ошибках или неясностях.

ГЛАВА 1

ОТ ЗАДАЧИ К ПРОГРАММЕ

Творчество есть выражение основных законов вселенной. Каждая истинная работа имеет свою красоту.

Н. К. Рерих

Деятельность программиста начинается с задачи. Именно необходимость решать сложные конкретные задачи, возникающие в различных областях науки, техники, производства, заставляет все большее число специалистов прибегать к помощи ЭВМ. И здесь возникает проблема: как с помощью ЭВМ решить поставленную задачу? Указанная проблема занимает центральное место среди проблем использования ЭВМ. Практически вся история развития вычислительной техники и

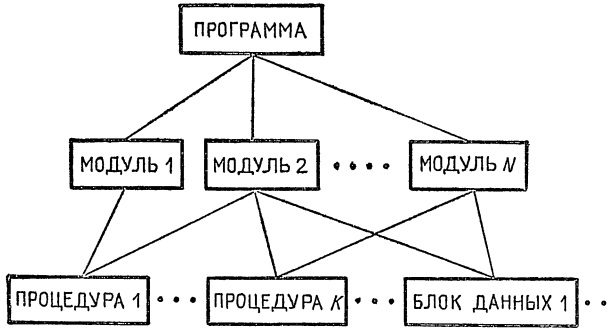

Рис. 1

информатики — это последовательность конкретных этапов на пути к решению этой проблемы, направленных на разработку языковых средств общения человека с ЭВМ. Одним из основных этапов решения задачи с помощью ЭВМ является разработка программы.

Программа — это описание на языке программирования процесса решения задачи на ЭВМ, начиная с ввода исходных данных и кончая

выводом результатов решения. В зависимости от используемых языковых средств, программа может состоять из одного или нескольких отдельно транслируемых модулей (рис. 1). Каждый модуль, в свою очередь, содержит описание одной или нескольких процедур, функций, блоков данных.

В программе на языке фортран каждый модуль содержит описание лишь одной программной единицы одного из четырех видов:

1. главный модуль программы (PROGRAM),
2. подпрограмма (SUBROUTINE),
3. подпрограмма-функция (FUNCTION),
4. общий блок данных (BLOCK DATA).

Разработка программы — сложный и трудоемкий процесс. В таблице 1.1 перечислены основные этапы решения задачи на ЭВМ и, в частности, разработки программы, начиная с постановки задачи и кончая интерпретацией получаемых результатов вычислений.

Таблица 1.1

Основные этапы решения задачи на ЭВМ

№	Этап	Форма представления
1	Осознание проблемы и формулировка задачи, нацеленной на ее решение	Естественный язык
2	Формальная постановка задачи	Естественный язык+язык математики
3	Разработка метода и алгоритма решения задачи	Язык математики
4	Разработка структур данных и процедур для работы с ними. Проектирование общей организации программы	Псевдоязык программирования
5	Перевод алгоритма на язык программирования	Язык программирования
6	Разработка системы тестов и отладка программы	Язык программирования+машинный язык
7	Решение поставленной задачи и получение результатов вычислений	Язык таблиц, графиков, рисунков
8	Обработка результатов вычислений и их интерпретация	Естественный язык+язык математики

Непосредственно к процессу разработки программы относятся этапы 4—6, но и этапы 2—3 оказывают на него определенное влияние. Каждый этап разработки программы характеризуется решениями по конкретизации программы, которые принимаются на этом этапе, и

формой представления программы. Рассмотрим более подробно некоторые из этапов.

Начнем с третьего этапа — этапа разработки метода и алгоритма решения задачи. Мы не случайно не отнесли этот этап к разработке программы. Существует важное различие между алгоритмом и программой. И это различие не только по форме, но и по содержанию. Программа — это не просто алгоритм, записанный на языке программирования. Программа — это результат отображения алгоритма на конкретную вычислительную среду, учитывающий все ее возможности и ограничения. Поясним сказанное.

Разрабатывая, например, алгоритм вычисления корней уравнения, мы не задумываемся о том, какие программные средства будут использоваться для обеспечения необходимой точности вычислений. Однако при записи этого алгоритма на языке программирования необходимо определить, с какой точностью будут представляться данные и как необходимо организовать эти вычисления, чтобы обеспечить эту заданную точность. Так, например, в фортране надо выбрать типы переменных REAL или DOUBLE PRECISION. Другой пример. Решая задачу, требующую большого времени вычислений, мы должны позаботиться о сохранении промежуточных результатов на внешних носителях на случай возможного сбоя в работе ЭВМ или на случай проведения вычислений в несколько приемов. Еще пример. Любая ЭВМ может работать лишь с числами в ограниченном диапазоне. Если в задаче приходится иметь дело с числами, выходящими за допустимый диапазон, то необходимо разработать дополнительно специальные способы представления данных и/или изменить алгоритм решения задачи.

Результатом третьего этапа является алгоритм, представленный на естественном языке с использованием математической терминологии и символики.

Четвертый этап, с которого, собственно, и начинается разработка программы, является самым ответственным. Именно на этом этапе принимаются основные решения по организации программы: происходит разбиение программы на модули, определяется форма представления данных и способы их обработки. Например, множество можно представить с помощью массива, списка, хеш-таблицы, функции, в зависимости от того, какие операции будут производиться над этим множеством: поиск элемента, включение или исключение элемента, упорядочение элементов множества и т. д.

Результатом четвертого этапа является программа, записанная на псевдоязыке программирования, представляющем смесь естественного языка и языка программирования, а также некоторых других вспомогательных средств, например, блок-схем.

На пятом этапе происходит окончательная конкретизация программы на каком-либо языке программирования. Абстрактные структуры данных отображаются в конкретные структуры данных выбранного

языка, что позволяет конкретизировать и процедуры обработки этих структур. Так, например, абстрактную структуру типа стек на языке фортран можно реализовать несколькими способами: как массив с указателем вершины стека и как связанный однонаправленный список.

Шестой этап как бы подводит итог проделанной работе. Здесь определяется, насколько полученная программа соответствует поставленной задаче. Для этого составляется набор тестов, на которых легко предсказывается поведение правильной программы. Выбор системы тестов — ответственное и трудное дело. Практически бывает невозможно построить систему тестов, обеспечивающих полный контроль правильности работы всех частей программы [15]. В дальнейшем будут подробно обсуждены трудности, возникающие на этом этапе.

На каждом этапе решения поставленной задачи могут быть допущены ошибки. Эти ошибки проявляются на последующих этапах решения задачи. В этой книге внимание будет сосредоточено главным образом на ошибках, возникающих на этапах 4—5 и обнаруживаемых на шестом этапе в ходе отладки программы. При этом из рассмотрения исключены синтаксические ошибки в предположении, что они уже устранены и мы имеем дело с программой, успешно прошедшей трансляцию, компоновку и загрузку в память ЭВМ для исполнения. Нас будут интересовать ошибки в программе, проявляющиеся в процессе выполнения отлаживаемой программы и не связанные непосредственно с алгоритмом решения задачи.

Устранение обнаруженной ошибки может потребовать пересмотра решений, принятых на нескольких вышестоящих уровнях, и частичного повторения некоторых этапов. Таким образом, получается итерационный процесс, в ходе которого программист, обнаружив ошибку, вносит изменения в программу и вновь производит тестирование.

Несмотря на кажущуюся очевидность, само понятие ошибки в программе требует дополнительного объяснения. Разрабатывая программу для решения какой-либо задачи, программист мысленно строит модель, отражающую взаимосвязь входных данных программы и получаемого результата. Эта модель, как правило, является неполной и покрывает лишь небольшое множество возможных входных данных, для которых можно «вручную» получить требуемый результат. Именно эти данные и используются впоследствии в качестве тестов для отладки программы. Пропуская тесты, программист пытается выявить ошибки, содержащиеся в программе. При этом в качестве критерия ошибки используется следующий тезис: *программа содержит ошибку, если ее реальное поведение отличается от ожидаемого, созданного по умозрительной модели.* Установление факта несовпадения предполагаемого и реального поведения программы называется обнаружением или фиксацией ошибки. Диагностирование ошибки — это выявление и локализация причин, приводящих к появлению ошибки.

Рассмотрим, как возникают ошибки программирования. Большинство ошибок вносится в программу при отображении алгоритма на язык программирования. Типичная ошибка на этом этапе — это неполный учет ограничений, обусловленных языком программирования и конкретной ЭВМ, на которой будет решаться задача. Другая ошибка — это различное использование одних и тех же объектов в разных местах программы, например работа с простой переменной как с массивом. Практически в каждой программе (особенно на фортране) встречается ситуация, когда используется неописанный объект. В ряде случаев эта ошибка может быть обнаружена транслятором, но очень часто она остается незамеченной.

Много досадных ошибок возникает в процессе переноса текста программы на машинные носители: перфокарты, перфоленту, ввод с терминала на магнитные носители. Хотя возникающие опечатки довольно легко обнаруживаются, тем не менее они доставляют программисту немало хлопот.

Помимо ошибок в программе часто возникают ошибки в данных, используемых программой. Выявление этих ошибок практически полностью ложится на обрабатывающую программу. Об этом надо всегда помнить и предусматривать в программе средства контроля правильности используемых данных.

Следующий вопрос, который будет рассмотрен,— каким образом производится обнаружение ошибок в программе?

Прежде чем приступить к выполнению предписанных действий, программа проходит через обработку ряда системных программ: транслятора, редактора внешних связей (компоновщика), загрузчика. Кроме того, процесс исполнения программы контролируется операционной системой и поддерживается аппаратными средствами.

При обработке текста программы транслятором выполняется синтаксический и, частично, семантический контроль правильности программы. Так как синтаксис языка строго формализован, то практически все синтаксические ошибки достаточно эффективно выявляются за несколько предъявлений программы транслятору. К сожалению, ни один из известных авторам трансляторов по тем или иным причинам не осуществляет полного синтаксического анализа программы, что оставляет возможность проникновения ошибок в программу. Следует также учитывать, что транслятор, как и другие упоминавшиеся системные программы,— это обычные программы, а значит, они сами могут содержать ошибки (правильнее было бы сказать — содержат).

Семантический контроль правильности программы гораздо более сложен, трудоемок и менее формализован. Кроме того, он осуществляется лишь в рамках одного отдельно транслируемого модуля. При этом ошибки, возникающие на «стыке» модулей (ошибки интерфейса), остаются незамеченными. Источники таких ошибок в программах на фортране обсуждаются в гл. 3.

После того как оттранслированы все модули программы, наступает очередь работы редактора внешних связей. Основная цель этой программы — связать в единое целое отдельные модули программы, организовав взаимодействие между ними. Кроме модулей, написанных пользователем, к программе подсоединяются некоторые стандартные системные модули, обеспечивающие, например: ввод-вывод данных, вычисление стандартных функций. Редактор внешних связей, в свою очередь, выявляет некоторые семантические ошибки интерфейса между модулями.

К числу ошибок, обнаруживаемых редактором внешних связей, могут относиться, например: несоответствие длины общего блока памяти в разных модулях, различное число параметров в описании подпрограммы и в обращении к ней и некоторые другие ошибки. Хотя в процессе компоновки программы могут быть выявлены многие из имеющихся в программе ошибок, тем не менее ни один из известных авторам редакторов внешних связей по разным причинам этого не делает. Основная трудность заключается в сопряжении программ, написанных на различных языках программирования, и в значительных накладных расходах, связанных с достаточно полным контролем. В настоящее время, когда стало очевидным, какие огромные трудозатраты приходятся на отладку программ, а технические возможности ЭВМ сильно увеличились, наметился определенный поворот в сторону разработки программно-аппаратных средств контроля правильности выполнения программ.

В некоторых реализациях языка фортран (а также в других языках программирования) имеются средства контроля за исполнением программы, например контроль выхода значения индекса за границы массива. Наличие такого контроля облегчает отладку программ. Однако это не решает проблему полностью, так как во время исполнения программы средства контроля исключаются из программы в целях экономии процессорного времени и памяти (в большинстве серийных ЭВМ эти средства аппаратно не поддерживаются).

Как следует из приведенного «определения» ошибки, не существует какого-либо формального способа установления факта наличия ошибок в программе. Все сводится к такой довольно расплывчатой категории, как «поведение программы», что само по себе ничего не дает для практики отладки программ.

Некоторую помощь в выявлении ошибок оказывают аппаратные и системные средства контроля правильности выполнения программы. Ряд ошибок может быть выявлен с помощью специальных систем отладки программ. Ниже приводится список «симптомов», указывающих на наличие ошибок в программе. Предлагаемый список не претендует на полноту. Возможности выявления ошибок в программе во многом зависят от средств контроля, имеющихся в аппаратной части ЭВМ и в операционной системе [18, 27, 35, 38] (см. приложение 3).

Основные формы проявления ошибок

1. Ошибки при выполнении арифметических операций и стандартных функций:
   1.1. деление на нуль,
   1.2. переполнение (как положительное, т. е. результат вычислений по модулю больше максимально возможного представления числа в ЭВМ, так и отрицательное, т. е. результат вычислений по модулю меньше минимально возможного отличного от нуля числа, представимого в ЭВМ),
   1.3. отличие результата арифметических операций от ожидаемого,
   1.4. обращение к стандартным функциям с недопустимыми значениями параметров.
2. Ошибки, связанные с затиранием команд и переменных.
3. Ошибки управления:
   3.1. зацикливание — бесконечное повторение одной и той же части программы,
   3.2. последовательность прохождения участков программы не соответствует ожидаемому,
   3.3. потеря управления, приводящая к ошибкам разного рода, например к таким, как обращение к запрещенной области памяти, попытка выполнить запрещенную команду или «не команду» и т. п.
4. Ошибки ввода-вывода:
   4.1. «странная» печать,
   4.2. сообщения об ошибках от системных программ ввода-вывода.

В процессе отладки и исполнения программы программист фактически имеет дело не только с программой, написанной на языке фортран, а и с программой в кодах ЭВМ, полученной в результате трансляции, компоновки и загрузки отдельных модулей, написанных программистом, и системных модулей. Поэтому для успешного проведения отладки программист должен хорошо представлять:

1) какую архитектуру имеет ЭВМ, на которой производится отладка и выполнение программы, и особенности выполнения команд обработки данных,

2) как транслируется программа на машинный язык,

3) как представляется программа в кодах ЭВМ,

4) как организована программа в памяти ЭВМ,

5) как происходит выполнение программы и как осуществляется взаимодействие между отдельными модулями,

6) как происходит взаимодействие программы с операционным окружением.

Архитектура современных ЭВМ весьма разнообразна [26, 29, 39, 40]. С точки зрения реализации языка фортран наиболее важными явля-

ются возможности представления стандартных типов данных языка и работа с ними (это касается вычислений с плавающей точкой).

В настоящее время теория и практика трансляции языков программирования является достаточно развитой дисциплиной [14, 31, 32]. Наиболее распространенными являются методы трансляции, основанные на целостном переводе программы, т. е. когда при трансляции каждого оператора программы учитывается не только его конкретное содержание, но и его место в программе и взаимосвязь с другими операторами. Цель таких методов трансляции — улучшить рабочие характеристики программы такие, как время выполнения и объем используемой памяти. Типичным примером применения такой методики являются оптимизирующие трансляторы [14, 31]. Существуют различные точки зрения на достоинства и недостатки применения таких методов трансляции [28]. Главным недостатком этих методов является отсутствие взаимно однозначного соответствия между операторами языка в программе и машинным кодом для этих операторов. Транслятор генерирует код, эквивалентный с функциональной точки зрения всей программе в целом, а не отдельному оператору программы. Такая ситуация существенно затрудняет процесс отладки программы, так как, локализовав ошибку в машинном коде, бывает трудно найти соответствующее ей место в исходной программе на фортране.

Как правило, машинный язык настолько богат, что один и тот же оператор языка фортран может быть отображен в различные последовательности кодов машинного языка (см. гл. 3 и Приложение 1, [26, 29, 32, 39, 40]). С функциональной точки зрения все эти отображения эквивалентны между собой, пока программа работает в «предусмотренных» рамках. Но в процессе отладки программист как раз сталкивается с нестандартными ситуациями и различия в представлении операторов языка уже играют роль. Знание используемого способа представления операторов языка помогает правильно диагностировать ошибки. Но иногда различие в трансляции приводит и к различным результатам вычислений. Рассмотрим в качестве примера следующий фрагмент программы:

```
      A=1./3.
      IF(A .EQ. 1./3.) PRINT 1
    1 FORMAT (' A=1./3.')
```

При выполнении этого фрагмента программы, полученной с помощью транслятора фортран-Дубна, оператор печати PRINT будет обойден, а при использовании транслятора с языка форекс будет напечатано:

```
A=1./3.
```

(В обоих случаях программа выполняется на БЭСМ-6.) Это различие результатов объясняется тем, что транслятор с языка форекс при трансляции вычисляет все выражения, состоящие из констант, и подставляет на их место в программе результат вычисления. Транслятор

с языка фортран-Дубна этого не делает. Он просто вычисляет эти выражения при выполнении программы. Поэтому выражение 1./3. в операторе IF будет по-разному обработано двумя указанными трансляторами. Для транслятора с языка форекс приведенный фрагмент эквивалентен следующему:

```
      A=0.333333E 0
      IF(0. .EQ. 0.) PRINT 1
    1 FORMAT(' A=1./3.')
```

Транслятор заменяет константное выражение 1./3. на константу 0.333333E 0. Сравнение двух чисел производится путем вычитания второго числа из первого и сравнения результата с нулем. Так как оба числа являются константами, транслятор выполняет вычитание, получая в результате константу 0. Поэтому в операторе IF происходит сравнение нуля с нулем.

Для транслятора фортран-Дубна фрагмент программы эквивалентен следующему:

```
      A=1./3.
      IF((—1.)/3.+A .EQ. 0.) PRINT 1
    1 FOPMAT(' A=1./3.')
```

При вычислении по программе значение переменной A имеет 40 разрядов, а результат вычисления выражения (—1.)/3. в операторе IF имеет 80 разрядов, так как в процессе вычисления арифметических выражений используется регистр младших разрядов. Эти дополнительные разряды и особенности представления чисел в ЭВМ приводят к несовпадению значений A и 1./3..

Кроме того, всегда важно помнить следующий момент. Трансляция каждого модуля происходит отдельно от других модулей программы. Это значит, что информация, содержащаяся в одном модуле, недоступна для других модулей во время трансляции.

Теперь об организации программы. В языке фортран определены следующие классы объектов:

1) модуль — главный модуль (в некоторых диалектах языка он имеет заголовок PROGRAM), подпрограмма (SUBROUTINE), функция (FUNCTION), блок данных (BLOCK DATA),

2) общий блок памяти (COMMON),

3) оператор-функция,

4) стандартная функция или подпрограмма,

5) переменная (массив или простая переменная),

6) константа,

7) метка,

8) оператор

В программе, которая выполняется ЭВМ, имеется, как правило, лишь один класс объектов — слова памяти или части слов, представ-

ленные своим адресом (и смещением внутри слова). Таким образом, вся совокупность классов объектов языка фортран отображается на единственный класс — класс адресов. Такая ситуация, когда в одну кучу сваливаются «и сапоги и яичница», является источником многих ошибок. Представление объектов всех классов лишь одним адресом приводит к их обезличиванию. Транслятор ставит в соответствие каждому объекту программы некоторое множество слов памяти (или частей слов), заполняя их соответствующим кодом [14]. Доступ к нему осуществляется по адресу одного из слов. Обычно транслятор отводит под объект несколько последовательных слов памяти, а доступ осуществляется по адресу первого слова. При этом транслятор может группировать в одном месте память, выделяемую объектам одного класса. Например, как правило, память при трансляции операторов FORMAT отводится в одном месте — где-нибудь в конце модуля, хотя в тексте модуля, написанном на языке фортран, они могут быть разбросаны произвольным образом. Объединяя объекты одного класса, транслятор формирует так называемые секции программного модуля. Перечислим некоторые из них:

— секция команд,
— секция констант,
— секция встроенных функций,
— секция форматов ввода-вывода,
— секция переменных,
— секция рабочей памяти.

Ниже приведен пример программы на языке фортран и возможная организация программы на машинном языке:

```
01        PROGRAM MAIN
02        DIMENSION A(10),C(5)
03        DIMENSION L(5)
04        COMMON A,B,K
05        DATA L/1,2,3,4,5/
06        EXTERNAL S
07        DO 1 N=1,10
08      1 A(N)=FUN(N)+FLOAT(N)
09        CALL SUBR(S,A,K)
10        IF(K-7) 10,2,10
11     10 WRITE(6,3) A
12      3 FORMAT(' A=', 5E10.3)
13      2 STOP
14        END
```

Результат трансляции приведенной программы в машинный код на ЭВМ серии АСВТ:

```
00000  000000  BSS 000017      секция переменных    отведение памяти
00017  000000C DEF 000000        адрес массива A
00020  000000P DEF 000000        адрес массива C
```

```
00021  000012R DEF 000012      адрес массива L
00022  000000  OCT 000000     входная точка программы
00023  000000  ORG 000012     оператор 05
00012  000001  OCT 000001     секция данных
00013  000002  OCT 000002
00014  000003  OCT 000003
00015  000004  OCT 000004
00016  000005  OCT 000005
00017  000000  ORG 000023     секция команд
00023  062126R LDA 000126     оператор 07
00024  072121R STA 000121       N=1 начальная засылка счетчика
00025  062121R LDA 000121     оператор 08
00026  042127R ADA 000127       вычисление адреса A(N)
00027  001200  OCT 001200
00030  042017R ADA 000017
00031  072122R STA 000122       сохранение адреса в рабочей памяти
00032  016001X JSB  FUN         обращение к функции FUN
00033  000035R DEF 000035       адрес возврата
00034  000121R DEF 000121       адрес параметра N
00035  016002X JSB  .DST        сохранение результата в рабочей памяти
00036  000123R DEF 000123
00037  062121R LDA 000121       вычисление функции FLOAT(N)
00040  016003X JSB  FLOAT
00041  016004X JSB  .FAD        добавление предыдущего результата
00042  000123R DEF 000123       вычисления функции FUN(N)
00043  016002X JSB  .DST        сохранение в памяти
00044  100122R DEF 000122,I     по адресу A(N)
00045  062121R LDA 000121       завершение цикла: увеличение N на 1
00046  002004  OCT 002004
00047  072121R STA 000121
00050  003004  OCT 003004       инверсия параметра цикла и добавление
00051  042130R ADA 000130       константы конца цикла
00052  002021  OCT 002021       пропуск команды, если результат < 0, пе-
00053  026025R JMP 000025       реход на продолжение цикла
00054  016005X JSB  SUBR      оператор 09 обращение к подпрограмме
00055  000061R DEF 000061       адрес возврата
00056  000006X DEF S            адрес первого параметра
00057  000000C DEF 000000       адрес второго параметра
00060  000026C DEF 000026       адрес третьего параметра
00061  062026C LDA 000026     оператор 10
00062  003004  OCT 003004       вычисление выражения K-7
00063  042131R ADA 000131
00064  003004  OCT 003004
00065  002020  OCT 002020       пропуск команды, если результат ⩾ 0,
00066  026072R JMP 000072       переход на метку 10
00067  002002  OCT 002002       пропуск команды, если результат = 0,
00070  026072R JMP 000072       переход на метку 10
00071  026114R JMP 000114       переход на метку 2
00072  062125R LDA 000125     оператор 11
00073  006400  OCT 006400       печать по устройству № 6 массива A
00074  016007X JSB  .DIO.
00075  000104R DEF 000104
00076  000103R DEF 000103
00077  062130R LDA 000130
00100  066017R LDB 000017
00101  016010X JSB  .RAR.
```

```
00102  016011X JSB   .DTA.
00103  026114R JMP   000114      оператор 12 обход текста FORMAT (форматы на ACBT не выносятся в отдельную секцию)
00104  024040    OCT  024040      текст (
00105  023440    OCT  023440      текст '
00106  040475    OCT  040475      текст A =
00107  023454    OCT  023454      текст ,
00110  032505    OCT  032505      текст 5E
00111  030460    OCT  030460      текст 10
00112  027063    OCT  027063      текст .3
00113  024400    OCT  024400      текст )
00114  002400    OCT  002400    оператор 13 установка номера STOP
00115  016012X   JSB   .STOP       выполнение останова
00116  016013X   JSB   EXEC       возврат в ДОС ACBT
00117  000121R   DEF  000121
00120  000125R   DEF  000125
00121  000000    BSS  000004    секция рабочих переменных
00125  000006    OCT  000006    секция констант
00126  000001    OCT  000001
00127  177777    OCT  177777
00130  000012    OCT  000012
00131  000007    OCT  000007
                 TRA  MAIN
*** END
 SYMBOL  TABLE                 таблица соответствия идентификаторов программы адресам памяти относительно начала программы
MAIN    R  000022
A       C  000000
C       R  000000
L       R  000012
B       C  000024U
K       C  000026U
S       X  177777
N       R  000121
FUN     X  177777
SUBR    X  177777
00001   R  000025
00010   R  000072
00002   R  000114
00003   R  000104
10000   R  000103
10001   R  000114
```

Результат трансляции той же программы на промежуточный язык (автокод мадлен [27, 36]) на БЭСМ-6 (нумерация операторов на БЭСМ-6 производится только для выполняемых операторов, в скобках приведена общая нумерация строк программы):

```
MAIN     :  ,NAME ,
**       : LC,BLOCK,
             , CONT,   A      (1)
             , CONT,   B      (1)
             , CONT,   K      (1)
FT*571   :  , SUBP,
FT*002   :  , SUBP,
FT*003   :  , SUBP,
```

```
          STOP*     :   , SUBP,
          S         :   , SUBP,
          SUBR      :   , SUBP,
          FLOAT     :   , SUBP,
          FUN       :   , SUBP,
          *C        :   , EQU  ,   * +45
0000      PROGRAM   :   ,ENTRY,                  секция команд
0000 0001               14 ,VTM  ,   MAIN        адрес начала на 14
                                                 регистр
—    0042                  ,UJ   ,   *BEGIN      выполнение входного блока
0001      MAIN      :   ,BSS  ,
          C*****НАЧАЛО ОПЕРАТОРА 1 (07)
0001                       ,XTA  , = 64000000    00000001
—                          ,ATX  ,   N           начальное присваивание
                                                 N=1
0002                       ,NTR  ,   3
—                          ,NTR  ,   18
0003      *W0001    :      ,BSS  ,
          C*****НАЧАЛО ОПЕРАТОРА 2 (08)
0003      *1        :      ,BSS  ,
0003 0061               :14 ,VTM  ,   N           засылка адреса N в стек
—                          ,ITA  ,   14
0004                      13,VJM  ,   FUN         исполнение функции FUN
—
0005                       ,ATX  ,   *ERR00       сохранение результата
—    0061                 14,VTM  ,   N
0006                       ,ITA  ,   14          засылка адреса N в стек
—                         13,VJM  ,   FLOAT       вычисление функции
                                                 FLOAT
0007                       ,A+X  ,   *ERR00       добавление FUN(N)
—    0061                  ,WTC  ,   N           вычисление адреса A(N)
0010                       1,VTM  ,
—                          1,UTC  ,   A—1         сохранение суммы по
                                                 адресу A(N)
0011                       ,ATX  ,
—                          ,XTA  ,   N           завершение цикла:
0012                       ,NTR  ,   3
—                          ,A+X  , = 64000000    00000001
0013                       ,ATX  ,   N           N=N+1
—                          ,X—A  , = 64000000    00000012
0014                       ,NTR  ,   18          проверка на окончание
—    0003                  ,UZA  ,   *W0001      продолжение цикла
          C*****НАЧАЛО ОПЕРАТОРА 3 (09)
0015                      14,VTM  ,   S           адрес S в стек
                           ,ITS  ,   4
0016                      14,VTM  ,   A           адрес A в стек
—                          ,ITS  ,   14
0017                      14,VTM  ,   K           адрес K в стек
—                          ,ITS  ,   14
0020                      13,VJM  ,   SUBR        выполнение подпрограм-
                                                 мы
          C*****НАЧАЛО ОПЕРАТОРА 4 (10)
0021                       ,UTC  ,   K
—                          ,XTA  ,                вычисление выражения
                                                 K—7
0022                       ,NTR  ,   3
—                          ,A—X  , = 64000000    0000007
```

```
0023                    ,NTR   ,   18
 —                      ,ASN   ,   57
0024  0041              ,UZA   ,   *2          переход на метку 2
                                                при =0
 —                      ,ARX   ,
0025  0026              ,UZA   ,   *10
      C*****НАЧАЛО ОПЕРАТОРА 5 (11)
 —
0026      *10       :  ,BSS   ,               печать массива A
0026  0052             14,VTM   ,   /3
 —                      ,ITS   ,   14
0027                   13,VJM   ,   FT*571
 —
0030                    ,XTA   , = 64000000    00000001
 —                      ,ATX   ,   IMPL*1
0031                    ,NTR   ,   3
 —                      ,NTR   ,   18
0032      *W0002    :  ,BSS   ,
0032  0062              ,WTC   ,   IMPL*1
 —                     1,VTM   ,
0033                   1,UTC   .   A—1
 —                     14,VTM   ,
0034                    ,ITS   ,   14
 —                     13,VJM   ,   FT*002
0035                    ,XTA   ,   IMPL*1
 —                      ,NTR   ,   3
0036                    ,A+X   , = 64000000    00000001
 —                      ,ATX   ,   IMPL*1
0037                    ,X—A   , = 64000000    00000012
 —                      ,NTR   ,   18
0040  0032              ,UZA   ,   *W0002
 —                     13,VJM   ,   FT*003
      C*****НАЧАЛО ОПЕРАТОРА 6 (13)
0041      *2        :  ,BSS   ,
0041                   13,VJM   ,   STOP*
 —
0042      *BEGIN    :  ,BSS   ,               секция входа в программу
0042                    7,MTJ   ,   8
 —                     14,MTJ   ,   10
0043  0055              7,VTM   ,   *C
 —                      ,ITS   ,   13
0044                    ,ITS   ,   1
 —                      ,ITS   ,   8
0045                    ,ITS   ,
 —                      ,NTR   ,   3
0046                    ,NTR   ,   2
 —                     10,UJ    ,
0047      *END      :  ,BSS   ,               секция выхода из про-
                                                граммы
0047                    ,STI   ,
 —                      ,STI   ,   7
0050                    ,STI   ,   1
 —                      ,STI   ,   8
0051                   8,UJ    ,
 —
0052      /3            ,OCT   ,   12023440    20236447 (оператор 12)
```

```
0053                    ,OCT   ,    13032505      14230056
0054                    ,OCT   ,    14624440      10020040
0060      *ERR00    :   ,BSS   ,    1             секция переменных
0061      N         :   ,BSS   ,    1
0062      1MPL*1    :   ,BSS   ,    1
0063      C         :   ,BSS   ,    5
0070      L         :   ,BSS   ,    5
                        ,DATA  ,
0060      *DATA     :   ,BSS   ,                  секция констант
0060                    ,OCT   ,    64000000      00000001
0061                    ,OCT   ,    64000000      00000002
0062                    ,OCT   ,    64000000      00000003
0063                    ,OCT   ,    64000000      00000004
0064                    ,OCT   ,    64000000      00000005
0065                   1,      ,    L
                        ,END   ,    MAIN
СТРУКТУРА ПОДПРОГРАММЫ:
        ЗАГОЛОВОК  00001
     ОБОЗНАЧЕНИЯ   00016
              —    00000
              —    00000
ИДЕНТИФИКАТОРЫ     00006
         КОМАНДЫ   00052
   ГРУППА BSS      00015
     КОНСТАНТЫ     00006
         ДАННЫЕ    00005
      РАССЫЛКИ     0000
```

Обратите внимание, в какую последовательность команд транслируются операторы языка фортран в приведенной программе.

Отметим некоторые важные моменты, связанные с процессом выполнения программы.

1. Доступ к любому объекту программы в процессе выполнения осуществляется по адресу объекта в памяти ЭВМ.

2. В процессе выполнения программы информация о классе объекта и его типе недоступна, так как после трансляции эта информация, как правило, не сохраняется.

3. Работа с каждым объектом производится на основе предположения, что информация о «правилах» работы с объектом в каком-то виде (в основном неявном) сосредоточена там, откуда произошло обращение к объекту. При этом никакой проверки соответствия класса объекта используемой процедуре обработки не производится.

Один из основных вопросов, относящихся к процессу исполнения программы,— это организация взаимодействия отдельных модулей между собой. Прежде всего это касается механизма передачи параметров. В каждой конкретной реализации языка этот вопрос решается по-своему и влияние принятого решения оказывается иногда значительным. Так, некоторые реализации допускают передачу параметров по ссылке и по значению, а также их варианты. Ряд реализаций допускает динамическую память для переменных модуля,

В последнее время большинство реализованных трансляторов имеют набор режимов генерации готовой программы (оптимизация, отладка, рабочий вариант). Каждый режим вносит свою особенность в выполнение программы, что следует также учитывать.

В заключение необходимо отметить следующее. Не существует двух абсолютно одинаковых реализаций трансляторов языка фортран. Различия могут быть весьма существенными, даже для нескольких трансляторов на одной и той же ЭВМ. Примером могут служить трансляторы фортран-Дубна, форекс и фортран-ГДР на БЭСМ-6 [3, 13, 27, 36]. Спектр различий довольно широк. В первую очередь различаются сами реализуемые диалекты языка. Каждая реализация содержит некоторые расширения языка и одновременно с этим некоторые ограничения по отношению к стандарту. Различия могут касаться типов данных (например, вводятся битовый и текстовый типы), состава управляющих операторов (например, структурные операторы выбора и цикла), набора стандартных функций, организации ввода-вывода. Причем одни и те же конструкции языка могут иметь несколько отличающийся синтаксис и, что хуже всего, различную семантику [3, 27, 28, 36]. Таким образом, программисту необходимо хорошо знать отличия от стандарта того диалекта входного языка транслятора, с которым он работает.

Результаты работы программы на различных ЭВМ (имеется в виду отличие не только в модели ЭВМ, но разные экземпляры ЭВМ одной и той же модели) могут различаться. В основном несовпадение результатов проявляется в вычислительных задачах и вызвано различной формой представления чисел с плавающей точкой, точностью вычислений, отличием в выполнении арифметических операций, различными алгоритмами перевода чисел из десятичной системы счисления в двоичную и другими особенностями архитектуры ЭВМ и реализации транслятора. Даже при вычислении на одной и той же ЭВМ по программам, оттранслированным различными трансляторами, возникают различия в результатах вычислений, что иногда приводит в недоумение начинающих программистов. Еще большее недоумение возникает в том случае, когда при изменении режима трансляции (например, переход от отладочного режима к рабочему режиму) изменяются результаты вычислений.

Все сказанное следует иметь в виду при решении задач, приведенных в книге, и разборе их решений.

ГЛАВА 2

ЗАДАЧИ-ЛОВУШКИ

Даже телефонный мастер может набрать не то, что нужно.

Р. Акофф

Во всех приводимых текстах программ введена нумерация строк. Номера строк используются для ссылок в условиях и решениях задач на конкретные операторы программ. Результаты, печатаемые программой, выделяются двумя звездочками в начале строки, а строки комментариев к печатаемым данным выделяются двумя символами С. Решения задач приведены в основном для БЭСМ-6, однако большинство решений с небольшими изменениями справедливо и для других ЭВМ. Везде, где это существенно, рассматриваются различия, возникающие при выполнении приведенных программ на ЕС ЭВМ по сравнению с БЭСМ-6. Программы, приведенные в условиях и решениях задач, оформлены в соответствии с правилами языка фортран-Дубна [23, 36, 37]. В случае использования транслятора с языка фортран на ЕС ЭВМ необходимо удалить из текста программы оператор заголовка PROGRAM и сделать некоторые изменения в операторе FORMAT [8, 13]. При переносе приведенных программ на другие ЭВМ необходимо учитывать особенности используемого диалекта языка фортран [13]. Для обозначения восьмеричных констант используется символ 'B', записываемый сразу за последней значащей цифрой константы.

Задача 1. Объясните, почему первым пяти элементам массива K в операторах 08 и 09 не присваиваются значения, равные 1?

```
01      PROGRAM TASK
02      DIMENSION K(10)
03      DATA K/10*2/
04      DO 1 J=1,5
05    1 K(J)=0
06      PRINT 2,K
07    2 FORMAT(' K=',10I2)
```

```
08      DO 3 J = 1,5
09    3 K(J) = 1
10      PRINT 2,K
11      STOP
12      END
```

```
** K = ␣0␣0␣0␣0␣0␣0␣2␣2␣2␣2␣2
** K = ␣0␣0␣0␣0␣0␣0␣1␣2␣2␣2␣2
```

Задача 2. Объясните, почему оператор 04 в приведенной программе не изменяет значения переменной ТОРМОЗ?

```
01      PROGRAM TASK
02      TOPMOЗ = 0.
03      PRINT 1,TOPMOЗ
04      TOPMOЗ = 1.0
05      PRINT 1,TOPMOЗ
06      STOP
07    1 FORMAT(' TOPMOЗ = ', F5.1)
08      END
```

```
** TOPMOЗ = ␣␣0.0
** TOPMOЗ = ␣␣0.0
```

Задача 3. Приведенная программа вычисляет сумму 1 000 000 первых членов гармонического ряда двумя способами: суммируя члены ряда в прямом и обратном порядке. Объясните, чем вызвано полученное различие результатов.

```
01      PROGRAM TASK
02      DATA A,B/0.,0./, M/1000 000/
03      DO 1 K = 1,M
04      A = A + 1./FLOAT(K)
05      B = B + 1./FLOAT(M — K + 1)
06    1 CONTINUE
07      PRINT 2,A,B
08    2 FORMAT('␣␣A = ',E18.12,'␣␣B = ', E18.12)
09      END
```

```
**␣␣A = ␣1.439271946924 + 01␣␣B = ␣1.439272606994 + 01
```

Задача 4. Подпрограмма FPRINT печатает таблицу значений функции FUNC, передаваемой ей в качестве параметра, для значений аргумента, задаваемых массивом X. Объясните, почему попытка напечатать таблицу значений стандартной функции SIN с помощью подпрограммы FPRINT приводит к ошибке.

```
01      PROGRAM TASK
02      REAL X(10)
03      DATA PI/3.1415/
04      DO 1 K = 1,10
```

```
05      1 X(K)=PI*FLOAT(K)/4.
06        CALL FPRINT(SIN,X)
07        END
    C ----------------------------------------
08        SUBROUTINE FPRINT(FUNC,X)
09        DIMENSION X(10)
10        DO 1 K=1,10
11        XK=X(K)
12        FX=FUNC(XK)
13      1 PRINT 2,XK,FX
14        RETURN
15      2 FORMAT('␣␣␣X=', E20.10,'␣␣␣␣FUNC=',E20.10)
16        END
```

Задача 5. Приведенная ниже программа присваивает элементам массива X значения по формуле:

$$X(K)=2^K, \quad \text{если } 1 \leqslant K \leqslant 5;$$
$$X(K)=3^{K-5}, \quad \text{если } 6 \leqslant K \leqslant 10.$$

Объясните, чем вызвано отличие полученного результата от ожидаемого.

```
01        PROGRAM TASK
02        INTEGER X(10)
03        DATA X/10*0/
04        DO 1 K=1,5
05      1 X(K)=2**K
06        DO 2 K=6.10
07      2 X(K)=3**(K-5)
08        PRINT 3,K,X
09      3 FORMAT('␣K=',I2/,'␣X=',10I3)
10        END
```

```
**␣K=␣6
**␣X=␣␣2␣␣4␣␣8␣16␣32␣␣3␣␣0␣␣0␣␣0␣␣0
```

Задача 6. Рассмотрите приведенную ниже программу. Она вычисляет сумму упорядоченных положительных чисел. Значения слагаемых заданы оператором DATA. Сумма чисел вычисляется в прямом и обратном порядке. Точное значение суммы заданной последовательности чисел равно 1111111.1111111. На ЭВМ БЭСМ-6 суммы чисел, вычисленных в порядке возрастания и убывания их величин, различаются в 12-й значащей цифре, на ЕС ЭВМ имеется совпадение по всем значащим цифрам. Все полученные значения суммы отличаются от точного значения. Объясните полученные результаты.

```
01        PROGRAM TASK
02        DIMENSION A(14)
```

```
03      DATA A/1.E+06, 1.E+05, 1.E+04, 1.E+03,
       *       1.E+02, 1.E+01, 1.E+00, 1.E-01,
       *       1.E-02, 1.E-03, 1.E-04, 1.E-05,
       *       1.E-06, 1.E-07/
04      N=14
05      S=0.
06      DO 1 I=1,N
07    1 S=S+A(I)
08      PRINT 3,S
09      S=0.
10      DO 2 I=1,N
11      J=N+1-I
12    2 S=S+A(J)
13      PRINT 4,S
14    3 FORMAT('␣СУММА␣В␣ПРЯМОМ␣␣␣ПОРЯДКЕ␣=
       *␣',E20.11)
15    4 FORMAT('␣СУММА␣В␣ОБРАТНОМ␣ПОРЯДКЕ␣=
       *␣',E20.11)
16      END
```

```
CC              НА ЭВМ БЭСМ-6:
**␣СУММА␣В␣ПРЯМОМ␣␣␣ПОРЯДКЕ␣=␣␣␣␣␣
  1.11111111110+06
**␣СУММА␣В␣ОБРАТНОМ␣ПОРЯДКЕ␣=␣␣␣␣␣
  1.11111111111+06
CC              НА ЕС ЭВМ:
**␣СУММА␣В␣ПРЯМОМ␣␣␣ПОРЯДКЕ␣=␣␣␣␣␣
   0.1111111+07
**␣СУММА␣В␣ОБРАТНОМ␣ПОРЯДКЕ␣=␣␣␣␣␣0.1111111+
+07
```

Задача 7. Напишите программу, в которой последовательность операторов

```
01      J=1
02      PRINT 1,J
03    1 FORMAT('␣J=',I1)
```

выдает на печать сообщение

```
**␣J=0
```

Задача 8. Напишите программу, которая затирает 100 слов памяти, расположенных следом за ней в памяти.

Задача 9. Почему при выполнении приведенной программы возникает аварийная ситуация?

```
01      PROGRAM TASK
02      CALL SUBR(X, Y)
03      STOP
```

```
04        END
    C ------------------------------------------
05        SUBROUTINE SUBR(X)
06        RETURN
07        END
```

Задача 10. Рассмотрите приведенную программу и объясните, чем вызваны такие результаты вычислений. Расчет выполнен на БЭСМ-6.

```
01        PROGRAM TASK
02        DATA A/1.E12/,B/1.E-12/
03        C=A+B
04        D=A-B
05        E=C-D
06        PRINT 1,A,B,C,D,E
07      1 FORMAT('␣A=',E20.13,'␣␣B=',E20.13/
08       *          '␣C=',E20.13,'␣␣D=',E20.13/
09       *          '␣E=',E20.13)
10        END
```

```
**␣A=␣␣1.0000000000000+12  B=␣␣9.9999999999947-13
**␣C=␣␣1.0000000000000+12  D=␣␣9.9999999999900+11
**␣E=␣␣1.0000000000000+00
```

Задача 11. Объясните, почему приведенная программа неверно вычисляет сумму первых 100 членов гармонического ряда:

$$1+1/2+1/3+\ldots+1/100$$

```
01        PROGRAM TASK
02        S=0.
03        DO 1 K=1,100
04      1 S=S+1/K
05        PRINT 2,S
06      2 FORMAT('␣S=',E15.7)
07        END
```

```
**␣S=␣␣␣1.0000000+00
```

Задача 12. Напишите программу, определяющую количество разрядов, используемых для представления мантиссы чисел с плавающей точкой.

Задача 13. Напишите программу, которая передает управление на 10-е слово ее внутреннего массива ARRAY.

Задача 14. Приведенная программа вычисляет число 2^{50} последовательным умножением. Чем объясняется то, что результат вычисления равен 0?

```
01        PROGRAM TASK
02        N=1
03        DO 1 K=1,50
```

```
04      1 N=N*2
05        PRINT 2,N
06      2 FORMAT('␣N=',I20)
07        END
**␣N=␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣0
```

Задача 15. Объясните, почему при выполнении приведенной программы возникает ошибка «деление на нуль».

```
01        PROGRAM TASK
02        EXTERNAL INTC
03        CALL INSUM(INTC,S)
04        END
      C ---------------------------------------------
05        SUBROUTINE INSUM(FUNCT,S)
06        S=0.
07        DO 1 K=1,10
08      1 S=S+1./FUNCT(K)
09        RETURN
10        END
      C ---------------------------------------------
11        FUNCTION INTC(K)
12        INTC=K**2
13        RETURN
14        END
```

Задача 16. Приведенная ниже программа суммирует массив из 15 малоотличающихся чисел двумя способами. В первом случае сумма вычисляется последовательным суммированием чисел с первого до пятнадцатого. Во втором случае числа разбиваются на четыре группы, вычисляются суммы чисел в каждой группе и затем вычисляется сумма сумм чисел в группах. Объясните причину различия результатов в зависимости от способа суммирования.

```
01        PROGRAM TASK
02        REAL A(15),SS(4)
03        DATA    A/0.5555555,0.5655555,0.556555,0.5556555,
         *          0.5555655,0.5555596,0.5555569,0.5555565,
         *          0.5555956,0.8765956,0.5559565,
         *          0.5557876,0.5553245,0.559998,0.666666/
04        N=4
05        N2=15
06        S=0.
07        DO 1 I=1,N2
08      1 S=S+A(I)
09        PRINT 2,S
10      2 FORMAT('␣S=',E20.11)
11        S=0.
```

```
12        DO 3 I=1,N
13      3 SS(I)=0.
14        NI=N
15        DO 5 I=1,N
16        IF(I .EQ. N) NI=3
17        DO 4 J=1,NI
18        JI=(I-1)*N+J
19      4 SS(I)=SS(I)+A(JI)
20      5 S=S+SS(I)
21        PRINT 2,S
22        END
CC                    НА ЭВМ БЭСМ-6
**␣S=␣␣␣␣␣8.78148379995+00
**␣S=␣␣␣␣␣8.78148379998+00
CC                    НА ЕС ЭВМ
**␣S=␣␣␣␣␣0.8781477E 01
**␣S=␣␣␣␣␣0.8781479E 01
```

Задача 17. Рассмотрите приведенную программу и объясните, почему суммирование элементов массива M с помощью подпрограммы SUBSUM приводит к ошибке.

```
01        PROGRAM TASK
02        INTEGER M(10)
03        DATA M/1,2,3,4,5,6,7,8,9,10/
04        ISUM=0
05        CALL SUBSUM(ISUM,M)
06        PRINT 1,ISUM,M
07      1 FORMAT('␣ISUM=',I3/10I3)
08        END
      C -----------------------------------------
09        SUBROUTINE SUBSUM(ISUM,M)
10        DO 1 K=1,10
11      1 ISUM=ISUM+M(K)
12        RETURN
13        END
```

Задача 18. Какое обращение к приведенной ниже подпрограмме могло вызвать указанную печать?

```
01        SUBROUTINE SUBR(X,Y,Z)
02        PRINT 1,X,Y,Z
03        X=1.
04        PRINT 1,X,Y,Z
05        RETURN
06      1 FORMAT('␣X=',F2.0,'␣Y=',F2.0,'␣Z=',F2.0)
07        END
**␣X=0.␣Y=0.␣Z=0.
**␣X=1.␣Y=1.␣Z=1.
```

Задача 19. Напишите программу, которая заносит нуль в тридцатое относительно входа в программу слово тела некоторой подпрограммы SUBR.

Задача 20. Сколько и какого типа ошибки содержатся в приведенной ниже программе? Каким будет результат работы этой программы?

```
01        PROGRAM PROGRAM
02        INTEGER FORMAT(5)
03        CALL SB=2.
04        ENTRY A=5.
05        DO 1 I3=1,5
06      1 FORMAT(I3)=I3
07      2 FORMAT(I3)
08        PRINT 2,FORMAT
09        CALL SUBROUTINE(CALL SB,ENTRY A)
10        PRINT 3, CALL SB, ENTRY A
11      3 FORMAT(1X,2F4.0)
12        END
    C ----------------------------------------
13        SUBROUTINE SUBROUTINE(RETURN,END)
14        REAL REAL(10), IF(6)
15        REAL X(M,N)=(M+N)*(M-N)
16        REAL (I)=REAL X(3,5)
17        DATA=RETURN+END
18        DO 3 K=1,5
19        DO 3 K=1.5
20        PRINT 1=FLOAT(K)
21        IF(K)=K**2
22        CONTINUE=PRINT 1
23      3 CONTINUE
24        DO IF GO TO WRITE=1 2 3 4 5 6 7 8 9 0
25        STOP=REAL(1)
26        GO TO 3=2.
27        IF(K)=GO TO 3
28        RETURN=END
29        PRINT 1, PRINT 1, IF(K-3), GO TO 3,
30      2 CONTINUE, RETURN, END
31      1 FORMAT(1X,6F4.0)
32        RETURN
33        END
```

Задача 21. В приведенной программе суммируются сто одинаковых чисел A и вычисляется произведение SN=100*A. Сравните вычисленные значения S и SN между собой и с точным значением, равным 69.92896. Объясните полученные результаты.

```
01        PROGRAM TASK
02        DATA A/0.6992896/
03        N=100
04        S=0.
05        DO 1 I=1,N
06      1 S=S+A
07        SN=N*A
08        PRINT 2,S,SN
09      2 FORMAT('␣S=',E16.11,5X,'SN=',E16.11)
10        END
CC              НА ЭВМ БЭСМ-6
**␣S=6.99285999967+01␣␣␣␣␣SN=6.99285999998+01
CC              НА ЕС ЭВМ
**␣S=0.6992819E␣02␣␣␣␣␣SN=0.6992859E␣02
```

Задача 22. Приведенная ниже программа вычисляет сумму всех целых чисел от 1 до 25 двумя способами:

1) непосредственным суммированием всех чисел (оператор 02),

2) последовательным суммированием в цикле (операторы 03—05).

Результаты вычислений оказались различными. Чем объясняется это различие?

```
01        PROGRAM TASK
02        N=1+2+3+4+5+6+7+8+9+10+11+12+...и так до 25
03        M=0
04        DO 1 K=1,25
05      1 M=M+K
06        PRINT 2,N,M
07      2 FORMAT('␣␣␣␣N=',I3,'␣␣␣␣M=',I3)
08        END
**␣␣␣␣N=302␣␣␣␣M=325
```

Задача 23. Объясните, почему при выполнении приведенной программы возникает аварийная ситуация.

```
01          PROGRAM TASK
02          Y=QUEST(X)
03          STOP
04          END
    C ------------------------------------
05          SUBROUTINE QUEST(X)
06          X=0.
07          RETURN
08          END
```

Задача 24. Напишите подпрограмму SUBR, которая при вызове ее из программы TASK, не используя общего блока памяти, пересылает содержимое переменной Z в переменную X.

```
01      PROGRAM TASK
02      COMMON /BLOCK/ X,Y,Z
03      CALL SUBR(Y)
04      STOP
05      END
```

Задача 25. Как закончится выполнение приведенной программы?

```
01      PROGRAM TASK
02      Y=SIGN(X)
03      STOP
04      END
```

Задача 26. Напишите программу, вызывающую подпрограмму SUBR таким образом, чтобы подпрограмма SUBR при этом зациклилась.

```
01      SUBROUTINE SUBR(K,L)
02      DIMENSION L(2)
03      DO 1 K=1,2
04    1 L(K)=0
05      RETURN
06      END
```

Задача 27. Напишите программу, которая заносит число нуль в слово памяти с абсолютным адресом 2000B.

Задача 28. Подпрограмма MULTM выполняет умножение матриц Z=X*Y. Объясните, чем вызвано различие полученных результатов вычисления произведения матриц A и B в операторах 04 и 06.

```
01    PROGRAM TASK
02    INTEGER A(2,2),B(2,2),C(2,2)
03    DATA A/1,2,3,4/,B/5,6,7,8/
04    CALL MULTM(A,B,C)
05    PRINT 2,C
06    CALL MULTM(A,B,A)
07    PRINT 2,A
08  2 FORMAT('␣МАТРИЦА=',4I5)
09    END
   C ----------------------------------------
10    SUBROUTINE MULTM (X,Y,Z)
11    INTEGER X(2,2),Y(2,2),Z(2,2)
12    DO 1 K=1,2
13    DO 1 J=1,2
14    IS=0
15    DO 2 I=1,2
16  2 IS=IS+X(K,I)*Y(I,J)
17  1 Z(K,J)=IS
```

```
18      RETURN
19      END
** ␣МАТРИЦА = ␣␣␣23␣␣␣34␣␣␣31␣␣␣46
** ␣МАТРИЦА = ␣␣␣23␣␣␣34␣␣185␣␣270
```

Задача 29. Каким образом следует вызвать подпрограмму SUBR, чтобы произошла аварийная ситуация типа «деление на нуль» при «не нулевом» значении переменной X?

```
01      SUBROUTINE SUBR(X)
02      IF(X .GE. 1) X=1./X
03      RETURN
04      END
```

Задача 30. В приведенной программе вычисляется значение выражений

$$S=(A+A+A+A+A+A+A+A+A+A)*B \text{ и } SN=10*A*B$$

для значений A=0.4929847, B=0.86422379. Объясните причину различия значений S и SN.

```
01      PROGRAM TASK
02      A=0.4929847
03      B=0.86422379
04      N=10
05      S=0.
06      DO 1 I=1,N
07    1 S=S+A
08      S=S*B
09      SN=N*A*B
10      PRINT 2,S,SN
11    2 FORMAT('␣S=' E16.11,5X,'SN=',E16.11)
12      END
CC              НА ЭВМ БЭСМ-6
**␣S=4.26049105844+00␣␣␣␣␣SN=4.26049105845+00
CC              НА ЕС ЭВМ
**␣S=0.4260484E 01␣␣␣␣␣SN=0.4260489E 01
```

Задача 31. Рассмотрите приведенную ниже подпрограмму, которая печатает содержимое общего блока памяти или обнуляет его содержимое в зависимости от значения формального параметра. Что произойдет, если выполнить следующие операторы:

1. CALL SUBR(0)
2. CALL SUBR(4)
3. CALL SUBR(1.0)
4. CALL SUBR(100)

```
01      SUBROUTINE SUBR(J)
02      COMMON/BLOCK/X,Y
```

```
03        GO TO (1,2,3),J
04      1 PRINT 4,X
05      4 FORMAT(' X=',F10.3)
06        RETURN
07      2 PRINT 5,Y
08      5 FORMAT('␣Y=',F10.3)
09        RETURN
10      3 X=0.
11        Y=0.
12        RETURN
13        END
```

Задача 32. Напишите программу, которая присваивает переменной X в качестве значения содержимое десятого слова памяти относительно входа в подпрограмму.

Задача 33. Подпрограмма PRINTA печатает массив M, начиная с элемента с номером N1 и кончая элементом с номером N2. Шаг печати равен N3. К чему приведет каждое из обращений к подпрограмме PRINTA (операторы 05—09)?

```
01      PROGRAM TASK
02      INTEGER M(10)
03      DO 1 K=1,10
04    1 M(K)=K
05      CALL PRINTA (M,1,7,2)
06      CALL PRINTA (M,8,3,—1)
07      CALL PRINTA (M,9,2,1)
08      CALL PRINTA (M,2,6,0)
09      CALL PRINTA (M,4,10,—2)
10      END
      C ----------------------------------------
11      SUBROUTINE PRINTA (M,N1,N2,N3)
12      INTEGER M(1)
13      DO 1 K=N1,N2,N3
14    1 PRINT 2,M(K)
15    2 FORMAT(I5)
16      RETURN
17      END
```

Задача 34. Стандарт языка фортран запрещает использование рекурсии — ситуации, когда подпрограмма или функция вызывает, прямо или косвенно, сама себя. Тем не менее ряд реализаций фортрана допускает, если не прямо, то с помощью некоторых ухищрений, написание рекурсивных программ. Подумайте, на каких свойствах реализации языка фортран основывается возможность рекурсии. Напишите рекурсивную функцию для вычисления факториала целого числа N. Учтите, что для этого вам нужно «обмануть» транслятор.

Задача 35. Напишите программу, которая вызывает приведенную подпрограмму таким образом, что в операторах 03 и 05 подпрограммы происходит указанная печать.

```
01        SUBROUTINE SUBR(L)
02        COMMON/BLOCK/K
03        PRINT 1,K
04        L=0
05        PRINT 1,K
06        RETURN
07      1 FORMAT('␣K=',I1)
08        END
**␣K=1
**␣K=0
```

Задача 36. Напишите программу, которая передает управление на слово памяти с абсолютным адресом 2000B.

Задача 37. Рассмотрите приведенную ниже программу, в которой вычисляются выражения S1=(A—B)/C и S2=A/C—B/C для следующих значений: A=0.4848488, B=0.4444123 и C=0.8787877. Сравните полученные результаты вычисленных значений S1 и S2 и объясните причину их различия.

```
01        PROGRAM TASK
02        DATA A/0.4848488/,B/0.4444123/,C/0.8787877/
03        S1=(A—B)/C
04        S2=A/C—B/G
05        PRINT 1,S1,S2
06      1 FORMAT('␣S1=',E16.11,5X,'S2=',E16.11)
07        END
CC              НА ЭВМ БЭСМ-6
**␣S1=4.60139576373—02␣␣␣␣␣␣S2=4.60139576371—02
CC              НА ЕС ЭВМ
**␣S1=0.4601396E—01␣␣␣␣␣␣S2=0.4601395E—01
```

Задача 38. Закончится ли нормально приведенная ниже программа?

```
01        PROGRAM TASK
02        CALL QUEST(X)
03        STOP
04        END
  C  ---------------------------------------
05        FUNCTION QUEST(X,Y)
06        QUEST=0.
07        RETURN
08        END
```

Задача 39. Подпрограмма SUBR подсчитывает и печатает число отрицательных элементов массива IA1.

1) измените один символ в подпрограмме SUBR таким образом, чтобы при ее исполнении произошла аварийная ситуация,

2) измените один символ в подпрограмме SUBR так, чтобы транслятор не выдал ошибку, а компоновка программы не смогла завершиться,

3) измените один символ в подпрограмме SUBR так, чтобы результат работы программы был неправильным,

4) измените один символ в программе TASK так, чтобы результат работы программы был неправильным.

```
01        PROGRAM TASK
02        DIMENSION IA1(4)
03        DATA IA1/—1,0,10,—300/
04        CALL SUBR(IA1)
05        STOP
06        END
      C   ----------------------------------------
07        SUBROUTINE SUBR(IA1)
08        DIMENSION IA1(4)
09        J=0
10        DO 1 K=1,4
11        IF(IA1(K) .LT. 0) J=J+1
12      1 CONTINUE
13        PRINT 2,J
14      2 FORMAT('␣J=',I1)
15        RETURN
16        END
```

Задача 40. Рассмотрите приведенную программу и объясните различие в печатаемых результатах.

```
01        PROGRAM TASK
02        COMMON X,Y
03        DOUBLE PRECISION X,Y
04        X=1.
05        Y=2.
06        PRINT 1,X,Y
07      1 FORMAT('␣X=',D25.18/'␣␣␣Y=',D25.18)
08        CALL SUB
09        END
      C   ----------------------------------------
10        SUBROUTINE SUB
11        COMMON X,Y
12        PRINT 1,X,Y
13      1 FORMAT('␣X=',D25.18/'␣␣␣Y=',D25.18)
14        RETURN
```

```
15        END
**␣X=␣1.000000000000000000+000 ␣␣
      Y=␣2.000000000000000000+000
**␣X=␣1.000000000000000000+000 ␣␣
      Y=␣0.000000000000000000+000
```

Задача 41. В приведенной программе вычисляется произведение всех целых чисел от 1 до 5. Чем можно объяснить такой странный результат?

```
01        PROGRAM TASK
02        INTEGER M(5)
03        DATA M/1,2,3,4,5/
04        DO 1 K=1,5
05      1 N=N*M(K)
06        PRINT 2,N
07        FORMAT('␣N=',I5)
08        END
**␣N=␣␣␣␣␣0
```

Задача 42. Что произойдет при выполнении приведенной ниже программы?

```
01          PROGRAM TASK
02          DO 1 K=1,40000
03          CALL SUBR(X)
04        1 CONTINUE
05          STOP
06          END
    C ------------------------------------------
07          SUBROUTINE SUBR(X, Y)
08          RETURN
09          END
```

Задача 43. В программе выполняются вычисления выражений: S1=(B—C)·A и S2=A·B—A·C. Согласно правилам арифметики должно быть равенство значений S1 и S2. Точное значение этих выражений для указанных ниже значений A, B и C равно 1.55804487649E—03. Однако, как видно из примера, результаты вычислений отличаются друг от друга. Объясните полученные результаты. Дайте оценку ошибок вычисленных значений S1 и S2.

```
01        PROGRAM TASK
02        DATA A/0.9643157/, B/0.6392145/, C/0.6375988/
03        S1=(B—C)*A
04        S2=A*B—A*C
05        PRINT 1, S1, S2
06      1 FORMAT('␣S1=',E16.11,5X,'S2=',E16.11)
07        END
```

```
CC              НА ЭВМ БЭСМ-6
**␣S1 = 1.55804487649 — 03␣␣␣␣␣␣S2 = 1.55804487622 — 03
CC              НА ЕС ЭВМ
**␣S1 = 0.1558047E — 02␣␣␣␣␣␣S2 = 0.1558065E — 02
```

Задача 44. Напишите программу, вызывающую подпрограмму SUBR таким образом, чтобы происходило зацикливание в подпрограмме SUBR.

```
01        SUBROUTINE SUBR(L)
02        DIMENSION L(2)
03        COMMON /BLOCK/ K
04        DO 1 K=1,2
05      1 L(K)=0
06        RETURN
07        END
```

Задача 45. Напишите программу, которая в процессе работы передает управление на слово, имеющее абсолютный адрес на 100B больший абсолютного адреса входа некоторой подпрограммы SUBR.

Задача 46. Приведенная программа вычисляет сумму элементов строк матрицы M, в которых первый элемент строки положителен. Объясните, почему результат работы программы отличается от правильного: L=4?

```
01        PROGRAM TASK
02        INTEGER M(2,2)
03        DATA M/—1,1,2,3/
04        L=0
05        DO 1 K=1,2
06        IF(M(K,1)␣.LE.␣0) GO TO 1
07        DO 1 N=1,2
08        L=L+M(K,N)
09      1 CONTINUE
10        PRINT 2,L
11      2 FORMAT('␣L=',I2)
12        STOP
13        END
**␣L=␣5
```

Задача 47. Приведенная далее программа вычисляет выражение A·B/C тремя способами: (A·B)/C, A (B/C), (A/C) B. Какими будут результаты вычислений?

```
01        PROGRAM TASK
02        DATA␣A,B,C/1.E15,␣1.E—15,␣1.E—15/
03        X=A*B/C
04        PRINT␣1,X
05        Y=A*(B/C)
```

```
06        PRINT␣1,Y
07        Z=A/C*B
08        PRINT␣1,X,Y,Z
09      1 FORMAT(3E20.12)
10        END
```

Задача 48. Напишите самовоспроизводящуюся программу, которая создает в памяти свою копию и передает ей управление. Копия, в свою очередь, создает новую копию и передает ей управление и так далее. Для начального запуска процесса «порождения» используйте специальную программу.

Задача 49. Почему при выполнении приведенной программы возникает ошибка?

```
01        PROGRAM TASK
02        CALL QUEST(X, Y)
03        STOP
04        END
      C ---------- ---------
05        FUNCTION QUEST(X,Y)
06        QUEST=0.0
07        RETURN
08        END
```

Задача 50. Напишите программу, которая присваивает переменной X содержимое слова памяти с абсолютным адресом 2000B.

Задача 51. Объясните, почему при выполнении приведенной программы возникает ошибка.

```
01        PROGRAM TASK
02        DIMENSION K(3)
03        DATA K/2,3,4/
04        CALL␣SUBR(K)
05        STOP
06        END
      C ----------
07        SUBROUTINE SUBR(IO
08        INTEGER I0(3)
09        I=0
10        DO 1 J=1,3
11      1 I=I+IO(J)
12        RETURN
13        END
```

Задача 52. Что произойдет при выполнении приведенной ниже программы?

```
01        PROGRAM TASK
02        DO 1 K=1,40000
```

```
03        CALL SUBR(X,Y)
04      1 CONTINUE
05        STOP
06        END
    C ------------------------------------------
07        SUBROUTINE SUBR(X)
08        RETURN
09        END
```

Задача 53. Рассмотрите приведенную программу и объясните, почему использование подпрограммы COPY для копирования значений массивов приводит к ошибке.

```
01        PROGRAM TASK
02        INTEGER A(30),B(20),C(20)
03        EQUIVALENCE (A,B), (C,A(11))
04        DO 1 K=1,20
05      1 B(K)=K
06        CALL COPY(B,C,20)
07        PRINT 2,B,C
08      2 FORMAT('B='/,9I2, 11I3/' C='/,9I2, 11I3)
09        END
    C ------------------------------------------
10        SUBROUTINE COPY(M,N,L)
11        INTEGER M(L),N(L)
12        DO 1 K=1,L
13      1 N(K)=M(K)
14        RETURN
15        END
** B=
**  1 2 3 4 5 6 7 8 9 10  1  2  3  4  5  6  7  8  9  10
** C=
**  1 2 3 4 5 6 7 8 9 10  1  2  3  4  5  6  7  8  9  10
```

Задача 54. Сумму из $(n+1)$ членов геометрической прогрессии можно вычислить тремя способами:

$$S_1 = A_0 + A_0 q + A_0 q^2 + \ldots + A_0 q^n,$$
$$S_2 = A_0 q^n + \ldots + A_0 q^2 + A_0 q + A_0,$$
$$S_n = A_0 (1 - q^{n+1})/(1 - q),$$

т. е. суммируя члены убывающей геометрической прогрессии в прямом и обратном порядке, а также по формуле для суммы членов геометрической прогрессии. Объясните полученное различие результатов.

```
01        PROGRAM TASK
02        DATA N/1000/, Q/0.97/, A0/1.
03        A=A0
```

```
04          S1=A0
05          DO 1 I=1,N
06          A=A*Q
07        1 S1=S1+A
08          NN=N+1
09          QN=Q**NN
10          SN=A0*(1.-QN)/(1.-Q)
11          S2=0.
12          DO 2 I=1,N
13          J=N+1-I
14          SS=A0*Q**J
15        2 S2=S2+SS
16          S2=S2+A0
17          PRINT 3,S1,S2,SN
18        3 FORMAT('␣S1=',E16.11/'␣S2=',
19         *E16.11/'␣SN=',E16.11)
20          END
```

```
CC              НА ЭВМ БЭСМ-6
**␣S1=3.33333333094+01
**␣S2=3.33333333316+01
**␣SN=3.33333333326+01
CC              НА ЕС ЭВМ
**␣S1=0.3333030E+02
**␣S2=0.3333297E+02
**␣SN=0.3333329E+02
```

Задача 55. Напишите подпрограмму SUBR такую, чтобы в результате обращения к ней из приведенной ниже программы (оператор 02) операторы 03—05 печатали бы указанные данные.

```
01          PROGRAM TASK
02          CALL SUBR(4)
03          DO 1 K=1,4
04        1 PRINT 2,K
05        2 FORMAT('␣K=',I1)
06          END
```

```
**␣K=1
**␣K=2
**␣K=3
**␣K=4
**␣K=5
**␣K=6
**␣K=7
**␣K=8
```

Задача 56. Напишите программу, которая в процессе выполнения передает управление на десятое слово общего блока памяти:

```
COMMON /BLOCK/ ARRAY(100)
```

Задача 57. Рассмотрите приведенную ниже программу. Правильно ли будет выполнена эта программа? Если нет, то в чем заключается ошибка?

```
01        PROGRAM TASK
02        DIMENSION A(5)
03        DATA A/0.,1.0,10,15000.0,100./, B/15000/
04        S=0.
05        DO 1 I=1,5
06      1 S=S+A(I)
07        C=S/B
08        PRINT 2,S,C
09      2 FORMAT(1X,'S=',E12.5,10X, 'C=',E12.5)
10        END
```

ГЛАВА 3

ИСТОЧНИКИ ОШИБОК В ПРОГРАММАХ НА ФОРТРАНЕ

Отыщи всему начало, и ты многое поймешь.

Козьма Прутков

При оценке языка фортран необходимо учитывать конкретную историческую обстановку, в которой происходило зарождение и становление этого языка. В отличие от современных языков программирования, язык фортран прошел довольно длинный путь эволюции от первой своей версии, зародившейся в недрах фирмы IBM, до той формы, в какой он пребывает в настоящее время. Надо сказать, что попытки дальнейшего развития языка фортран не прекращаются и в настоящее время. Примером тому может служить фортран-77 [24] и многие другие современные диалекты языка, например форекс [3]. Все это определило некоторую непоследовательность, свойственную языку фортран.

История языка фортран насчитывает уже более трех десятилетий и берет свое начало примерно в 1953—1954 гг., когда Джон Бекус с сотрудниками приступили к разработке системы программирования, использующей специальные языковые средства. Главной целью их работы была разработка системы программирования, позволяющей автоматизированно создавать программы, лишь незначительно (примерно в 1,5—2 раза) уступающие по эффективности программам, написанным вручную квалифицированными программистами в машинных командах. Эффективность получаемой программы и определяла выбор решений при разработке системы программирования. Поэтому сам по себе язык программирования отошел на второй план, а все усилия были направлены на создание эффективной транслирующей программы. Разработчики считали, что для каждой новой ЭВМ должен создаваться свой язык программирования, максимально учитывающий систему команд и организацию памяти ЭВМ. А значит, язык должен прежде всего быть ориентирован на ЭВМ, а не на решаемые с его помощью задачи. Результатом

работы явились язык фортран и транслирующая программа с этого языка для ЭВМ IBM-704, о чем было доложено на конференции, состоявшейся в 1957 г. Язык фортран довольно быстро начал распространяться. И вопреки исходному предположению, стали появляться трансляторы с языка фортран на других ЭВМ, архитектура которых отличалась от архитектуры IBM-704.

Развитие языка фортран довольно хорошо отражает историю программирования и вычислительной техники. Фортран жил, развивался, впитывал новые идеи, расширялись возможности аппаратуры, и это находило отражение в языке. В истории языка фортран можно выделить три этапа, каждый продолжительностью примерно в десять лет. Первый и второй этапы завершились знаменательными событиями. В течение первого этапа (1957—1966 гг.) сформировалась идея языка программирования высокого уровня. Была понята роль языка как инструментального средства разработки программ, позволившего значительно повысить эффективность труда программистов. В языке фортран выделяются два диалекта — фортран IV и бейсик-фортран, которые закрепляются в качестве стандартных.

Второй этап (1967—1977 гг.) характеризуется активным проникновением ЭВМ в различные «невычислительные» области применения. Существенно расширяются аппаратные возможности ЭВМ. Все большее значение преобретает качество и надежность программного обеспечения, сокращение сроков разработки программ, повышение эффективности и удобства инструментальных средств. В центре внимания разработчиков языков программирования находятся идеи абстрактных типов данных и структурного программирования. Все это естественным образом оказывает влияние и на фортран. Итогом этого становится появление языка фортран-77.

В настоящее время мы подошли к концу третьего этапа (1978—1987 гг.) развития языка фортран и естественно задать вопросы: чем будет отмечено это десятилетие? Какие тенденции в развитии языков программирования и вычислительной техники найдут свое отражение в языке фортран и будет ли вообще фортран-88? Трудно ответить что-либо определенное на поставленные вопросы — это требует глубокого анализа состояния дел. Можно лишь отметить, что одним из лозунгов сегодняшнего дня является параллельность вычислений. Уже существуют и проектируются вычислительные системы, содержащие не десятки или сотни, а десятки и сотни тысяч (в перспективе и миллионы) параллельно работающих процессоров. Может быть, именно эта тенденция в развитии вычислительной техники и явится причиной рождения нового языка, тем более, что в этом направлении уже делаются некоторые шаги.

Язык фортран относится к языкам, имеющим слабую защищенность от ошибок. Рассмотрим подробно основные источники возникновения ошибок в программах на фортране.

3.1. Особенности синтаксиса языка фортран и связанные с ними ошибки

Одним из основных средств защиты от ошибок в современных языках программирования является такой выбор синтаксиса конструкций языка, что любые небольшие ошибки в записи могут быть обнаружены транслятором при анализе программы. К сожалению, фортран таким свойством не обладает. Синтаксис языка фортран таков, что иногда замена одного символа другим превращает одну правильную конструкцию языка в другую, также синтаксически верную конструкцию. Таким образом, если такая подстановка возникает случайным образом, например из-за опечатки при вводе текста программы с терминала, то транслятор не может обнаружить эту «ошибку» в силу того, что она является ошибкой с точки зрения программиста (изменяется смысл программы), а для транслятора это совершенно правильно написанная программа. Такая слабая защищенность синтаксиса языка фортран обусловлена следующими причинами.

Визуальное разбиение конструкций (операторов) языка на составные части, выполняемое человеком, не всегда совпадает с разбиением, осуществляемым транслятором на основе синтаксических правил языка. Визуальное разбиение оператора основывается на выделении ключевых слов и разделителей. В качестве разделителей, как правило, используются пробелы, скобки, запятые, точки, знаки операций и т. п. С использованием пробела в качестве разделителя и связаны многие ошибки. При визуальном контроле программы человек использует пробел как один из основных разделителей. Напротив, в языке фортран пробел не является разделителем, что отчасти и приводит к различию визуального и синтаксического разбиений оператора. В программе на фортране все пробелы просто игнорируются. Они исключаются при предварительном просмотре программы. Например, в операторе

```
COMMON A, B C
```

пробелы между B и C игнорируются транслятором. В результате объявляются две переменные A и BC, а не три A, B, C, как это может показаться. Подобная ошибка возникает довольно часто из-за неизбежности всякого рода опечаток. Так, последний оператор мог явиться следствием того, что в операторе

```
COMMON A, B, C
```

вместо второй запятой был случайно сделан пробел.

Аналогичная ошибка может иметь место практически в каждом операторе. Хотя внешне она может в разных ситуациях выглядеть несколько по-иному, идея остается неизменной — в результате замены пробелами (или пропуска) в операторе одного или нескольких символов может получиться синтаксически правильный оператор. Ниже приводится далеко не полный список возможных случаев и примеров к ним (первая строка каждого примера показывает, как должна была быть

напечатана правильная конструкция, вторая — как она напечатана в действительности, и третья — как она «воспринимается» транслятором).

1. Операторы описаний REAL, INTEGER, COMMON, DATA и т. д.

```
REAL ALFA, BETA, GAMA, DELTA,   X, Y
REAL ALFA, BE A, GAMA, DELTA    X, Y
REAL ALFA, BEA, GAMA, DELTAX,   Y
```

2. Вычисляемый оператор перехода.

```
GO TO (1, 243, 7, 6, 5), M
GO TO (1, 2 3, 7 65), M
GO TO (1,  23, 765), M
```

3. Оператор цикла.

```
DO 1 K=1,10,3
DO 1 K=1,10 3
DO 1 K=1,103
```

4. Целые и действительные числа.

```
2.7118
2 7118
27118
```

5. Обращение к элементу массива.

```
X(17,25)
X(␣7,25)
X(7,25)
```

6. Вызов подпрограммы или функции.

```
CALL SUB(A, FUN, X, 3)
CALL SUB(A, F␣N, X 3)
CALL SUB(A, FN, X3)
```

Еще одна возможность для появления ошибки в программе кроется в случайной замене разделителя точки '.' на запятую ',' и наоборот. Целый ряд конструкций языка допускает взаимозамену этих разделителей, при которой не возникает синтаксических ошибок, но изменяется семантика операторов.

Причина возникновения такой ошибки довольно очевидна. Если M и N — две целые десятичные константы (N — константа без знака), то запись

```
M.N
```

обозначает в фортране действительную константу с целой частью M и дробной частью N, Заменив точку запятой, получим

```
M,N
```

что обозначает последовательную запись элементов списка параметров некоторого оператора языка фортран. Если в указанный список могут входить как целые, так и действительные числа, то замена запятой на точку приводит к тому, что вместо двух целых элементов списка возникает один действительный элемент, и наоборот. Наиболее часто такая ситуация возникает в обращении к подпрограммам и функциям, когда список фактических параметров содержит целые и действительные константы. Например, оба обращения к функции FUNC

```
FUNC(X,2,3,N)
FUNC(X,2.3,N)
```

совершенно правильны синтаксически. При этом в первом случае функция FUNC имеет четыре параметра, а во втором — три. Вместо двух целых чисел 2 и 3 во втором случае имеется одно действительное число 2.3. Учитывая, что при трансляции модуля, содержащего обращение к функции FUNC, транслятору недоступна информация о фактическом числе параметров этой функции, то возникшая ошибка не будет зафиксирована. Результат этой ошибки проявится лишь при выполнении программы. При этом определить источник ошибки, как правило, оказывается очень трудно.

Очевидно, что и, наоборот, замена точки на запятую может привести к ошибке, которую нельзя обнаружить при трансляции. Кстати сказать, такая ошибка возникает очень часто и причина ее порой кроется в том, что мы еще со школы привыкли использовать запятую в качестве разделителя целой и дробной части числа, в то время как в языке фортран (да и практически во всех других языках программирования) для этих целей используется точка. Такая двойственность ситуации и приводит к ошибке, когда программист машинально использует в качестве разделителя запятую. Особенно часто это случается с новичками.

В языке фортран имеется целый ряд операторов, использующих списки параметров. В этих операторах рассмотренная ошибка может остаться незамеченной. Ниже приводятся возможные случаи,

1. Оператор DATA.

```
DATA M/1,2,3,4/
DATA M/1,2.3,4/
```

2. Оператор DO.

```
DO 7 K=1,5
DO 7 K=1.5
```

3. Обращение к подпрограмме или функции.

```
CALL SUBP(1,2,3,4)
CALL SUBP(1,2.3,4)
```

4. Оператор WRITE (PRINT, PUNCH).

```
WRITE(41,25) A,5,7
WRITE(41,25) A,5.7
```

Отметим, что не все диалекты языка фортран допускают использование констант в списке вывода.

Еще одна часто встречающаяся ошибка связана с синтаксической неразличимостью обращения к функции и к элементу массива. Например, в операторе

$$X=3.141592*F(N,K,5)+A(3,7)$$

не имея дополнительной информации, нельзя определить, являются ли F и A функциями или массивами. Все зависит от того, имеются ли в модуле описания F и A как массивов в операторах COMMON, DIMENSION, REAL, DOUBLE PRECISION, COMPLEX, INTEGER, LOGICAL или как функций в EXTERNAL. Если программист случайно забыл описать массив, что довольно часто случается на практике, и при этом массив используется только в режиме выбора его элементов, т. е. имя этого массива не встречается слева от знака присваивания, в списке ввода операторов READ или в других подобных операторах, то транслятор будет воспринимать обращение к массиву как вызов функции с тем же именем, что и имя массива.

Хотя в языке фортран есть средства явного определения переменных в операторах REAL, COMMON, DATA и других, имеется возможность неявного определения переменных. Большинство программистов явным образом описывают лишь массивы, а простые переменные, как правило, определяются неявно. Для этого достаточно, чтобы имя переменной встретилось где-то в программе. Например, при наличии в программе любого из перечисленных ниже операторов в модуле будет определена переменная с именем N:

```
DO 1 N=2,10
N=K+3
CALL SUBP(N,A,5)
A=FUNC(N,3,B)
READ 7,N,A,B
```

Такое неявное определение переменных часто приносит больше вреда, чем дает преимуществ. Неизбежно возникающие опечатки при записи программы или вводе с терминала, при пробивке на внешних устройствах иногда остаются незамеченными только в силу действия неявного определения переменных.

Наиболее часто возникает замена литер, схожих по написанию:

```
0—O  (ЦИФРА 0 — БУКВА O)
1—I  (ЦИФРА 1 — БУКВА I)
2—Z  (ЦИФРА 2 — БУКВА Z)
```

3—З (ЦИФРА 3 — БУКВА З)
4—Ч (ЦИФРА 4 — БУКВА Ч)
5—S (ЦИФРА 5 — БУКВА S)
.—, (ТОЧКА . — ЗАПЯТАЯ ,)
'—, (АПОСТРОФ' — ЗАПЯТАЯ ,)
Y—У (ЛАТИНСКАЯ БУКВА Y—РУССКАЯ БУКВА У)

и ряда других схожих по написанию латинских и русских букв, если их представление в ЭВМ различается. Например,

```
CALL SUB(X,I)   CALL SUB(X,1)
READ 2,A,B0     READ 2, A, BO
N3=N+1          N3=N+1
```

В первом случае вместо переменной I второй параметр равен единице. Во втором случае произошла замена цифры нуль на букву О, а в последнем случае — буквы З на цифру три. Ни одну из указанных ошибок транслятор с языка фортран обнаружить не может. Это послужило одной из причин, по которой практически во всех современных языках программирования (например, паскаль и ада) требуется описывать все используемые объекты.

Язык фортран унаследовал от машинных языков (автокод, ассемблер) формат записи предложений. Прежде всего это касается выделения особых полей, которым соответствуют определенные позиции в строке: поле метки (позиции 1—5), поле признака продолжения (позиция 6), поле оператора (позиции 7—72), поле комментария или поле идентификатора оператора (позиции 73—80). Достоинства такого деления каждой строки весьма сомнительны. Одна из распространенных ошибок связана с полем идентификатора. Если оператор оказался настолько длинным, что его окончание попало в поле идентификатора (позиции 73—80), то с большой вероятностью это не будет отмечено транслятором. Например,

```
N ПОЗИЦИИ:

7 8 9  ... 66 67  68 69 70 71 72 73 74 75 76 77 78 79 80
A=B+... — C  /  2  .  + B  A  C  —  1  .  7
```

В этом случае заключительная часть оператора, попадающая в поле идентификатора, а именно

```
AC—1.7
```

будет отсечена от оператора присваивания и транслятор воспримет оператор

```
A=B+. . . . . —C/2. +B
```

Так как переменная В определена (она встречается в этом операторе), то никакой ошибки не будет зафиксировано, С переходом от перфокарт

к терминальному режиму работы описанная ситуация возникает довольно часто, так как в отличие от перфокарт, на которых напечатан номер каждой позиции, при вводе с терминала довольно трудно контролировать номер текущей позиции.

Ряд ошибок связан с полем признака продолжения. Рассмотрим следующий пример:

```
ПОЗИЦИЯ 7
          ↓
          INTEGER K,L,M,A
        REAL I,C,X(5)
        ↑
ПОЗИЦИЯ 6
```

Сдвиг второго оператора на одну позицию влево привел к тому, что второй оператор стал продолжением первого, так как буква R попала в поле признака продолжения. Транслятор воспримет это, как если бы был написан оператор

```
INTEGER K,L,M,AEALI,C,X(5)
```

Обратите внимание на то, что произошло «склеивание» символов A, EAL, I в идентификатор AEALI. Это связано с тем, что пробелы в языке фортран игнорируются. В результате переменная I, которая должна была иметь тип REAL, по умолчанию получает тип INTEGER. Переменная A получает тип REAL вместо INTEGER. Появляется новая переменная AEALI, тоже имеющая тип INTEGER, а переменные C и X получают тип INTEGER, вместо предполагавшегося типа REAL.

Естественно, что ошибки типа опечатки могут произойти в любом месте программы, вызывая те или иные последствия. Мы рассмотрели лишь некоторый класс таких ошибок.

Одним из слабых мест языка фортран является отсутствие в языке зарезервированных символов, так называемых ключевых слов. С учетом сказанного ранее это приводит к тому, что распознать конструкцию можно, лишь проанализировав контекст, в котором она встречается. Ниже приведены примеры таких неоднозначностей. Левый столбец соответствует традиционному использованию «ключевых слов» языка фортран, а правый — одному из возможных вариантов их «фальсификации»:

```
REAL K(5)                 REAL K(5)=12.98*ABC
DO 1 K=1,25,2             DO 1 K=1.252
IF(K) GO TO 777           IF(K)=GO TO 777
READ(41,999) A,B,C        READ(41,999)=A+B+C
GO TO 53                  GO TO 5=3
ENTRY S                   ENTRY S=SUBRUT/FUNCT,
FORMAT(I6)                FORMAT(I6)=E10+3.0
PRINT 12, X,Y,Z           PRINT 12=PI*17.4
RETURN                    RETURN=A*B/C
```

END	END SUB(2,K) = EX — X
STOP	STOP IF(K) = 2.875
WRITE(61,2) I,J,K	WRITE(61,2) = F(I,J,K)
GO TO (1,3,2), K	GO TO (1,3,2) = K
ASSIGN 55 TO M	ASSIGN = 55 — M
COMMON /ABC/ X,Y,Z	COMMON(ABC) = D(X,Y,Z)
DATA NS/3/	DATA NS(3) = 7.546
PUNCH 5, M, N	PUNCH 5 = M*N
CALL SB(K,N)	CALL SB(K,N) = 3.1415

Операторы в каждой из перечисленных пар имеют совпадающую начальную часть. Это может послужить источником ошибки при недостаточно внимательном анализе программы. В качестве упражнения предлагаем читателю придумать совокупность правил, описывающих синтаксис языка фортран.

3.2. Ошибки, связанные с типами данных

Язык фортран зарождался в те времена, когда ясного представления о типах данных в языках программирования еще не существовало. Уместен вопрос: а имеются ли вообще в фортране типы данных? С точки зрения современного представления ответ на этот вопрос будет во многом отрицательным. Иначе как можно расценивать тот факт, что переменная одного типа может принимать значения другого типа и, в принципе, не существует возможности определить тип значения, которым обладает переменная к определенному моменту вычислений.

Формально в языке фортран имеются следующие типы данных, которые могут быть явно или неявно присвоены переменным:

1) логический (LOGICAL),
2) целый (INTEGER),
3) действительный (REAL),
4) действительный с двойной точностью (DOUBLE PRECISION),
5) комплексный (COMPLEX),
6) массивы (DIMENSION и др.).

Кроме того, в языке фортран используются:

7) двоичные, восьмеричные и шестнадцатеричные числа,
8) строки символов,
9) метки,
10) подпрограммы и функции.

Одна из ошибок, относящаяся к типам данных, связана с неявным присваиванием переменной одного типа значения другого типа. Например, присваивание переменной целого типа действительного значения, а переменной действительного типа — значения метки или текста и т. д. Такой эффект достигается различными средствами языка, но идея в каждом случае одна и та же. А именно: для доступа к одному и тому же элементу памяти используются переменные различного типа. Самый

простой способ сделать это — воспользоваться оператором EQUIVALENCE. Например, для операторов

```
LOGICAL L
DOUBLE PRECISION D,P
COMPLEX R
REAL X(10)
EQUIVALENCE (A,K),(X(5),L),(M,D),(R,P)
```

Доступ к пятому элементу массива X может быть осуществлен через переменную L. Последствия от такой эквивалентности во многом зависят от способа представления данных в ЭВМ и от особенностей выполнения машинных команд. Но в любом случае это приводит к ошибке при попытке использовать эти значения так, как, например, в следующих операторах:

```
L=.TRUE.
X(1)=X(2)+X(5)
K=5
X(3)=X(4)/A
M=7
R=D*A
```

Такой же эффект может быть достигнут и другими средствами языка фортран, такими как общие блоки памяти, параметры подпрограмм и функций. Это будет рассмотрено далее в разделе 3.4.

Ряд ошибок при программировании на фортране связан с метками. Существуют следующие причины возникновения таких ошибок: во-первых, синтаксически метки неотличимы от целых чисел, во-вторых, в языке фортран отсутствует возможность явного определения переменной типа МЕТКА, хотя имеется оператор ASSIGN, присваивающий переменной значение, являющееся меткой. Например, оператор

```
ASSIGN 25 TO M
```

присваивает переменной целого типа метку 25. Это значение не эквивалентно целому числу 25, поэтому использование переменной M в арифметических выражениях приводит к ошибке, как, например, в операторе

```
M=M+1
```

Кроме того, значение целого типа не может быть использовано в качестве метки, например, следующим образом:

```
M=38
GO TO M
```

Это также приводит к ошибке.

Необходимо отметить еще один факт. В языке фортран существуют два типа меток — метки, используемые для перехода, и метки для ссыл-

ки на оператор FORMAT. Номера этих меток в некоторых диалектах языка фортран (например, в фортране-Дубна) могут совпадать, что может привести к дополнительным трудностям. В качестве иллюстрации рассмотрим следующий фрагмент программы:

```
      INTEGER M(5)
      DO 1 K=1,5
      M(K)=K
      PRINT 1, K, M(K)
    1 FORMAT ('␣␣K=',I1,' M(K)=',I1)
      IF(K␣.GT.␣5) GO TO 2
      PRINT 2,K
    2 FORMAT (' ЗНАЧЕНИЕ К ПОСЛЕ ВЫХОДА ИЗ ЦИКЛА'
     *          'ПРЕВЫШАЕТ 5:K=',I1)
```

В приведенном фрагменте программы метки операторов FORMAT используются в одном случае как метка оператора, завершающего цикл (метка 1), в другом случае как метка перехода для оператора GO TO (метка 2). Трансляторами с языка фортран на БЭСМ-6 и на ЕС ЭВМ во втором случае будет зафиксирована ошибка трансляции. Такое положение меток операторов FORMAT легко объяснимо, если учесть тот факт, что оператор FORMAT является неисполняемым оператором, но для него генерируется некоторый код, который должен «обходиться» при выполнении программы.

Строки символов, двоичные, восьмеричные и шестнадцатеричные числа находятся в положении, аналогичном положению меток. Например, можно записать присваивания:

```
N=5HТЕКСТ
M='ТЕКСТ'
K='26'
L=1375B
```

но какие-либо операции, кроме сравнения на тождественность, над значениями K, M, N выполнять нельзя. В частности, значение переменной K не эквивалентно целому числу 26.

Довольно много ошибок возникает из-за неправильного использования массивов. Хотя при описании массивов указывается максимально допустимое значение индекса по каждой из размерностей массива, фактический контроль за выходом значений индексов за допустимые границы в процессе исполнения программы осуществляется крайне редко из-за накладных расходов, связанных с таким контролем. Например, ничто не мешает написать программу, содержащую операторы, подобные следующим:

```
REAL X(10)
M=-3
```

```
N=17
X(M)=X(N)
```

Подобные ошибки часто приводят к затиранию программы. Так, в указанном примере будет испорчена часть памяти, в которой записаны данные (или команды), находящиеся в третьем от начала массива X слове памяти, если считать в сторону уменьшения адресов (здесь предполагается, что один элемент массива занимает одно слово памяти).

3.3. Ошибки в использовании операторов управления

Синтаксис управляющих операторов, рассматриваемых в этом разделе, представлен с учетом возможных расширений по сравнению со стандартом языка фортран, как, например, в [3, 24]. Реализация операторов управления в трансляторе фортран-Дубна на БЭСМ-6 приведена в приложении 1.

3.3.1. Вычисляемый оператор перехода. Оператор имеет вид

GO TO (M1, M2, ..., MN), AB

где M1, M2, ..., MN — список меток, на которые возможна передача управления (N > 1), AB — арифметическое выражение. Если значение выражения AB находится в диапазоне $1 \leqslant AB \leqslant N$, то оператор передает управление в соответствии с подстановкой

значение AB	1	2	...	N
метка перехода	M1	M2	...	MN

В соответствии с описанием языка фортран, значение выражения AB должно удовлетворять условию $1 \leqslant AB \leqslant N$. Выполнение оператора в случае $AB < 1$ или $AB > N$ в описании языка не оговорено и полностью зависит от реализации транслятора. В большинстве трансляторов предполагается, что программист сам должен обеспечивать контроль за выполнением этого условия. Поэтому транслятор не вставляет в программу команды, проверяющие принадлежность значения AB указанному диапазону. Программист, как правило, уверен в правильности своей программы и, желая избежать дополнительных накладных расходов на проверке, также не делает никаких проверок. Если, однако, оказывается, что $AB < 1$ или $AB > N$, то возникшая ошибка останется незамеченной, а результат выполнения оператора вычисляемого перехода будет непредсказуем. В большинстве случаев при этом происходит потеря управления.

Если программа содержит команды, контролирующие значение выражения AB, то при нарушении условия $1 \leqslant AB \leqslant N$, в зависимости от реализации транслятора, возможны, например, следующие действия:

— передача управления на метку M1,

— передача управления на метку MN,

— передача управления на оператор, следующий за оператором GO TO,

— передача управления на метку MK, где K — увеличенный на 1 остаток от деления AB на N,

— печать сообщения об ошибке.

Эти действия реализуются за счет дополнительной проверки при выполнении программы пользователя.

Чтобы избежать ошибок, возникающих при нарушении условия $1 \leqslant AB \leqslant N$, следует всегда перед использованием оператора вычисляемого перехода выполнять соответствующую проверку. Это может быть сделано, например, так:

```
      K=AB
      IF((K .GE. 1) .AND. (K .LE. N))
     *          GO TO (M1, M2, ..., MN), K
      операторы, выполняемые, если K < 1 или K > N
```

или как в следующем фрагменте программы:

```
      K=AB
      IF(K .LT. 1) GO TO 1
      IF(K .GT. N) GO TO 2
      GO TO (M1, M2, ..., MN), K
      ...
    1 операторы, выполняемые, если K < 1
      ...
    2 операторы, выполняемые, если K > N
```

Еще об одном типе ошибок (случайный пропуск запятой в списке меток), возможных при использовании вычисляемого оператора перехода, уже говорилось в п. 3.1.

3.3.2. Оператор перехода по предписанию. Оператор имеет вид

```
GO TO M, (M1, M2,..., MN)
```

где M — переменная целого типа, которой с помощью оператора ASSIGN была присвоена одна из меток перехода M1, M2,..., MN. Список меток в скобках в некоторых диалектах языка можно опускать, так как эта информация необходима лишь для контроля правильности значения, присвоенного переменной M, и используется не всеми трансляторами.

Наиболее типичная ошибка, связанная с оператором перехода по предписанию, возникает в случае, когда переменной M не присвоено никакого значения. Например, при выполнении следующего фрагмента программы со значением K=0:

```
IF(K .LT. 0) ASSIGN 1 TO M
IF(K .GT. 0) ASSIGN 2 TO M
GO TO M, (1,2)
```

Переменной M не будет присвоено никакого значения. Выполнение оператора перехода по предписанию в такой ситуации приводит к непредсказуемым результатам.

Нередко начинающие программисты используют вместо оператора ASSIGN оператор арифметического присваивания «равно» (=), как, например,

```
M=3
...
GO TO M, (7, 3, 6, 11)
```

Эта ошибка, как указывалось ранее, частично обусловлена синтаксической неразличимостью меток и положительных целых чисел. Несмотря на такое чисто внешнее сходство, внутреннее представление целых чисел и меток различно. Поэтому в ситуации, показанной в последнем примере, при выполнении оператора перехода по предписанию возникнет ошибка (если только транслятор не вставил в программу команды проверки значения переменной M), приводящая к непредсказуемому результату. Наиболее часто в этом случае происходит потеря управления.

Иногда оператор перехода по предписанию используется для перехода из одного модуля в другой. Например, как в приведенной ниже программе:

```
      PROGRAM A
      READ(41) K
      ASSIGN 1 TO M
      CALL SUBP(K, M)
      PRINT 2,K
    1 PRINT 3
    2 FORMAT ('␣K=',I3)
    3 FORMAT('␣КОНЕЦ')
      STOP
      END
C-----------------------------------
      SUBROUTINE SUBP(N, L)
      IF(N) 1,2,3
    1 ASSIGN 4 TO L
      GO TO 2
    3 ASSIGN 5 TO L
    2 GO TO L,(1,4,5)
    4 N=-1
      RETURN
    5 N=1
      RETURN
      END
```

Если при выполнении программы окажется, что K=0, то оператор перехода по предписанию в подпрограмме выполнит переход на оператор

с меткой 1 в главном модуле. Последствия этой ошибки непредсказуемы. Эта ошибка может быть обнаружена при выполнении программы, но, как уже говорилось выше, большинство трансляторов не вставляют в программу команды, проверяющие принадлежность значения переменной списку меток в операторе перехода по предписанию.

3.3.3. Оператор условного перехода. В языке фортран имеются два вида оператора условного перехода IF: арифметический и логический. Арифметический оператор IF имеет в некоторых диалектах две конструкции:

```
IF (АРИФМЕТИЧЕСКОЕ ВЫРАЖЕНИЕ) М1, М2, М3
IF (АРИФМЕТИЧЕСКОЕ ВЫРАЖЕНИЕ) М1, М2
```

вторая конструкция эквивалентна первой с записью

```
IF (АРИФМЕТИЧЕСКОЕ ВЫРАЖЕНИЕ) М1, М2, М2
```

Какие же ошибки могут быть сделаны при употреблении оператора IF?

Первый тип ошибок связан с опечатками — в арифметическом операторе IF опускается одна из запятых. Как уже упоминалось в п. 3.1, пробел не является разделителем, метки «склеиваются» и, в случае наличия в программе получившейся метки (случайного совпадения), получается синтаксически правильная конструкция. Например, в следующем фрагменте программы:

```
    IF( X ) 1,1 2
  1 X=— X
  2 CONTINUE
    .....
 12 .....
```

Вместо перехода на метку 1 при $X=0$ и на метку 2 при $X>0$ произойдет переход на метку 12 при $X\geqslant 0$.

Второй тип ошибок связан с использованием чисел с плавающей точкой. При сравнении чисел с плавающей точкой необходимо учитывать как ошибки представления действительных чисел в ЭВМ, так и ошибки вычислений (см. п. 3.5). Наличие этих ошибок может оказать влияние на результат сравнения чисел, если эти числа очень близки по величине друг к другу. По этой причине не имеет смысла сравнивать в программе на равенство числа с плавающей точкой, как, например, это сделано в операторе

```
IF(A.EQ.B)GOTO100
```

так как в этом случае передача управления на метку 100 с большой вероятностью выполнена не будет. Правильной будет запись

```
IF(ABS(A—B).LE.EPS)GOTO 100
```

где EPS — допустимая точность несовпадения чисел A и B.

3.3.4. **Оператор цикла.** Оператор цикла занимает особое место среди операторов языка фортран. Это единственный оператор, допускающий использование «внутри себя» других операторов, что и является источником многих ошибок. Оператор цикла состоит из заголовка, тела и завершающего оператора:

```
    DO 1 K=N,M,L
    ⟨тело цикла⟩
1   ⟨завершающий оператор⟩
```

где K — простая переменная, а N,M,L — целые константы или простые переменные. Рассмотрим типичные ошибки, связанные с оператором цикла.

1. Неверно заданы параметры цикла: начальное значение переменной цикла, конечное значение переменной цикла, шаг изменения переменной цикла, как например, в следующем фрагменте программы:

```
    N=1
    M=10
    L=-1
    DO 1 K=N,M,L
1   A(K)=0.
```

В большинстве случаев эти ошибки приводят к затиранию используемой программой области памяти и зацикливанию.

Рассмотрим пример:

```
    N=1
    M=10
    L=0
    DO 1 K=N,M,L
1   A(K)=0.
```

В этом случае происходит зацикливание — бесконечное выполнение цикла с K=1.

Еще один пример:

```
    N=10
    M=1
    L=1
    DO 1 K=N,M,L
1   A(K)=0.
```

В этом примере цикл либо не будет выполнен ни разу, либо будет выполнен один раз, в зависимости от реализации оператора цикла. Более подробно об этом сказано ниже.

2. Некорректная работа с переменной и параметрами цикла. Большинство диалектов языка фортран запрещают изменять переменную и параметры в теле цикла, осуществляя соответствующий контроль при

трансляции программы. Но, несмотря на такие запрещения, имеются способы обойти эти ограничения. Например, с помощью оператора

```
EQUIVALENCE (K,K1), (N,N1), (M,M1), (L,L1)
```

заводятся переменные-синонимы K1, N1, M1, L1 для переменной цикла и его параметров. После этого можно сделать присваивание значений переменным K1, N1, M1, L1 в теле цикла, изменяя тем самым и переменные K, N, M, L. Аналогичный эффект можно получить, если переменные K, N, M, L являются элементами общих блоков памяти или формальными параметрами подпрограммы.

Помимо такого неявного способа изменения переменной и параметров цикла, можно воспользоваться и явным способом. Это достигается, если в процессе выполнения цикла сделать выход из его тела в «окружающую» его часть программы, минуя завершающий оператор, и вернуться затем снова в тело цикла. Например, как в приведенном ниже фрагменте программы:

```
      DO 1 K=N,M,L
      ...
      GO TO 2
      ...
3     CONTINUE
      ...
1     CONTINUE
      ...
2     K=K-1
      N=0
      M=M/2
      L=2
      GO TO 3
```

Приведенный пример показывает, как, используя оператор перехода, можно «расширить» тело цикла, включив в него произвольные части программы. Изменения переменной и параметров цикла, совершаемые вне тела цикла, никак не контролируются.

Каким бы способом ни происходило изменение переменной и параметров цикла, это всегда может привести к ошибкам, аналогичным перечисленным в п. 1.

3. Использование переменной цикла после его завершения. В соответствии с описанием языка фортран, переменная цикла считается неопределенной после окончания цикла (в языке фортран-77 оговаривается значение переменной при выходе из цикла). Тем не менее в большинстве реализаций трансляторов языка фортран переменная цикла имеет вполне определенное значение после его окончания. Это значение может быть равно последнему значению переменной цикла, с которым выполнялось тело цикла, или превышать его на величину шага цикла

(как в трансляторе фортран-Дубна на БЭСМ-6). Использование этого значения может привести к появлению ошибок в программе при переходе к другому диалекту языка.

4. Вход в тело цикла, минуя заголовок. Примером такой ошибки может служить следующий фрагмент программы:

```
      ...
      GO TO 2
      ...
      DO 1 K=N,M,L
      ...
    2 CONTINUE
      ...
    1 CONTINUE
```

Результат выполнения этого фрагмента программы будет непредсказуем, так как многое зависит от того, какие установочные операции производятся в начале цикла. Но в большинстве случаев эффект определяется значением переменной K, которое она имеет при «входе» в цикл.

Реализация оператора цикла в разных трансляторах настолько разнообразна, что трудно даже перечислить имеющиеся варианты [3, 13, 24]. Главное, на что необходимо обратить внимание,— это различия в выполнении этого оператора. При трансляции программы разными трансляторами количество повторений тела цикла может отличаться на единицу. Это объясняется различием в используемом тесте на завершение цикла. Возможны, например, следующие варианты:

1) проверка на окончание цикла производится перед выполнением тела цикла,

2) проверка на окончание цикла производится после выполнения тела цикла,

3) изменение параметра цикла происходит перед проверкой на окончание цикла,

4) изменение параметра цикла происходит после проверки на окончание цикла.

Рассмотрим на примере, как выбор того или иного варианта реализации оператора цикла может повлиять на количество повторений тела цикла:

```
      DO 1 K=1,4,2
      ⟨тело цикла⟩
    1 CONTINUE
```

Реализация этого цикла по вариантам 2,3 эквивалентна выполнению следующих операторов:

```
      K=1
99999 CONTINUE
```

```
      <тело цикла>
   1  K=K+2
      IF(K .LE. 4) GO TO 99999
```

В этом случае тело цикла будет выполнено два раза. Значение параметра цикла после его окончания будет равно 5.

Реализация этого же цикла по вариантам 2,4 эквивалентна выполнению следующих операторов:

```
      K=1
99999 CONTINUE
      <тело цикла>
   1  IF (K .GE. 4) GO TO 88888
      K=K+2
      GO TO 99999
88888 CONTINUE
```

При такой реализации тело цикла будет выполнено три раза. Значение параметра цикла после его окончания будет равно 5.

При реализации цикла по варианту 2 тело цикла выполняется по крайней мере один раз, независимо от заданных параметров цикла. Например, цикл

```
      DO 4 K=5,2
   4  A (K)=7.
```

будет выполнен в этом случае один раз. При реализации по варианту 1 цикл не будет выполнен ни разу.

Таким образом, незнание особенностей реализации транслятором оператора цикла может привести к ошибкам при его использовании в программе.

3.4 Ошибки, связанные с модульной организацией программ

Принцип модульной организации программ в языке фортран является важным, удобным и весьма полезным средством. Принцип модульности позволяет разрабатывать большие программы по частям и обеспечивает возможность использования различных библиотечных модулей, созданных другими программистами. Но, как говорится, за все нужно платить. И программист расплачивается за возможность модульной организации программ большим числом сложных ошибок, возникающих из-за несогласованности интерфейсов отдельных модулей.

В некоторых современных системах программирования транслятор передает компоновщику дополнительную информацию о способе использования каждого внешнего имени, а компоновщик проверяет правильность использования этих имен (переменной, подпрограммы, функции).

Такая проверка позволяет выявлять ошибки, связанные с раздельным определением и использованием имен.

Рассмотрим подробнее организацию интерфейса в программных модулях, написанных на языке фортран.

В языке фортран используется память двух типов:

— локальная память модулей,

— глобальная память общих блоков.

Каждый модуль имеет доступ к своей локальной памяти и к глобальной памяти общих блоков, определенных в данном модуле. Кроме того, каждый модуль, посредством формальных параметров модуля, имеет доступ к локальной памяти вызвавших его модулей и к глобальной памяти, определенной в этих модулях.

3.4.1. Использование общих блоков памяти. Доступ к элементам общего блока памяти организуется на основе описания структуры этого блока, имеющегося в модуле. Общий блок памяти — это память, которая служит для обмена данными между модулями, имеющими доступ к этому блоку памяти. Выделение памяти для общего блока происходит при загрузке программы.

На этапе компоновки и загрузки программы общий блок памяти характеризуется своим именем и размером памяти, отводимой для него. В процессе выполнения программы доступ к общему блоку памяти осуществляется по адресу начала блока. Какова структура общего блока памяти (порядок следования переменных, входящих в блок, их тип и размерность), определяется в модулях, использующих этот блок памяти. При этом разные модули могут иметь различные, несогласованные описания структуры одного и того же общего блока памяти. Это возможно по той причине, что трансляция модулей производится раздельно, а компоновщик проверяет лишь совпадение длин общего блока памяти, даваемых различными описаниями, хотя далеко не все реализации компоновщиков делают даже эту проверку. Такая неоднозначность описания одного и того же объекта является источником многих ошибок.

Рассмотрим пример. Ниже приведены описания одного и того же общего блока памяти в различных модулях и соответствующая этим описаниям структура блока памяти. В примере предполагается, что переменные типа INTEGER, REAL и LOGICAL занимают одно слово, а переменные типа COMPLEX и DOUBLE PRECISION — два слова памяти.

```
Модуль M1: COMMON/BLOCK/ X(10)
Модуль M2: COMMON/BLOCK/ A,B,C(3),D(5)
Модуль M3: COMMON/BLOCK/ P(4),N,F,K(3),S
Модуль M4: COMMON/BLOCK/ E,J(2),R,L,T
           COMPLEX R(2)
           DOUBLE PRECISION T
           LOGICAL L
```

```
M1: X(1) X(2) X(3) X(4) X(5) X(6) X(7) X(8) X(9) X(10)
M2: A    B    C(1) C(2) C(3) D(1) D(2) D(3) D(4) D(5)
M3: P(1) P(2) P(3) P(4) N    F    K(1) K(2) K(3) S
M4: E    J(1) J(2)      R(1)      R(2)      L         T
```

Несмотря на то что во всех случаях длина общего блока памяти, определяемая описаниями, одинакова, его структура оказывается различной. При определении соответствия между переменными общего блока и элементами памяти (словами), используется позиционный принцип:

> если в списке описания общего блока размер памяти, отведенной для всех переменных, предшествующих текущей переменной, равен N, то текущей переменной будут соответствовать слова блока с номерами N+1, N+2, ..., N+M (M — размер памяти, отводимой для этой переменной).

Как видно из приведенного примера, на одни и те же слова памяти общего блока памяти в разных модулях отображаются различные переменные. При этом различается даже тип переменных. Так, восьмому слову блока соответствуют:

— элемент массива действительных чисел X (8) в модуле M1,
— элемент массива действительных чисел D (3) в модуле M2,
— элемент массива целых чисел K (2) в модуле M3,
— логическая переменная L в модуле M4.

Таким образом, при использовании общих блоков памяти может возникнуть эффект, аналогичный эффекту от некорректного использования оператора EQUIVALENCE (см. п. 3.2). Только в данном случае это связано с глобальной памятью общих блоков памяти, а не с локальной памятью модуля.

В отличие от общих блоков памяти, обеспечивающих статический доступ к одним и тем же элементам памяти, формальные параметры модуля позволяют динамически в процессе выполнения программы получать доступ к различным элементам памяти, принадлежащим либо общим блокам памяти, либо локальной памяти вызывающих модулей.

3.4.2. Вызов подпрограмм и передача параметров. Оператор вызова подпрограммы имеет вид

```
CALL ПП
```

при отсутствии параметров, или

```
CALL ПП(П1, . . .,ПN)
```

при передаче N параметров.

Оператор CALL осуществляет передачу параметров и переход на подпрограмму с именем ПП.

В современных языках программирования используются различные способы передачи параметров между модулями: по ссылке, по зна-

чению, по имени и ряд других [14]. Основным способом передачи параметров в языке фортран является передача по ссылке, хотя, в зависимости от реализации, допускаются и некоторые другие способы передачи параметров.

Как уже говорилось в разделе 3.2, каждому объекту — переменной, константе, массиву, подпрограмме, функции — соответствует адрес этого объекта в памяти. Именно этот адрес и передается в качестве фактического параметра при вызове модуля (подпрограммы или функции). В большинстве трансляторов при передаче параметров формируется таблица адресов или значений фактических параметров. Эта таблица располагается либо непосредственно после оператора обращения к подпрограмме, либо в секции рабочих переменных и констант данного модуля, либо в стековой памяти [14, 31, 39]. В таблице адресов, как правило, отсутствует информация о типах фактических параметров, их размерности, количестве и способе передачи параметров. Ниже представлены возможные варианты таких таблиц:

1) ЧИСЛО ПАРАМЕТРОВ	2) ЧИСЛО ПАРАМЕТРОВ	3) АДРЕС П1
АДРЕС П1	АДРЕС П1	АДРЕС П2
РАЗМЕРНОСТЬ П1	АДРЕС П2	. . .
ТИП П1	. . .	АДРЕС ПN
ПРИЗН П1	АДРЕС ПN	
АДРЕС П2		
РАЗМ П2		
ТИП П2		
ПРИЗН П2		
. . .		
АДРЕС ПN		
РАЗМ ПN		
ТИП ПN		
ПРИЗН ПN		

В первом случае в таблице содержится полная информация о передаваемых параметрах. Во втором и третьем случаях имеется лишь частичная информация. ПРИЗН указывает способ передачи соответствующего параметра.

При входе в подпрограмму могут выполняться проверки на соответствие числа параметров, размерности массивов, типа и способа передачи значений переменных.

При установлении соответствия между фактическими и формальными параметрами используется позиционный принцип:

> первый фактический параметр соответствует первому формальному параметру, второй фактический параметр соответствует второму формальному параметру и т. д.

Например, если заголовок программы имеет вид

```
SUBROUTINE EX(A,N,B,K,L,S)
```

то при вызове этой подпрограммы оператором

```
CALL EX(Y,M,3.1415,12,'TEXT',P+T)
```

будет установлено следующее соответствие:

Формальный параметр	Фактический параметр
A	Y
N	M
B	3.1415
K	12
L	'TEXT'
S	*результат выполнения* P+T

При вызове подпрограммы/функции на время выполнения подпрограммы/функции происходит установление эквивалентности между фактическими и формальными параметрами. При этом количество фактических и формальных параметров должно совпадать, иначе в процессе выполнения программы возникнет ошибка, характер которой зависит от реализации механизма передачи параметров (об этом см. далее).

Теперь рассмотрим вопрос согласования типов фактических и формальных параметров. Учитывая, что при передаче фактический параметр представляется, как правило, лишь своим адресом, то программист должен следить за соответствием типов фактического и формального параметров. Это условие приводит к возникновению большого количества ошибок, когда типы параметров не согласуются, как это может иметь место при использовании оператора EQUIVALENCE.

Параметром подпрограммы/функции могут быть объекты следующих классов:

1) константы любого типа, допустимого в языке (целые, действительные, логические, текстовые и т. д.),
2) переменные любого типа,
3) массивы элементов любого типа,
4) подпрограммы,
5) функции,
6) метки (только для подпрограмм).

Фактический и формальный параметры могут не только иметь различный тип, но и принадлежать различным классам, перечисленным выше. Так, если имеется подпрограмма

```
SUBROUTINE SB(A,B,C,D,L,K,SUBP,FUN,M,N)
REAL A,B(5)
COMPLEX C
DOUBLE PRECISION D
LOGICAL L
INTEGER K,M(N)
. . . . . .
```

```
CALL SUBP(B,P)
. . . . . .
A=FUN(K)
. . . . . .
END
```

то ничто не мешает обратиться к этой подпрограмме, например, следующим образом:

```
EXTERNAL S,F
INTEGER F,J(6),C
REAL X,Y(3)
COMPLEX P
. . . . . .
CALL SB(C,Y,3,P,X—1.,F,J,S,1.)
. . . . . .
```

Возникшие при этом ошибки будут зафиксированы не в момент передачи параметров, а лишь при выполнении других операторов подпрограммы SB.

Для сохранения адреса возврата (так называют адрес, куда необходимо передать управление после выполнения подпрограммы), как правило, используются специальные аппаратные средства, имеющиеся практически в каждой ЭВМ, например:

— адрес возврата сохраняется в стеке,

— адрес возврата сохраняется в указанном регистре,

— адрес возврата сохраняется по адресу ПП, а управление передается по адресу ПП+1 и т. д.

Если при вызове подпрограммы отсутствует контроль за интерфейсом, то возможны следующие ошибки:

1. Неверное число параметров. Если количество фактических и формальных параметров не совпадает, то может иметь место:
   — затирание памяти при использовании несуществующих параметров,
   — потеря управления программой, если несуществующий параметр используется как метка или как функция,
   — арифметические ошибки при несоответствии типа несуществующей переменной и содержимого памяти,
   — если передача параметров производится через стек, то при возврате из подпрограммы произойдет сдвиг стека и результаты дальнейшего выполнения программы непредсказуемы.
2. Несовпадение способа передачи фактических параметров. Это может привести к ошибкам при выполнении арифметических операций и к затиранию памяти.
3. Несовпадение типов фактического и формального параметров. Эта ошибка может повлиять на результаты арифметических операций и вызвать затирание памяти.
4. Классы фактического и формального параметров не совпадают,

Наиболее часто это приводит к потере управления программой.

3.4.3. Вызов функций. В языке фортран, как и в большинстве других языков программирования, отсутствует специальный оператор для вызова функций. Обращение к функциям является составной частью арифметического выражения. Синтаксически вызов функции выглядит как имя функции, за которым в скобках следует список фактических параметров.

Переход к выполнению тела модуля функции и передача фактических параметров осуществляется точно так же, как и в операторе вызова подпрограмм. Основное различие реализаций трансляторов в организации вызова функций заключается в способе передачи в вызывающую программу значения функции.

Возможные способы передачи значения функции определяются аппаратными возможностями ЭВМ. Перечислим некоторые способы передачи значения функции:

— значение функции находится на сумматоре или в фиксированной группе регистров центрального процессора,

— значение функции находится в слове памяти, определяемом идентификатором с именем функции,

— значение функции находится в фиксированной области памяти, одинаковой для всех функций,

— значение функции находится в стеке.

При обращении к функции могут произойти те же самые ошибки, что и при вызове подпрограммы (см. п. 3.4.2). Кроме этих ошибок добавляются следующие ошибки, присущие неверному использованию функций:

1. Несовпадение типа вызывающей и вызываемой функции, приводящее к неверному результату вычислений, затиранию памяти и другим «фатальным» результатам.

2. Побочный эффект. Во многих описаниях языка фортран указано, что фактические параметры не могут быть изменены в теле функции (как это принято в математических формулах). Тем не менее большинство реализаций трансляторов допускают это изменение. Возникающий при этом побочный эффект может привести к непредсказуемым результатам.

3. Функция вызывается как подпрограмма или подпрограмма вызывается как функция. Если при вызове функции ее значение передается не через стек, то в этом случае возможны самые различные последствия. При использовании стека для сохранения результата вычисления значения функции в этом случае возникает ошибка из-за дефицита размера стека — стек либо увеличивается, либо уменьшается при обращении к модулю. Это приводит к неправильной передаче управления на более поздних этапах выполнения программы. При организации таблицы параметров в стеке и передаче результата вычисления с помощью стека возможна взаимная компенсация ошибки несоответствия

количества параметров и способа обращения к функции как к подпрограмме. Это может привести к затиранию памяти и другим фатальным ошибкам.

В качестве иллюстрации сказанного в п. 3 разберем следующий пример (слева программа с ошибкой, справа правильная программа):

```
01      PROGRAM TASK
02      CALL SUBP
03      STOP
04      END
     C  ------------------------------------
05      SUBROUTINE SUBP
06      CALL F(1.0)
07      RETURN
08      END
     C  ------------------------------------
09      FUNCTION F(X)
10      RETURN
11      END
```

```
01      PROGRAM TASK
02      CALL SUBP
03      STOP
04      END
     C  ------------------------------------
05      SUBROUTINE SUBP
06      I=F(1.0)
07      RETURN
08      END
     C  ------------------------------------
09      FUNCTION F(X)
10      RETURN
11      END
```

В операторе 06 вызывается подпрограмма F, а в операторе 09 F определена как функция. Предположим, что для передачи параметров и значений функции используется стек. Рассмотрим последовательность выполнения операторов в данном примере.

Текущий выполняемый оператор	Состояние стека после выполняемого оператора в неправильной программе	Состояние стека после выполняемого оператора в правильной программе
оператор 02	адрес оператора 03	адрес оператора 03
оператор 06	. . . . .	. . . . .
	адрес оператора 03	адрес оператора 03
	адрес факт. пар. 1	адрес факт. пар. 1
	адрес оператора 07	адрес оператора 06
оператор 09	адрес оператора 03	адрес оператора 03
	адрес оператора 07	адрес оператора 06
оператор 10	адрес оператора 03	адрес оператора 03
	значение функции F	значение функции F
оператор 06	адрес оператора 03	адрес оператора 03
	значение функции F	
оператор 07	адрес оператора 03 (значение функции F интерпретируется как адрес возврата, результат непредсказуем — программа теряет управление)	нормальное окончание задачи

Некоторые трансляторы к стандартным функциям обращаются нестандартно. Если имя функции пользователя совпадает со стандартным, то при этом возможна ошибка. Это следует учитывать при создании собственных стандартных функций.

3.5. Ошибки, связанные с особенностью выполнения арифметических операций и представлением чисел в ЭВМ

Первоначально вычислительные машины разрабатывались и использовались исключительно для автоматизации вычислительных работ в узком смысле этого слова, т. е. для выполнения операций над действительными и, как частный случай, над целыми числами с использованием четырех арифметических действий — сложения, вычитания, умножения и деления. Такая направленность вычислительных машин нашла свое отражение и в их названии — электронно-вычислительные машины (ЭВМ) (английское название COMPUTER буквально можно перевести как «вычислитель»). С тех пор содержание термина «вычисление» значительно расширилось и стало означать выполнение некоторой совокупности операций, не обязательно арифметических, над объектами, которые не обязательно являются числами, каким-то образом представленными в памяти ЭВМ. Под вычислением сегодня понимают, в частности, такие неарифметические действия, как обработка текстовой информации и работа с данными, представляющими множества, отношения, графы и т. п. При этом доля чисто арифметических операций в системе команд современных ЭВМ довольно небольшая. И как это ни парадоксально, современные ЭВМ из всего спектра решаемых ими задач менее всего приспособлены для выполнения именно арифметических вычислений. Принципиальное ограничение на выполнение арифметических вычислений накладывает способ представления чисел в памяти ЭВМ. А именно, ЭВМ допускает использование лишь конечных и конструктивных объектов, т. е. таких объектов, которые могут быть представлены последовательностью конечной длины, состоящей из символов некоторого, также конечного, алфавита. Из этого ограничения сразу следует, что в ЭВМ нельзя представить не только все трансцендентные и иррациональные числа, но и даже все рациональные числа. ЭВМ позволяет представить лишь конечное, хотя, быть может, и очень большое подмножество действительных чисел [1, 11, 26, 30, 39, 40, 41]. При этом особо выделяется множество целых чисел, для которых в ЭВМ, как правило, используется специальный способ представления. Происходит раздвоение множества целых чисел. Одно и то же целое число можно представить как:

— машинное целое число,

— машинное число с плавающей точкой.

Так, например, число семь можно записать как 7 и как 7. либо даже как 7.0. При этом не следует ожидать, что 7 обязательно равно 7.0.

В языке фортран логическое выражение

7. EQ. 7.0

не обязательно является истинным. Такое неестественное положение дел объясняется тем, что в ЭВМ используются обычно различные способы представления целых чисел и чисел с плавающей точкой.

В современных ЭВМ для кодирования данных, и в том числе чисел, как правило, используется двоичный алфавит, содержащий символы 0 и 1. Для представления числа отводится одно или несколько слов памяти либо часть слова. Таким образом, число в ЭВМ (машинное число) — это последовательность фиксированной длины из нулей и единиц:

AN...AK...A3A2A1

где AK — двоичная цифра 0 или 1. Форма представления числа, т. е. способ интерпретации указанной последовательности как некоторого действительного или целого числа, целиком зависит от конкретной ЭВМ. Рассмотрим некоторые широко используемые формы представления чисел в ЭВМ [11, 25, 33, 36].

Любое число A может быть представлено в виде: $A=m\cdot q^p$, где p — некоторое целое число, называемое порядком числа A, q — основание системы счисления, m — число, называемое мантиссой числа A. Если выполнено условие $q^{-1}\leqslant|m|<1$, то число A называется нормализованным. При представлении действительных чисел в памяти ЭВМ возникают ошибки, обусловленные тем, что дробное десятичное число не всегда точно представимо в двоичной системе счисления. Эти ошибки называются ошибками представления чисел в ЭВМ.

Рассмотрим коротко представление чисел в БЭСМ-6 и ЕС ЭВМ. Машинное число БЭСМ-6 содержит 48 разрядов [21, 36]. Для хранения мантиссы числа в слове БЭСМ-6 отводится 40 разрядов (с 1 по 40), а для хранения порядка — 6 разрядов (с 42 по 47). Разряды 48 и 41 указывают соответственно знак порядка числа A и знак мантиссы числа A. Шесть разрядов, отводимые под порядок, дают диапазон изменения двоичного порядка от —64 до +63. В десятичном представлении это соответствует диапазону представления абсолютных величин чисел в БЭСМ-6 примерно от 10^{-19} до 10^{19}. Числа, по абсолютной величине меньшие 10^{-19}, считаются равными нулю. Мантисса действительного числа может иметь до 12 десятичных цифр. Кроме того, для изображения действительных чисел используются числа с двойной точностью. Число с двойной точностью представляется в виде 96-разрядного двоичного числа и хранится в двух последовательных словах. Абсолютная величина чисел с двойной точностью находится в диапазоне примерно от 10^{-1232} до 10^{1232}. Мантисса действительного числа с двойной точностью содержит 80 разрядов, что соответствует представлению числа с 24 десятичными цифрами.

Целые числа представляются в ненормализованном виде с порядком 40 и с точкой, фиксированной после младшего разряда мантиссы. Абсолютное значение целых чисел находится в диапазоне от 0 до $2^{40}-1=1099511617775$.

Машинное слово ЕС ЭВМ в стандартной форме состоит из четырех последовательных байтов (32 разряда) [1, 33, 34]. Имеется также возможность работать со словами половинной и удвоенной длины. Используются три формы представления чисел: числа с плавающей точкой, числа с фиксированной точкой, целые числа.

Число с фиксированной точкой может быть представлено в формате полуслова и в формате полного слова. При этом используется соответственно 15 и 31 разряд. Максимальную величину целого положительного числа с фиксированной точкой представляют нулем в знаковом разряде и единицами во всех остальных разрядах. Этому соответствуют значения целого числа $2^{31}-1=2147483647$ в стандартном формате и $2^{15}-1=32767$ в нестандартном формате (формате полуслова). Минимальные значения целых отрицательных чисел соответственно равны -2147483648 и -32768.

Числа с плавающей точкой представляются в ЕС ЭВМ в двух форматах фиксированной длины: стандартном и длинном. Эти форматы отличаются только длиной мантиссы. Стандартное число с плавающей точкой располагается в одном слове, длинное — в двойном слове. Для знака числа отводится самый левый разряд, 7 разрядов после знакового занимает двоичный порядок, остальные 24 разряда отводятся для записи мантиссы. Абсолютные значения действительных чисел лежат в пределах примерно от $5.4\cdot10^{-79}$ до $7.2\cdot10^{75}$ [33]. Мантиссы стандартных чисел обеспечивают обычную точность вычислений, а мантиссы длинных чисел — более высокую точность. Мантисса действительного числа в стандартном формате имеет 7 десятичных цифр, в длинном — 17 десятичных цифр.

Использование ЭВМ для арифметических вычислений основывается на отображении аппроксимации F, отображающем множество действительных чисел R на множество машинных чисел M [11]:

$$F: R \longrightarrow M.$$

Арифметические операции $+$, $-$, $*$, $/$ (сложить, вычесть, умножить, разделить) над элементами множества R отображаются с помощью отображения α в машинные арифметические операции $+$, $-$, $*$, $/$ над элементами множества M. Машинные арифметические операции несколько отличаются от математических (используются округления и нормализация чисел, различная точность вычислений и т. п.) [11, 30, 41]. Отображения F и α обладают следующими свойствами:

1. отображение F определено не на всем множестве действительных чисел R, а на его части,

2. множество $M=F(R)$ конечно,

3. отображение F однозначно, обратное отображение F^{-1} неоднозначно,

4. из равенства $X \# Y = Z$, где $\# = +, -, *, /$, не следует, что $F(X)\ \alpha\ (\#)\ F(Y) = F(Z)$.

Свойство 4 является камнем преткновения на пути использования ЭВМ для арифметических вычислений.

Для машинных арифметических операций не выполняются ассоциативный и дистрибутивный закон арифметики. Отсюда, в частности, следует, что порядок выполнения арифметических операций на ЭВМ играет существенную роль.

Еще одно следствие состоит в том, что если имеются два различных алгоритма вычисления одной и той же величины, то их применение на ЭВМ может дать совершенно различные результаты.

При решении конкретной задачи получаемый результат вычислений будет иметь ошибку. Ошибка результата будет складываться из:

— неточности используемой математической модели,

— погрешности исходных данных,

— погрешности используемого приближенного метода,

— погрешности округления при вычислениях,

— погрешности выполнения арифметических операций.

При расчетах те или иные ошибки могут отсутствовать или их влияние может быть мало. Погрешность математической модели связана с принятыми допущениями, упрощающими решаемую задачу. Исходные данные, которые могут быть получены в результате экспериментальных измерений, как правило, имеют погрешность. Погрешность исходных данных является неустранимой. Погрешность метода связана, например, с тем, что при решении задачи обычно производную заменяют разностью, интеграл — суммой или бесконечный итерационный процесс обрывают после некоторого конечного числа итераций. Вычисления на ЭВМ выполняются с конечным числом значащих цифр, определенных конечностью разрядной сетки ЭВМ. Это свойство ЭВМ вносит в результат погрешность округления, которая может накапливаться в процессе вычислений.

На каждом шаге вычислений появляются ошибки округления. В зависимости от реализованного в алгоритме метода решения, эти ошибки округления могут либо расти, либо уменьшаться. Поэтому очень важно использовать алгоритмы, в которых ошибки округления не накапливаются. Такие алгоритмы называются *устойчивыми* [4, 38].

В данной книге при рассмотрении примеров мы ограничимся анализом погрешностей округления и погрешностей арифметических операций.

При выполнении арифметических операций на ЭВМ могут появляться числа с бóльшим количеством значащих цифр, чем у исходных чисел. Например, при умножении число значащих цифр может удвоиться, а при делении их можно получить в частном сколько угодно. Чтобы не

выходить за пределы разрядной сетки ЭВМ, производят округление полученных результатов до количества значащих цифр, определяемого разрядностью ЭВМ. Так, на БЭСМ-6 при выполнении арифметических операций используются 80 разрядов для представления мантиссы промежуточного результата вычислений, а мантисса окончательного результата содержит 40 разрядов. При этом естественно брать ближайшее число из представимых чисел, но возможны и другие варианты. Разность между округленным и округляемым числами называют ошибкой округления. Один из простейших способов округления заключается в следующем [12]. Пусть в результате вычислений получено число, представимое q-ичной дробью

$$X=0.A_nA_{n-1}A_{n-2}\ldots A_SA_{S-1}A_{S-2}\ldots A_1,$$

которую для простоты будем считать неотрицательной. В качестве результата выполнения операции округления числа X до S разряда берется число

$$X_S=0.A_nA_{n-1}\ldots A_S$$

Этот способ округления называется отбрасыванием младших разрядов [30]. Для ошибки округления такого способа округления справедливо соотношение

$$|X_S-X|\leqslant q^{-[n-(S-1)]},$$

где $n-(S-1)$ — число значащих цифр X_S. Равенство достигается в случаях, когда во всех разрядах числа X, начиная с некоторого $(S-1)$-го, стоят нули.

Достоинство данного способа округления — простота реализации, недостатки — большие ошибки округления и, кроме того, независимо от своей величины ошибка округления имеет один и тот же знак, противоположный знаку округляемого числа. А это может приводить к быстрому накоплению ошибок при вычислениях. Максимально возможную относительную ошибку для случая отбрасывания младших разрядов определим как отношение абсолютной ошибки числа X к самому числу X

$$\delta_X=\frac{\Delta_X}{X},$$

$\Delta_X=|X_S-X|\leqslant q^{-[n-(S-1)]}$ — минимально возможное значение $X=0.1$. Тогда

$$\delta_X\leqslant\frac{q^{-[n-(S-1)]}}{1/q}=q^{-(n-S)}.$$

На этом способе округления основаны другие способы округления. В самом деле, операцию округления можно рассматривать как отбрасывание младших разрядов, начиная с $S-1$ до первого, и последу-

ющее прибавление или вычитание некоторого числа, кратного $q^{-(n-S)}$. Чтобы ошибка была небольшой, надо, чтобы было малым и это число.

Более общепринятым способом округления является так называемое симметричное округление [30], которое можно записать следующим образом:

$$X_S=\begin{cases} X_S & , \text{ если } |X_S-X|<0.5\cdot q^{-(n-S)} \\ X_S+q^{-(n-S)}, & \text{если } |X_S-X|\geqslant 0.5\cdot q^{-(n-S)}. \end{cases}$$

Для этого способа округления максимально возможное значение относительной ошибки округления равно

$$\delta_X=\frac{\Delta_X}{X}\leqslant\frac{0.5\cdot q^{-(n-S)}}{1/q}=0.5\cdot q^{-(n-S)+1},$$

таким образом, симметричное округление имеет вдвое меньшую ошибку, чем простое отбрасывание младших разрядов. Иногда применяется более точное правило округления, учитывающее случай $|X_S-X|=0.5\cdot q^{-(n-S)}$. В зависимости от значения округляемой цифры X не изменяется либо становится равным $X_S+q^{-(n-S)}$. Приведенное описание способов и процесса округления дано здесь в схематичном виде. Подробное описание процесса округления приведено в [4, 12, 30].

Ограниченность разрядной сетки и как следствие этого необходимость выполнять округление получаемого результата приводит к тому, что арифметические операции, выполняемые на ЭВМ, имеют совершенно иные свойства, чем точные операции. Об этом уже говорилось. Далее, при реализации на ЭВМ различных форм записи одного и того же алгоритма возникают неодинаковые эффекты. Это связано с тем, что при описании алгоритмов допускается некоторая неоднозначность, определяемая коммутативностью, ассоциативностью и дистрибутивностью арифметических операций, которые не выполняются при вычислениях на ЭВМ. Арифметические операции на БЭСМ-6 и особенности их выполнения кратко описаны в приложении 2. На ЕС ЭВМ существует большой набор команд, выполняющих арифметические операции. Подробно информацию о выполнении арифметических операций на ЕС ЭВМ можно найти, например, в [1, 33, 34].

Для оценки ошибок, получаемых при вычислениях, нам потребуются формулы оценки абсолютной и относительной ошибок арифметических операций. Выпишем эти формулы [4].

1. Сложение: $X=X_1+X_2$, $X_1>0$, $X_2>0$.

При сложении чисел их абсолютные Δ_{X_1}, Δ_{X_2} и относительные δ_{X_1}, δ_{X_2} погрешности складываются:

$$\Delta_X=\Delta_{X_1}+\Delta_{X_2},$$
$$\delta_X=\delta_{X_1}+\delta_{X_2}.$$

При этом относительная погрешность суммы будет заключена между наибольшей и наименьшей относительными погрешностями слагаемых.

2. Вычитание: $X = X_1 - X_2$, $X_1 > X_2 > 0$.

При вычитании чисел абсолютная погрешность будет равна сумме абсолютных погрешностей уменьшаемого и вычитаемого:

$$\Delta_X = \Delta_{X_1} + \Delta_{X_2};$$

относительная погрешность результата будет больше, чем каждая из относительных погрешностей:

$$\delta_X = \frac{X_1 \cdot \delta_{X_1} + X_2 \cdot \delta_{X_2}}{X}.$$

Из формулы для относительной погрешности следует, что если $X_1 \gg X_2$ то $\delta_X \approx \delta_{X_1}$. Следовательно, в этом случае можно действовать, как и при сложении. Если $X_1 \approx X_2$, т. е. уменьшаемое и вычитаемое близки друг другу, то знаменатель дроби для относительной погрешности δ_X мал и, следовательно, относительная погрешность при вычитании близких чисел будет очень велика. Поэтому в данном случае получается большая потеря верных знаков. Отсюда следует, что надо стремиться избегать вычитания близких по величине чисел.

3. Произведение чисел: $X = X_1 \cdot X_2$.

При умножении чисел абсолютные и относительные погрешности их складываются:

$$\Delta_X = \frac{X}{X_1}\Delta_{X_1} + \frac{X}{X_2}\Delta_{X_2},$$
$$\delta_X = \delta_{X_1} + \delta_{X_2}.$$

4. Частное двух величин: $X = X_1/X_2$.

В этом случае относительные погрешности будут складываться.

$$\Delta_X = \frac{|X_2| \cdot \Delta_{X_1} + |X_1| \cdot \Delta_{X_2}}{X_2^2},$$
$$\delta_X = \delta_{X_1} + \delta_{X_2}.$$

Приведенные формулы дают выражение для определения ошибки результата каждого из четырех арифметических действий как функции от величин чисел, участвующих в арифметических операциях, и абсолютных ошибок для них. Ошибка округления результата в данных операциях здесь не учитывается, поэтому при необходимости определения полной ошибки результата надо к вычисленной по одной из четырех формул ошибке результата добавить отдельно ошибку округления.

Рассмотрим на примере суммирования n положительных упорядоченных по возрастанию чисел A_i ($i=1, \ldots, n$) влияние порядка выполнения операций на ошибку получаемого результата. Формула для оценки полной ошибки суммы положительных чисел имеет вид

$$\Delta_S = A_1\delta_{A_1} + A_2 \cdot \delta_{A_2} + \ldots + A_n\delta_{A_n} + \delta_1 \cdot (A_1 + A_2) + \ldots$$
$$\ldots + \delta_{n-1}(A_1 + \ldots + A_n) + \delta_n.$$

В этой формуле δ_{A_i} — относительные ошибки представления чисел в ЭВМ, δ_i — относительные ошибки округления чисел при каждой следующей операции сложения. Положим все $\delta_{A_i}=\delta_A$ и $\delta_i=\delta$, тогда получаем:

$$\Delta_S=\delta_A\Sigma_S+\delta[(n-1)A_1+(n-1)A_2+\ldots+2A_{n-1}+A_n].$$

Здесь $\Sigma_S=A_1+A_2+\ldots+A_n$. Из последнего выражения очевидным образом вытекает, что максимально возможная ошибка суммы упорядоченных положительных чисел будет меньше, если начинать складывать с меньших по величине чисел.

Рассмотрим ряд примеров того. к чему могут привести особенности выполнения операций над числами, представленными в форме с плавающей точкой. Во всех приведенных примерах вычисления выполнены с обычной точностью.

1. Сравним результаты вычисления выражений

S1 = A + 0.5*A**2*B**5

S2 = A + A**2*B**5

Пусть A будет маленьким числом, например A = 0.00001, а B = 0.1. Программа на БЭСМ-6 для вычисления S1 и S2 имеет вид

```
01      PROGRAM COMPR
02      DATA A/1.0E—05/ ,B/0.1/
03      S1 = A + 0.5*A**2*B**5
04      S2 = A + A**2*B**5
05      PRINT 1,S1,S2
06    1 FORMAT(1X,3HS1=,E18.11,4X,3HS2=,E18.11)
07      END
```

При расчетах на ЕС ЭВМ приводимые в примерах программы необходимо незначительно изменить: убрать заголовок программы и изменить спецификации формата вывода, поскольку для обычной точности числа на ЕС ЭВМ имеют 7 значащих цифр и нет смысла выводить больше 7 цифр.

В результате выполнения программы получаем:

— на БЭСМ-6

S1 = ␣␣1.00000000005—05␣␣␣␣S2 = ␣␣1.00000000010—05

— на ЕС ЭВМ

S1 = ␣␣0.1000000E—04␣␣␣␣S2 = ␣␣0.1000000E—04

S1 и S2 должны отличаться друг от друга на величину 0.5*A**2**B**5 = 0.5*10**(—15). В данном случае на БЭСМ-6 хватает точности представления чисел и S2 = S1 + 0.5*A**2*B**5. А на ЕС ЭВМ из-за того, что точность вычислений имеет 7 значащих цифр, получаем S1=S2. Таким образом, при вычислениях на ЭВМ может получиться

так, что числа будут равны, в то время как на самом деле должны получиться разные числа.

2. Сравним результаты вычислений по формулам

```
S1 = (A + B)**3 - A**3
S2 = 3*A**2*B + 3*A*B**2 + B**3
```

для A = 199.0 и B = 0.0001. С точки зрения обычной арифметики S1 = S2, но при вычислении на ЭВМ можем получить S1 ≠ S2. Соответствующая программа вычисления S1 и S2 на БЭСМ-6 имеет вид

```
01        PROGRAM COMPP
02        DATA A/199.0/,B/0.0001/
03        S1 = (A + B)**3 - A**3
04        S2 = 3.0*A**2*B + 3.0*A*B**2 + B**3
05        PRINT 1,S1,S2
06      1 FORMAT(1X,3HS1=,E18.11,4X,3HS2=,E18.11)
07        END
```

В результате выполнения программы получаем:

— на БЭСМ-6

S1 = ␣␣1.18802795410 + 01␣␣␣␣S2 = ␣␣1.18803059700 + 01

— на ЕС ЭВМ

S1 = ␣␣0.1000000E 02␣␣␣␣S2 = ␣␣0.1188030E 02

Таким образом, вычисления по этим двум формулам приводят к разным результатам. Но если на БЭСМ-6 результат вычисления совпадает по пяти значащим цифрам, то на ЕС ЭВМ — только по одной. Вычисление по второй формуле дает меньшую погрешность. Результаты вычислений на БЭСМ-6 и ЕС ЭВМ совпадают в семи значащих цифрах.

3. Вычислить

```
S1 = A**3
S2 = A*A*A
S3 = A**3.0
```

для A = 0.4568 и A = 0.976854321. Должно быть: S1 = S2 = S3. Однако, вычисляя на ЭВМ, получаем (как видно из примеров 1 и 2, программы для вычислений приводимых выражений тривиальны, поэтому для следующих ниже примеров читатель легко сможет написать соответствующие программы):

БЭСМ-6	ЕС ЭВМ
для A = 0.4568:	
S1 = 9.53187384318 — 02	S1 = 0.9531867E — 01
S2 = 9.53187384318 — 02	S2 = 0.9531867E — 01
S3 = 9,53187384317 — 02	S3 = 0,9531873E — 01

Сравнивая эти результаты, имеем на БЭСМ-6 совпадение в одиннадцати значащих цифрах, на ЕС ЭВМ — в пяти значащих цифрах. Наилучшее совпадение результатов, полученных на БЭСМ-6 и ЕС ЭВМ, имеет место при вычислении по формуле для S3.

БЭСМ-6	ЕС ЭВМ
для A = 0.976854321:	
S1 = 9.32157730706 — 01	S1 = 0.9321574E 00
S2 = 9.32157730706 — 01	S2 = 0.9321574E 00
S3 = 9.32157730708 — 01	S3 = 0.9321576E 00

На БЭСМ-6 имеем совпадение в одиннадцати значащих цифрах, на ЕС ЭВМ — в шести. Так же, как и в предыдущем случае, наилучшее совпадение результатов на обеих ЭВМ — при вычислении по формуле для S3.

4. Вычислить

```
B = SQRT(A)
B1 = A**0.5
```

При вычислении этих выражений получилось:

БЭСМ-6	ЕС ЭВМ
для A = 0.4568:	
B = 6.75869809948 — 01	B = 0.6758698E 00
B1 = 6.75869809947 — 01	B1 = 0.6758697E 00
для A = 9.76854321:	
B = 3.12546687872 + 00	B = 0.3125466E 00
B1 = 3.12546687872 + 00	B1 = 0.3125466E — 01

Для значения $A<1$ B и B1, полученные на БЭСМ-6, отличаются в двенадцатой значащей цифре, а на ЕС ЭВМ — в седьмой. Для $A>1$ значения B и $B1$, вычисленные на обеих ЭВМ, совпадают во всех значащих цифрах.

5. Вычислить

```
S1 = 1.0/(1.0/A)
S2 = 1.0 + A*(1.0/A)
```

По правилам обычной арифметики должно быть S1 = A. Но при вычислении S1 на ЭВМ может получиться $S1 \neq A$. Во втором случае должно быть S2 = 2, но при вычислении выражения S2 на ЕС ЭВМ для значения A = 9876.543 получаем S2 = 0.1999999E 01.

6. Вычислить

```
S1 = A + B
S2 = (A**2 — B**2)/(A — B)
```

В этом случае S1 = S2. Посмотрим, что получится при вычислении S1 и S2 на ЭВМ для A и B, не сильно отличающихся друг от друга.

Вычисления дают:

БЭСМ-6	ЕС ЭВМ
для A = 0.532578591233, B = 0.521287952311.	
S1 = 1.05386654354	S1 = 0.1053866E 01
S2 = 1.05386654355	S2 = 0.1053862E 01
БЭСМ-6:	ЕС ЭВМ:
для A = 532578.591233, B = 521287.952311.	
S1 = 1.05386654354 + 06	S1 = 0.1053866E 07
S2 = 1.05386654351 + 06	S2 = 0.1053863E 07

Для выбранных значений A и B S1 и S2 отличаются в двенадцатой значащей цифре (БЭСМ-6) и в седьмой (ЕС ЭВМ). Для расчетов, проведенных на БЭСМ-6 и ЕС ЭВМ, имеет место совпадение семи значащих цифр для $A<1$, $B<1$ и шести значащих цифр для $A>1$, $B>1$.

7. Во многих книгах по численным методам приводится пример вычисления корня квадратного уравнения

$$X**2+B*X+C=0,$$

у которого коэффициент B велик, а коэффициент C мал. Корни этого уравнения можно вычислить по хорошо известным формулам:

X1 = —0.5*B + SQRT(0.25*B**2 — C),
X2 = —0.5*B — SQRT(0.25*B**2 — C).

Вычислим значения X1 и X2 при значениях коэффициентов квадратного уравнения B = 2000 и C = 1. Получаем:

БЭСМ-6	ЕС ЭВМ
X1 = —5.00001013279 — 04	X1 = —0.4882812E — 03
X2 = —1.99999949999 + 03	X2 = —0.1999999E + 04

Вычисленные на БЭСМ-6 и ЕС ЭВМ значения корня X2 совпадают в семи значащих цифрах, а значения корня X1 имеют весьма существенное различие. Определим теперь значение корня X1 по формуле

X1 = —C/(0.5*B + SQRT(0.25*B**2 — C),

полученной из предыдущей формулы для вычисления X1, умножая и деля ее на выражение

0.5*B + SQRT(0.25*B**2 — C).

В этом случае получаем:

БЭСМ-6	ЕС ЭВМ
X1 = —5.00000125000 — 04	X1 = —0.5000000E — 03

В этом случае результаты вычислений на ЕС ЭВМ и БЭСМ-6 отличаются в седьмом знаке. По сравнению с результатом, полученным по первой

формуле, результат, полученный по второй формуле, имеет гораздо более высокую точность.

Другой алгоритм, в котором учитывается особенность арифметики с плавающей точкой и конечность разрядной сетки ЭВМ, использует формулу Виета

X1*X2=C,

т. е. сначала вычисляется корень с бо́льшим значением по абсолютной величине из двух корней этого уравнения. Другой корень (в данном примере это $X1$) может быть вычислен так:

X1=C/X2.

Приведенные примеры подтверждают тот факт, что машинная арифметика с плавающей точкой порождает ошибки, обусловленные тем, что числа в ЭВМ представляются не точно. При этом нужно иметь в виду, что ошибки округления часто зависят от конкретных чисел, с которыми производятся вычисления, а также от метода и алгоритма решения задачи. Следует отдавать себе отчет в том, что при достаточно длинной последовательности вычислений возникающие ошибки могут привести к потере всех значащих цифр. Поэтому, выбирая тот или иной метод для решения на ЭВМ поставленной задачи, необходимо его тщательно исследовать с точки зрения точности получаемого решения.

3.6. Ошибки ввода-вывода

В языке фортран используются два способа обмена данными с внешними устройствами: форматный и бесформатный. Большинство ошибок связано именно с использованием форматного обмена, поэтому здесь будет рассмотрен в основном именно этот способ ввода-вывода данных.

Формат вводимых и выводимых данных описывается оператором FORMAT, который задает синтаксическую структуру представления и интерпретации данных. Главная особенность форматного обмена состоит в том, что данные не имеют самостоятельной интерпретации — интепретация данных возможна только совместно со спецификациями, задаваемыми оператором FORMAT. Причем одни и те же данные могут быть интерпретированы по-разному, в зависимости от используемого формата. Например, данные, представленные на перфокарте (магнитной ленте и т. п.) как

номер позиции 12...

123456789ABCDE␣3.1415␣29.347E12—16.592

могут быть интерпретированы, например, следующими способами:

1) целое число 123456789, текст 'ABCDE' и действительные числа 3.1415, 29.347E12, —16.592

```
      READ (K,1) N,X,Y,Z
    1 FORMAT(I9,'XXXXX',F7.4,E10.3,F7.1)
```

2) целое число 3456, целое число 3, целое число 1415029, шестнадцатеричное число 7E12

```
      READ (K,1) N,M,J,L
1  FORMAT(2X,I4,8X,I2,1X,I7,3X,Z4)
```

3) целые числа 2, 45, 67, тексты '89AB' и 'CDE', действительные числа 3.1 и 41.5

```
      READ (K,1) N1,N2,N3,L,X,Y
1  FORMAT(1X,I1,1X,I2,I2,A4,'ABC', 1X,2F3.1)
```

4) целое число 56 и строка символов 'BCDE', целое число 3, действительные числа 1.415029 и 34.7

```
      READ (K,1) N,L,M,X,Y
1  FORMAT(4X,I2,4X,A4,1X,I1,1X,F7.6,1X,F3.1)
```

5) целое число 234, шестнадцатеричное число 9ABC и строка символов 'DE␣3'

```
      READ (K,1) N,B,L
1  FORMAT(1X,I3,4X,Z4,A4)
```

Хотя приведенный список возможных интерпретаций далеко не полон, тем не менее он наглядно показывает, сколь велика возможность сделать ошибку, которая может остаться незамеченной.

При организации форматного ввода следует помнить следующее:

1. Пробелы внутри числа, вводимого по спецификациям I, F, E, D, интерпретируются как цифра нуль '0', Например, 3.14 67 эквивалентно 3.14067.

2. При использовании спецификаций F, E, D точка в числе может быть опущена. Ее положение определяется по спецификации.

3. Если положение точки в записи числа не соответствует используемой спецификации, то ее положение определяется записью, а не спецификацией.

4. При записи числа по спецификации формата E буква E перед порядком может отсутствовать, но при этом обязательно нужно указывать знак порядка.

5. При записи числа в форме с порядком порядок может состоять из одной цифры.

Еще одна ошибка форматного ввода вывода возникает при несоответствии типа вводимого (выводимого) значения используемой для этого спецификации. Особенно это опасно при вводе данных, когда вводимые данные таковы, что они допускают интерпретацию и с ошибочной спецификацией, как это было показано выше. В этом случае ошибки не будут зафиксированы, но переменным будут присвоены не те значения, какие следовало бы.

Оператор FORMAT занимает особое положение среди других операторов. Это связано с тем, что он имеет свойства, схожие со свойствами как исполняемых, так и неисполняемых операторов. Хотя оператор

FORMAT и относится к числу неисполняемых операторов, в отличие от последних он никак не влияет на трансляцию других операторов программы (аналогично исполняемым оператором). Кроме того, как и всякому исполняемому оператору, ему, в отличие от других неисполняемых операторов, при трансляции соответствует некоторый код в результирующей программе. Этот код представляет собой текстовую константу, соответствующую тексту, следующему за словом FORMAT. Такая трансляция оператора FORMAT является источником целого ряда ошибок. Прежде всего многие трансляторы не анализируют текст, следующий за словом FORMAT, а просто переносят его один к одному в код программы. Таким образом, ошибки, допущенные при записи спецификаций, будут обнаружены только при исполнении программы, как, например, в операторе

```
1 FORMAT(X5,'ABC=',E20.10,S12.6)
```

Этот оператор мог появиться из-за опечаток, например, в операторе

```
1 FORMAT(X5,'ABC=',E20.10,F12.6)
```

Более того, есть реализации трансляторов, которые в программе

```
      REAL FORMAT(3)
      DO 1 I2=1,3
      PRINT 2,I2
    2 FORMAT(I2)=0.
    1 CONTINUE
```

воспримут оператор

```
    2 FORMAT(I2)=0.
```

как оператор формата, а не как оператор присваивания нуля элементу массива FORMAT. При выполнении программа напечатает:

```
**␣1
**␣2
**␣3
```

Одна из распространенных ошибок возникает в том случае, когда программа пытается прочитать больше данных, чем их записано на носителе. Нередки случаи, когда данные, записанные бесформатным способом, пытаются прочитать по формату (или наоборот). В двух последних случаях результат этих неправомерных действий непредсказуем, так как зависит от многих факторов.

Вообще, при работе на ЭВМ программисту всегда следует хорошо ознакомиться с организацией ввода-вывода в используемой реализации языка. Дело в том, что здесь возможны отклонения от стандарта языка, т. к. транслятор, как правило, использует средства ввода-вывода, предоставляемые ему операционной системой, а эти средства могут довольно существенно отличаться в различных операционных системах.

ГЛАВА 4

РЕШЕНИЯ ЗАДАЧ

Эврика!
Архимед

Решение задачи 1. В некоторых реализациях транслятора фортрана операторы FORMAT не анализируются при трансляции на наличие в них ошибок, а просто переводятся в текстовую константу. В приведенной программе оператор цикла в строке 08 начинается с шестой позиции, используемой для признака продолжения, и поэтому строка 08 воспринимается транслятором как продолжение оператора в строке 07, т. е. оператора FORMAT. Если транслятор не анализирует оператор FORMAT, то любая ошибка, аналогичная данной, в нем зафиксирована не будет (даже при выполнении программы), а строка 08 просто «приклеится» к строке 07 и не окажет никакого влияния на выполнение программы, как будто ее вовсе нет в тексте программы. Значение счетчика цикла в операторе 04 после выхода из цикла равно 6 (для транслятора фортран-Дубна на БЭСМ-6). Поэтому в операторе 09 единица будет присвоена шестому элементу массива.

Результат выполнения этой программы зависит от используемого транслятора (см. п. 3.3.4). Так, например, на ЕС ЭВМ получим:

```
** K=0 0 0 0 0 2 2 2 2 2
** K=0 0 0 0 1 2 2 2 2 2
```

Различие результатов, полученных на БЭСМ-6 и ЕС ЭВМ, объясняется разными способами трансляции оператора цикла DO на этих ЭВМ. Значение переменной цикла после выхода из тела цикла при выполнении программы на ЕС ЭВМ равно 5.

Решение задачи 2. В операторе 04 в идентификаторе ТОРМОЗ буква «З» заменена на цифру «3». Это привело к тому, что в программе появилась новая переменная ТОРМО3, которой и присвоено значение, равное 1.0 в операторе 04. Аналогичный эффект вызовет, например, замена буквы «О» цифрой «0». При этом транслятор не зафик-

сирует никакой ошибки, так как переменная ТОРМОЗ стоит слева от знака присваивания (см. п. 3.1).

Решение задачи 3. Различие полученных результатов вызвано тем, что в первом случае происходит потеря большего количества значащих цифр слагаемых, чем во втором. Это следствие ограниченности разрядной сетки ЭВМ. В п. 3.5 уже обсуждалась причина потери точности при операциях сложения и вычитания: при сложении чисел A и B происходит потеря $|P(A)-P(B)|$ значащих знаков меньшего по абсолютной величине из чисел (здесь $P(X)$ — двоичный порядок числа X). Известно, что сумма N членов гармонического ряда растет как функция $\ln(N)$. Поэтому при суммировании первым способом мы сталкиваемся с ситуацией, когда порядок одного слагаемого (частичной суммы) постоянно увеличивается, а порядок второго слагаемого (очередного члена ряда) уменьшается, а значит, растет число теряемых значащих цифр второго слагаемого.

При суммировании в обратном порядке — от последнего члена суммы к первому,— хотя увеличивается порядок частичной суммы, но и порядок очередного суммируемого члена растет, так что имеет место потеря меньшего количества значащих цифр. Таким образом, суммирование вторым способом дает более точный результат.

Если вычислять сумму очень большого числа членов гармонического ряда, то наступит такой момент, когда первая сумма перестанет расти, хотя вторая сумма будет давать хорошее приближение.

Подумайте над вопросом: при каком числе членов ряда будет наблюдаться такой эффект на ЭВМ, на которой вы работаете?

Из этой задачи можно сделать следующие выводы:

1. При суммировании (вычитании) необходимо группировать слагаемые таким образом, чтобы разница порядков была минимальна.

2. Точность получаемого результата определяется не только формулами, по которым производятся вычисления, но и порядком использования этих формул.

Решение задачи 4. Ошибка в программе обусловлена некорректным использованием имени стандартной функции SIN. Как уже упоминалось в п. 3.1, в языке фортран нет зарезервированных ключевых слов. Это относится и к именам стандартных функций. Кроме того, сам идентификатор не определяет объект полностью. Необходимо также знать, в каком контексте он используется. В данном случае из представленной программы не следует, что SIN — это имя функции. Транслятор интерпретирует SIN как имя переменной. Чтобы исправить ситуацию, необходимо в вызывающую программу поместить описание:

```
EXTERNAL SIN
```

и, кроме того, иметь модуль, вычисляющий функцию SIN, так как некоторые трансляторы реализуют стандартные функции не в виде отдельных модулей, а вставляют соответствующие команды вычисления функ-

ции непосредственно в программу (см., например, [3]) или обращение к стандартным функциям происходит особым образом, что никаким образом не отражается в операторе 12 подпрограммы.

Решение задачи 5. Как уже отмечалось в п. 3.1, в языке фортран отсутствуют зарезервированные ключевые слова, используемые во многих языках программирования для выделения операторов. Это, в частности, относится и к оператору цикла DO. Распознавание этого оператора транслятором производится не по первым двум буквам, а по символу, идущему за параметром, указывающим начальное значение переменной цикла. Для оператора цикла DO этот символ должен быть запятой «,». В приведенной программе в операторе 06

```
DO 2 K=6.10
```

допущена ошибка: вместо запятой, разделяющей параметры, указывающие начальное и конечное значение переменной цикла, стоит точка. Учитывая то, что при записи программы на языке фортран пробелы не имеют значения, становится ясно, что транслятор воспринимает это как оператор присваивания:

```
DO2K=6.10
```

Следует отметить, что на ЕС ЭВМ результат выполнения программы будет несколько иным. А именно:

```
** K=5
** X=␣␣2␣␣4␣␣8␣16␣␣1␣␣0␣␣0␣␣0␣␣0␣␣0
```

Различие вызвано тем, что на БЭСМ-6 и на ЕС ЭВМ используются разные тесты на завершение цикла (см. п. 3.3.4). В фортране Дубна на БЭСМ-6 сначала вычисляется очередное значение параметра цикла, а затем проверяется, не превосходит ли оно конечное значение. На ЕС ЭВМ в конце цикла сначала проверяется условие окончания цикла, и если это условие не выполняется, то вычисляется очередное значение параметра цикла.

Решение задачи 6. Данная задача аналогична задаче 3. Причина различия полученных результатов всех трех сумм обусловлена округлением (см. п. 3.5). ЭВМ не может производить вычисления с бесконечно большой точностью. Из-за ограниченной разрядности любое действительное число в ЭВМ представлено конечным набором значащих цифр. Поэтому при выполнении арифметических операций над числами с плавающей точкой почти всегда происходит округление, что приводит к искажению окончательных результатов. При сложении (вычитании) сначала выравниваются порядки слагаемых, в результате чего мантисса меньшего по абсолютной величине числа сдвигается вправо на столько разрядов, сколько нужно для того, чтобы порядки стали равными. При сдвиге происходит потеря значащих цифр. При сложении очень большого и очень малого (относительно первого) числа может получиться

так, что сумма будет равна большему числу, т. е. добавление к сумме малого числа не изменяет ее. Поэтому, чтобы ошибка была меньше, надо складывать сначала маленькие по модулю числа в порядке возрастания, а затем к этой сумме прибавлять остальные слагаемые также в порядке возрастания их абсолютной величины. Обоснование этого факта приведено в п. 3.5.

Решение задачи 7. В качестве решения задачи можно предложить следующую программу:

```
01        PROGRAM TASK
02        CALL ZERO(1)
03        J=1
04        PRINT 1,J
05      1 FORMAT(' J=',I1)
06        STOP
07        END
      C ----------------------------------------
08        SUBROUTINE ZERO(K)
09        K=0
10        RETURN
11        END
```

Приведенное решение справедливо не для всех реализаций языка фортран. Предлагаемое решение предполагает, что:

1) каждая константа, используемая в модуле, представляется в нем единственным экземпляром, независимо от того сколько раз и где она встречается в этом модуле,

2) при использовании константы в качестве фактического параметра подпрограммы/функции передается адрес этой константы.

Следует иметь в виду, что не для всех реализаций трансляторов фортрана эти предположения выполняются. Некоторые ЭВМ имеют формат команд, при котором константа включается непосредственно в код команды. При этом каждому использованию константы соответствует свой экземпляр, хранимый в памяти ЭВМ.

В ряде реализаций транслятора фортрана для каждой константы, передаваемой в качестве фактического параметра, создается ее копия, адрес которой и передается в подпрограмму/функцию.

Если указанные предположения имеют место, то в подпрограмме ZERO затирается константа 1 (оператор 09). В слово, содержащее единицу, записывается константа, равная нулю. При выполнении оператора 03 на самом деле переменной J присваивается значение, равное нулю, из-за того, что в слово, в котором находилась константа, равная 1, была записана константа, равная 0.

Решение задачи 8. Программа может выглядеть так:

```
01        PROGRAM TASK
02        DIMENSION X(10)
```

```
03        DO 1 K=1,200
04      1 X(K)=0.
05        STOP
06        END
```

При решении задачи использованы два предположения, которые справедливы для большинства реализаций языка фортрана. Во-первых, во время выполнения программы не производится контроль за выходом значения индекса массива за границы массива. Во-вторых, размер программы TASK довольно мал (примерно 100 слов), поэтому независимо от того, в каком месте внутри программы будет отведена память для массива X, при записи нуля в элементы массива с номерами от 11 до 200 часть памяти будет затерта.

Решение задачи 9. В данном случае в программе имеется несоответствие количества фактических и формальных параметров (операторы 02 и 05). Для решения задачи необходимо знать, как передаются адреса параметров при обращении к фортранным подпрограммам (см. п. 3.4.2). При выполнении программы, написанной на языке фортран, передаются не сами фактические параметры, а их адреса. Передача адресов происходит через выделенный участок памяти. При выполнении оператора CALL происходит последовательная засылка адресов фактических параметров в этот участок памяти. Выполнение каждой подпрограммы начинается с того, что из выделенного участка памяти выбираются адреса фактических параметров и эти адреса засылаются в рабочую память модуля.

В данной задаче подпрограмма SUBR (оператор 05) имеет один формальный параметр, а обращение к ней записано с двумя параметрами (оператор 02). Вследствие этого при выполнении оператора CALL SUBR(X,Y) происходит обращение за пределы памяти, выделенной для параметров подпрограммы. В этом случае результат выполнения программы непредсказуем. На БЭСМ-6 при выполнении программы в мониторной системе Дубна возможна ошибка типа «число в чужом листе» или затирание памяти (см. п. 3.4.2).

Решение задачи 10. Первое, на что следует обратить внимание,— это различие значений переменной B: присваиваемого и печатаемого. Различие составляет:

1. 0*10**(—12)—9.99999999947*10**(—13)=5.3*10**(—25)

Хотя различие и мало, оно имеется и демонстрирует тот факт, что представление рациональных чисел в ЭВМ не всегда является точным, так как для записи чисел в программе мы пользуемся десятичной системой счисления, а в ЭВМ используется двоичная система счисления.

Теперь о переменных C и D. При сложении A и B переменная B воспринимается как нуль, так как разность порядков чисел A и B слишком велика (24 десятичных порядка), и поэтому ее значение не влияет на результат. При вычитании этих же чисел значение B уже влияет на

результат. Это вызвано тем, что при вычитании приходится «занимать» из старших разрядов и этот процесс доходит до самой старшей значащей цифры.

Наибольшее недоумение вызывает полученное значение E. Теоретически

```
E=C—D= (A+B)—(A—B)=2*B=2*10**(—12)
```

Различие почти на 12 порядков кажется невероятным.

Этот эффект объясняется спецификой представления чисел с плавающей точкой и особенностями выполнения арифметических операций в ЭВМ (см. п. 3.5 и приложение 2). Эта же программа на других ЭВМ даст иные результаты.

На ЕС ЭВМ:

```
A =␣0.9999999959040E␣12   B =␣0.9999995623233E — 12
C =␣0.9999999959040E␣12   D =␣0.9999999959040E␣12
E =␣0.0
```

на СМ-2:

```
A =␣␣.1000000102399E+11   B =␣␣.9999999439625E — 10
C =␣␣.1000000102399E+11   D =␣␣.1000000102399E+11
E =␣␣.0000000000000E+00
```

Решение задачи 11. Ошибка в программе типична для начинающих программистов, которые забывают о различии между действительными и целыми числами. Целые числа (тип INTEGER) в фортране не являются подмножеством действительных чисел (тип REAL), хотя для обозначения арифметических операций как над действительными, так и над целыми числами используются одни и те же знаки операции, и даже допускаются смешанные выражения, включающие переменные обоих типов. Об этом всегда следует помнить. В частности, операция деления чисел, для которой используется символ /, интерпретируется различным образом для действительных и целых чисел. Если для действительных чисел эта операция достаточно хорошо соответствует ее традиционному пониманию, то при делении целых чисел частное всегда является целым числом, которое получается в результате отбрасывания дробной части. Как раз такой случай имеет место в приведенной программе в операторе 04:

```
1 S=S+1/K
```

Здесь происходит деление двух целых чисел 1 и K. Результат деления для всех K>1 будет равен нулю, а при K=1 получается 1. Исправить эту ошибку можно, сделав либо K, либо 1 действительным числом. Из трех приводимых ниже вариантов наиболее предпочтительным является последний:

```
S=S+1./K
S=S+1/FLOAT(K)
S=S+1./FLOAT(K)
```

Решение задачи 12. Как упоминалось в п. 3.5, числа с плавающей точкой представляются мантиссой и порядком. Мантисса числа хранится в нормализованном виде, т. е. первая старшая цифра мантиссы равна 1. Таким образом, мантисса имеет вид 1 A2 A3...AN.

Рассмотрим два числа: X и Y=X/2**N. Если операция сложения выполняется без округления, то X+Y=X, так как при выравнивании порядков для сложения значащие цифры числа Y выйдут за разрядную сетку. Этот эффект и лежит в основе решения данной задачи. Если положить X=1.0, то нужно найти такое минимальное N, что

1.+1./2**N=1.

Максимальное число E такое, что на ЭВМ 1+E=1 является мерой точности представления чисел (иногда оно называется *машинное эпсилон*). Ниже приведена программа вычисления N и E и результат ее работы на БЭСМ-6:

```
01      PROGRAM TASK
02      DATA K/0/
03      A=FLOAT(1)
04      X=A
05    1 X=X/2.
06      K=K+1
07      B=A+X
08      IF(B .GT. A) GO TO 1
09      X=X*2.
10      PRINT 2,K,X
11    2 FORMAT('␣␣␣␣N=',I3,'␣␣␣␣X=',E20.12)
12      END
**␣␣␣␣N=␣40␣␣␣␣X=␣␣␣␣1.818989403545-12
```

Отметим, что к приведенной программе следует относиться с осторожностью. На результат ее вычислений могут существенно повлиять особенности представления действительных чисел, выполнение арифметических операций и трансляция программы. В качестве подтверждения сказанного ниже приводится несколько иной вариант такой программы и результат ее работы на БЭСМ-6:

```
01      PROGRAM TASK
02      DATA K/0/,X/1.0/
03    1 X=X/2.
04      K=K+1
05      IF(1.+X .GT. 1.) GO TO 1
06      PRINT 2,K,X
07    2 FORMAT('␣␣N=',I3,'␣␣X=',E20.12)
08      STOP
```

```
09        END
**␣␣N=␣66␣␣X=␣␣␣0.000000000000+00
```

Подумайте, как можно объяснить такой результат.

Следует заметить, что на некоторых ЭВМ при представлении чисел с плавающей точкой не все разряды числа явно представляются в памяти, например, в микро-ЭВМ Электроника-60. Так как старший двоичный разряд мантиссы нормализованного положительного числа всегда должен быть равен 1, то поэтому он не хранится в памяти ЭВМ. В первой из приведенных программ будет вычислено на один разряд меньше, чем есть на самом деле.

Решение задачи 13. Пример решения задачи:

```
01        PROGRAM TASK
02        DIMENSION ARRAY(10)
03        CALL SUBRGO(ARRAY(10))
04        STOP
05        END
      C-----------------------------------------
06        SUBROUTINE SUBRGO(SUBR)
07        CALL SUBR
08        RETURN
09        END
```

При решении задачи использован эффект несоответствия классов фактического и формального параметров в операторах 03 и 06. Оператором 03 адрес 10-го слова массива ARRAY будет передан в подпрограмму. А в подпрограмме SUBRGO формальным параметром является имя подпрограммы и для ее выполнения (оператор 07) управление будет передано по адресу этой подпрограммы.

Решение задачи 14. В программе не учтено ограничение на диапазон целых чисел, представимых в ЭВМ. Максимальное целое положительное число для БЭСМ-6 равно $2^{40}-1$ (для ЕС ЭВМ — $2^{24}-1$). Поэтому невозможно непосредственно представить число, большее этого предела. Кроме того, как на БЭСМ-6, так и на ЕС ЭВМ не фиксируется переполнение при выполнении оператора 04:

```
1 N=N*2
```

Так как любая степень числа 2 представляется в ЭВМ как 100...0 в двоичной системе счисления, то, начиная с некоторого шага (в зависимости от числа разрядов, отводимых для представления целых чисел), когда возникает переполнение, число будет «представлено» одними нулями.

Подумайте, как, используя эти факты, написать программу, определяющую число разрядов в представлении целого числа.

Решение задачи 15. Тип переменной или функции при выполнении программы определяется либо идентификатором, либо явным описанием типа. Тип функции FUNCT в подпрограмме INSUM, ввиду отсутствия явных описаний, считается действительным (REAL), так как идентификатор начинается с буквы F. При обращении к подпрограмме INSUM в качестве фактического параметра передается имя целой функции INTC, имеющей тип INTEGER, тип которой также устанавливается по идентификатору. В результате этого в операторе 08 происходит деление на ненормализованное число (см. п. 3.5 и [21, 36]), что и фиксируется как деление на нуль.

Решение задачи 16. Применяя формулу для определения ошибки результата вычисления суммы N положительных чисел (см. п. 3.5), в данном случае имеем:

$$\Delta = \sum_{i=1}^{15} A_i \delta_A + \delta (14A_1 + 14A_2 + 13A_3 + \ldots + 2A_{14} + A_{15})$$

Так как первое слагаемое в этой формуле не влияет на способ суммирования, то в дальнейших рассуждениях оно не учитывается. Поскольку слагаемые не сильно различаются между собой, то можно записать каждое слагаемое в виде

$$A_i = A_0 + \varepsilon_i, \qquad i = 1, \ldots, 15,$$

т. е. в виде суммы некоторого числа A_0 и малой добавки ε_i. Учитывая это, оценку ошибки вычисления суммы запишем как

$$\tilde{\Delta} = 119 (A_0 + \varepsilon_m) \cdot \delta,$$

где $\varepsilon_m = \max\limits_i \varepsilon_i$, а δ — ошибка округления при выполнении арифметической операции.

Рассмотрим теперь другой способ суммирования — суммирование по группам (три группы по четыре числа и одна группа из трех чисел). Используя ту же формулу, для каждой группы чисел запишем:

$$\begin{aligned}
\Delta_1 &= \delta (3A_1 + 3A_2 + 2A_3 + A_4) \leqslant 9\delta (A_0 + \varepsilon_m), \\
\Delta_2 &= \delta (3A_5 + 3A_6 + 2A_7 + A_8) \leqslant 9\delta (A_0 + \varepsilon_m), \\
\Delta_3 &= \delta (3A_9 + 3A_{10} + 2A_{11} + A_{12}) \leqslant 9\delta (A_0 + \varepsilon_m), \\
\Delta_4 &= \delta (2A_{13} + 2A_{14} + A_{15}) \leqslant 5\delta (A_0 + \varepsilon_m);
\end{aligned}$$

складывая Δ_i, получаем

$$\Delta_\Sigma = \sum_{i=1}^{4} \Delta_i \leqslant 32\delta (A_0 + \varepsilon_m).$$

Сравнивая полученные оценки ошибок округления для двух способов вычисления одной и той же суммы, отмечаем, что при втором спосо-

бе суммирования оценка для ошибки имеет меньшее значение. Следовательно, способ суммирования по группам дает меньшую ошибку результата.

Решение задачи 17. Ошибка в программе состоит в том, что описание массива M и его использование разделены по модулям. Так как подпрограмма SUBSUM не содержит описания, указывающего на то, что M — это идентификатор массива, а информация, содержащаяся в главном модуле, недоступна, то в этой подпрограмме M воспринимается как имя подпрограммы-функции. Это возможно по той причине, что обращение к элементу массива и вызов функции синтаксически неразличимы (см. п. 3.1). В результате управление передается на первый элемент массива M, что и приводит к ошибке.

Решение задачи 18. Решение может иметь вид:

```
01      PROGRAM TASK
02      X=0.
03      CALL SUBR(X,X,X)
04      STOP
05      END
```

При решении задачи использовано имеющееся противоречие, основанное на неявном предположении, принятом в подпрограмме SUBR и заключающееся в том, что формальным параметрам X, Y, Z соответствуют разные переменные в вызывающей в программе. Адреса фактических параметров, передаваемых в подпрограммаму SUBR, совпадают. Присваивание X=1. фактически изменяет все три переменные, так как они имеют один и тот же адрес. При этом в слово с константой 0. засылается константа 1.. Это и фиксируется на печати — печатается на самом деле три раза одно и то же число.

Решение задачи 19. Одно из возможных решений задачи следующее:

```
01      PROGRAM TASK
02      EXTERNAL SUBR
03      CALL ZERO(SUBR)
04      STOP
05      END
C-----------------------------------------
06      SUBROUTINE ZERO(ARRAY)
07      DIMENSION ARRAY(30)
08      ARRAY(30)=0.
09      RETURN
10      END
```

В решении использовано то, что транслятор не производит контроля за соответствием классов фактического и формального параметров (см. п. 3.4.2). Подпрограмме ZERO передается адрес подпрограм-

мы SUBR, а в подпрограмме ZERO формальный параметр является массивом. На этом несоответствии и построено решение задачи.

Решение задачи 20. Все синтаксические ошибки в программе имеют один и тот же вид — длина идентификатора превышает 6 символов. Большинство трансляторов языка фортран (включая трансляторы фортран-Дубна на БЭСМ-6 и фортран IV на ЕС ЭВМ) игнорируют эту ошибку, делая лишь предупреждающее сообщение. При этом идентификаторы различаются по первым шести символам. Других ошибок в приведенной программе нет. В заблуждение может ввести «вольное» обращение с идентификаторами типа CALL, REAL, GO TO и др. Но, как уже говорилось в п. 3.1, в языке фортран нет выделенных символов (ключевых слов). Такая свобода в выборе имен идентификаторов нередко приводит к трудно обнаруживаемым ошибкам в программе. Результат выполнения программы:

```
**␣␣␣1
**␣␣␣2
**␣␣␣3
**␣␣␣4
**␣␣␣5
**␣␣␣␣5.␣␣␣9.␣␣␣2.␣␣␣5.␣␣␣5.␣␣␣5,
**␣␣␣␣5.␣␣␣5.
```

Решение задачи 21. Абсолютная величина ошибки результата при суммировании ста одинаковых положительных чисел $A = 0.6992896$ равна: для БЭСМ-6 $\Delta \simeq 3.3 \cdot 10^{-8}$, для ЕС ЭВМ $\Delta \simeq 4.1 \cdot 10^{-4}$. При вычислении произведения $S = 100 * A$ абсолютная величина ошибки результата равна: для БЭСМ-6 $\Delta \simeq 2{,}0 \cdot 10^{-10}$, для ЕС ЭВМ $\Delta \simeq 1.0 \cdot 10^{-5}$.

Для объяснения полученных результатов оценим ошибки вычислений. Ошибку округления при суммировании ста одинаковых чисел определим с помощью уже упоминавшейся формулы (см. п. 3.5). Подставив числовые значения, имеем:

$$\Delta \leqslant \delta \cdot A \cdot (99 + 99 + 98 + \ldots + 1) = \delta \cdot A \cdot 5049$$

или для БЭСМ-6 $\Delta \leqslant 3.2 \cdot 10^{-9}$, а для ЕС ЭВМ $\Delta \leqslant 3.4 \cdot 10^{-3}$.

Ошибку результата при вычислении произведения $S = 100 * A$ определим как сумму ошибки представления в памяти ЭВМ числа A и ошибки округления при умножении:

$$\Delta < (\delta_A + \delta) \cdot A.$$

Подставим значения для δ_A, δ, A. Получаем, что для БЭСМ-6 $\Delta < 2.5 \cdot 10^{-10}$, для ЕС ЭВМ $\Delta \leqslant 7.1 \cdot 10^{-5}$.

Таким образом, ошибка результата при вычислении выражения $SN = 100 * A$ значительно меньше (более чем на порядок), чем при вычислении результата выражения S, полученного суммированием ста одина-

ковых чисел A. Этот пример показывает, что при вычислениях надо стремиться сокращать число арифметических операций. Действительно, вычисляя значение выражения SN=100*A, имеем одну арифметическую операцию — операцию умножения, а при вычислении выражения S=A+A+...+A имеем 99 операций сложения.

Решение задачи 22. Для решения задачи достаточно подсчитать число символов во втором операторе программы. Он заканчивается в 73 позиции. Это означает, что последнее слагаемое 25 при синтаксическом анализе будет разбито на две части: 2 и 5. Цифра 2 будет отнесена к оператору присваивания, а цифра 5 (так как она находится в 73 позиции) будет воспринята как комментарий. Таким образом, последнее слагаемое в этом операторе есть 2, а не 25, что и вызывает расхождение результатов (25—2=23, 325—302=23).

Решение задачи 23. В приведенной программе обращение к подпрограмме QUEST оформлено как обращение к функции (операторы 02 и 05). В этом причина неправильного выполнения программы (см. п. 3.4.2).

При выполнении на БЭСМ-6 оператора Y=QUEST(X) адрес параметра X считывается на сумматор, а содержимое стека при этом не меняется, в то время как при выполнении оператора CALL QUEST(X) содержимое стека должно увеличиться на единицу. Таким образом, при выполнении программы QUEST, которая оформлена как подпрограмма, а не как функция, из стека будет взято одно число и воспринято как адрес параметра X. При этом поскольку адрес X был на сумматоре, то он будет взят правильно, но возврат из подпрограммы QUEST произойдет не по адресу возврата, а по адресу, который находился в предыдущем слове стека. Поэтому в данном случае произойдет потеря управления.

Решение задачи 24. Подпрограмма, удовлетворяющая условию задачи, может выглядеть так:

```
01      SUBROUTINE SUBR(ARRAY)
02      DIMENSION ARRAY(2)
03      J=0
04      ARRAY(J)=ARRAY(2)
05      RETURN
06      END
```

Решение задачи основано на двух предположениях: на отсутствии контроля при исполнении программы за выходом значения индекса за границы массива (операторы 03, 04 приведенной в решении подпрограммы) и последовательном расположении в памяти переменных общего блока памяти (оператор 02 условия задачи).

Решение задачи 25. При выполнении программы будет обнаружена ошибка, причиной которой является несоответствие количества фактических параметров в операторе вызова функции формаль-

ному описанию стандартной функции SIGN, имеющей два параметра. Результат вычисления функции SIGN будет неправильным, так как не задан второй фактический параметр функции. Кроме того, из-за затирания выделенного для параметров функции участка памяти произойдет возврат из функции по неправильному адресу, что вызовет потерю управления программой.

Решение задачи 26. Пример решения задачи:

```
01      PROGRAM TASK
02      CALL SUBR(J,J)
03      STOP
04      END
```

Решение задачи построено на зависимости фактических параметров при вызове подпрограммы SUBR. Поскольку переменная K увеличивается в цикле каждый раз на единицу, а элементу массива присваивается значение нуль, то это приведет к зацикливанию при выполнении операторов 03 и 04 подпрограммы. Возможен и другой вариант решения:

```
01      PROGRAM TASK
02      DIMENSION J(2)
03      CALL SUBR(J(2),J)
04      STOP
05      END
```

Обратите внимание на то, как теперь будет выполняться подпрограмма SUBR.

Решение задачи 27. Пример решения задачи:

```
01      PROGRAM TASK
02      CALL SUBR1
03      STOP
04      END
  C-----------------------------------------
05      SUBROUTINE SUBR1
06      EXTERNAL TASK
07      CALL ZERO(TASK)
08      RETURN
09      END
  C-----------------------------------------
10      SUBROUTINE ZERO(IARRAY)
11      DIMENSION IARRAY(10000)
12      IARRAY(513)=0
13      RETURN
14      END
```

Программа написана применительно к БЭСМ-6. Решение задачи основано на том, что во время исполнения программы соответствие классов фактического и формального параметров при обращении к подпрограммам не контролируется. Фактическим параметром при обращении к подпрограмме ZERO является PROGRAM — имя главного модуля (операторы 06, 07), а соответствующий ему формальный параметр является массивом (операторы 10, 11). Кроме того, в решении учитывается то, что загрузка программы в мониторной системе Дубна начинается с адреса 1000В. Поэтому в операторе 12 смещение равно 513= =1001В — это соответствует адресу 2000В.

Решение задачи 28. Во втором обращении к подпрограмме MULTM массив-матрица A используется одновременно как один из сомножителей и как результат произведения матриц. В языке фортран для передачи параметров используется метод, получивший название «по ссылке». Это значит, что в программу передается не сам объект (переменная, массив, подпрограмма) или его копия, а лишь адрес объекта. В нашем случае это приводит к тому, что в подпрограмме MULTM имена X и Z являются синонимами, т. е. обозначают один и тот же массив A, описанный в главном модуле. Таким образом, массив A одновременно используется как для выборки из него элементов, так и для записи в него новых значений.

Ниже приводится последовательность изменений массива A (он же X и Z при втором обращении в подпрограмме MULTM), с указанием значений переменных цикла, в программе MULTM после каждого завершения самого внутреннего цикла DO.

$$\begin{matrix} & K=1,\ J=1 & & K=1,\ J=2 \\ \begin{Vmatrix} 1 & 3 \\ 2 & 4 \end{Vmatrix} & \begin{Vmatrix} 23 & 3 \\ 2 & 4 \end{Vmatrix} & \begin{Vmatrix} 23 & 185 \\ 2 & 4 \end{Vmatrix} \\ & K=2,\ J=1 & & K=2,\ J=2 \\ & \begin{Vmatrix} 23 & 185 \\ 34 & 4 \end{Vmatrix} & \begin{Vmatrix} 23 & 185 \\ 34 & 270 \end{Vmatrix} \end{matrix}$$

На первом шаге вычисляется и заменяется элемент A(1,1). Для вычисления элемента A(1,2) необходимо значение A(1,1), но так как это значение уже изменено, то это и приводит к возникновению различия результатов вычисления.

Эта задача показывает, что, во-первых, всегда нужно знать условия, накладываемые на параметры, передаваемые в подпрограмму, и, во-вторых, к каким ошибкам в работающей программе может привести несоблюдение этих условий (каждая подпрограмма имеет определенный набор входных параметров, при которых она выполняется правильно и решает поставленную задачу).

Решение задачи 29. Достаточно передать в качестве формального параметра любое целое число, например

```
CALL SUBR(2)
```

При этом подпрограмме SUBR будет передан адрес целой константы 2, которая представляется как ненормализованное число [36]. Несоответствие типов фактического и формального параметров приведет к ошибке типа «деление на нуль», так как в операторе 02 производится деление на ненормализованное число, что и фиксируется машиной как деление на нуль.

Решение задачи 30. Причина различия результатов вычислений по разным формулам — ошибки округления. Точное значение выражения

$$S=(A+A+\ldots+A)*B$$

в котором число A складывается десять раз, равно 4.26049105845993. В значении S имеется 15 значащих цифр. На БЭСМ-6 и ЕС ЭВМ мантисса числа имеет меньшее число значащих цифр, поэтому вычисленные значения будут отличаться от точного.

Значение результата, вычисленного по первой формуле, имеет погрешность $\Delta\leqslant 1.0\cdot 10^{-11}$ на БЭСМ-6 и $\Delta\leqslant 7.0\cdot 10^{-6}$ на ЕС ЭВМ.

Результат вычисления по второй формуле имеет погрешность $\Delta\leqslant 9.93\cdot 10^{-12}$ на БЭСМ-6 и $\Delta\leqslant 2.0\cdot 10^{-6}$ на ЕС ЭВМ.

По обеим формулам погрешность значений, вычисленных на ЕС ЭВМ, больше, чем у значений, вычисленных на БЭСМ-6. Значение, вычисленное по второй формуле, имеет меньшую погрешность, чем значение, вычисленное по первой формуле.

Оценим ошибку вычисления выражения S. Для первого способа вычисления:

$$\begin{aligned}\Delta &= \Delta' + \Delta_B + \delta B,\\ \Delta' &\leqslant \delta A\,(9+9+8+\ldots+1) = 54A\delta,\\ \Delta &\leqslant 54A\delta + \Delta_B + \delta B.\end{aligned}$$

Для значения результата, вычисленного на БЭСМ-6, имеем $\Delta\leqslant 2.8\cdot 10^{-11}$, для значения, вычисленного на ЕС ЭВМ имеем $\Delta\leqslant 2.7\cdot 10^{-6}$.

Для второго способа вычисления:

$$\Delta = \Delta_{AB} + \delta_B A + \delta C.$$

Ошибка значения, вычисленного на БЭСМ-6: $\Delta\leqslant 2.8\cdot 10^{-11}$, ошибка значения, вычисленного на ЕС ЭВМ: $\Delta\leqslant 2.7\cdot 10^{-6}$.

Таким образом, оценка ошибки вычисления выражения S по обоим способам подтверждает, что результат вычисления выражения S по второму способу имеет меньшую погрешность.

Этот пример еще раз подтверждает мысль о том, что уменьшение числа выполняемых арифметических операций, как правило, приводит к уменьшению ошибки результата. Это следует помнить при разработке вычислительных алгоритмов и их реализации в программах.

Решение задачи 31. При решении задачи следует иметь в виду, что целые числа представляются в памяти ЭВМ в ненормализованном виде. Например, целые числа в памяти БЭСМ-6 представляются в виде

1 — 6400000000000001
10 — 6400000000000012
100 — 6400000000000144

и т. д., а действительные числа — в виде

1.0 — 4050000000000000
10.0 — 4212000000000000
100.0 — 4354400000000000

и т. д.

1. При вызове подпрограммы SUBR оператором CALL SUBR(0) имеем J = 0, что на БЭСМ-6 приводит к зацикливанию в операторе 03 (см. п. 3.3.1 и приложение 1).

2. При вызове подпрограммы SUBR оператором CALL SUBR(4) управление будет передано на команду, следующую за оператором GO TO (1, 2, 3), J т. е. на оператор 04.

3. При вызове подпрограммы SUBR оператором CALL SUBR(1.0) имеет место несоответствие типов формального и фактического параметров. Так как формальный параметр имеет целый тип, а при вызове подпрограммы фактический параметр имеет действительный тип, то в этом случае младшие 15 разрядов переменной J — нули. В результате в программе произойдет зацикливание, так же как в первом случае.

4. При вызове подпрограммы SUBR оператором CALL SUBR(100) управление в программе будет передано в слово памяти, расположенное примерно через сто слов после тела оператора GO TO (1, 2, 3), J. Программа в этом случае теряет управление. Результат дальнейшего выполнения программы непредсказуем.

Решение задачи 32. Пример решения задачи:

```
01      PROGRAM TASK
02      EXTERNAL SUBR
03      CALL SET(X, SUBR, 10)
04      STOP
05      END
  C .....
06      SUBROUTINE SET(VAR, ARRAY, N)
07      DIMENSION ARRAY(1)
08      VAR=ARRAY(N)
09      RETURN
10      END
```

При решении задачи использован эффект несоответствия классов фактического и формального параметров в операторах 03 и 06—07. В операторе 03 вместо адреса массива передается адрес входа в подпрограмму

SUBR. В подпрограмме десятый элемент массива (при таком вызове подпрограммы) будет переслан в переменную X.

Решение задачи 33. Практически все трансляторы языка фортран не вставляют в готовую программу команды проверки условий, которым должны удовлетворять параметры оператора цикла. Хотя, если эти параметры являются константами, а не переменными, как в нашем случае, транслятор при трансляции проверяет выполнение этих условий.

В первом случае будет выполняться оператор цикла

```
DO 1 K=1, 7, 2
```

Это приводит к следующему результату (на БЭСМ-6 и ЕС ЭВМ результат одинаковый):

```
**␣1␣3␣5␣7
```

Во втором случае выполняется оператор

```
DO 1 K=8, 3, -1
```

В случае работы программы на БЭСМ-6 никакой печати не будет, а на ЕС ЭВМ будет напечатано только одно число 8. Различие в поведении программы на разных ЭВМ вызвано тем, что транслятор фортран-Дубна на БЭСМ-6 вставляет команды проверки условия завершения цикла перед телом цикла, а транслятор на ЕС ЭВМ — после тела цикла. Поэтому на ЕС ЭВМ тело цикла всегда выполняется, хотя бы один раз, что и имело место в нашем случае. Хотя в идейном плане выполнение оператора цикла DO с такими параметрами, как во втором случае, не приводит к каким-либо неприятностям, исторически сложилось так, что шаг цикла должен быть положительным. И несмотря на то что циклы с отрицательным шагом часто оказываются очень полезными и удобными, их использование в большинстве реализаций языка фортран запрещено (хотя в некоторых, например в форексе [3], это допускается).

В третьем случае мы имеем оператор

```
DO 1 K=9, 2, 1
```

Как и в предыдущем случае, на БЭСМ-6 тело цикла не будет выполняться, на ЕС ЭВМ будет напечатано одно число 9.

В четвертом случае оператор цикла будет

```
DO 1 K=2, 6, 0
```

Выполнение цикла приведет к зацикливанию. Так как шаг цикла равен нулю, то тело цикла будет выполняться бесконечное число раз со значением K=2, печатая каждый раз число 2 (на самом деле цикл оборвет-

ся в момент заполнения буфера печати либо когда окончится выделенное время для решения задачи и операционная система прервет выполнение задачи).

В последнем, пятом случае программа напечатает сначала два числа 4 и 2. Дальнейшее поведение программы непредсказуемо, так как она начнет распечатывать память, не принадлежащую массиву M.

Решение задачи 34. Решением может быть следующая программа:

```
01        PROGRAM TASK
02        IRES=IFACTR(5)
03        PRINT 1, IRES
04      1 FORMAT('␣ФАКТОРИАЛ=',I4)
05        END
      C ------------------------------------
06        FUNCTION IFACTR(N)
07        EXTERNAL SUBFAC
08        M=1
09        CALL SUBFAC(SUBFAC, N, M)
10        IFACTR=M
11        RETURN
12        END
      C ------------------------------------
13        SUBROUTINE SUBFAC(SUBR, N, M)
14        IF(N .EQ. 1) RETURN
15        M=M*N
16        N=N-1
17        CALL SUBR(SUBR, N, M)
18        RETURN
19        END
**␣ФАКТОРИАЛ=␣120
```

Подпрограмма-функция IFACTR представляет пример организации рекурсии на фортране (для языка фортран-Дубна на БЭСМ-6). Рассмотрите работу программы и прежде, чем читать дальше, ответьте на следующие вопросы:

1. Какое свойство реализации языка фортран позволяет писать рекурсивные программы?

2. Зачем введена дополнительная подпрограмма SUBFAC?

3. Почему в качестве фактического параметра подпрограммы SUBFAC передается имя этой же подпрограммы?

4. Какая ошибка содержится в программе, но не влияет на результат?

Решение задачи 35. В качестве решения можно предложить следующую программу:

```
01        PROGRAM TASK
```

```
02      COMMON /BLOCK/ J
03      J=1
04      CALL SUBR(J)
05      STOP
06      END
```

Решение задачи основано на том, что адрес элемента общего блока памяти совпадает с адресом фактического параметра подпрограммы SUBR. Обнуление параметра L в операторе 04 подпрограммы фактически изменяет переменную общего блока, которая имеет идентификатор K.

Решение задачи 36. Пример решения задачи:

```
01      PROGRAM TASK
02      CALL SUBR1
03      STOP
04      END
  C -----------------------------------------
05      SUBROUTINE SUBR1
06      EXTERNAL TASK
07      CALL SUBR2(TASK)
08      RETURN
09      END
  C -----------------------------------------
10      SUBROUTINE SUBR2(ARRAY)
11      DIMENSION ARRAY(1024)
12      CALL SUBRGO(ARRAY(513))
13      RETURN
14      END
  C -----------------------------------------
15      SUBROUTINE SUBRGO(SUBR)
16      CALL SUBR
17      RETURN
18      END
```

При решении этой задачи дважды использовано несоответствие классов фактического и формального параметров: в операторах 06—07 и 10—11, 11—12 и 15—16. Кроме того, учитывается то, что в операционном окружении БЭСМ-6 программа загружается в память начиная с адреса 1000B (оператор 01). При работе с другой ОС на БЭСМ-6 или на другой ЭВМ этот адрес будет иным. В таком случае необходимо изменить оператор 12. В подпрограмму SUBRGO будет передан адрес 513-го слова массива ARRAY. Адрес начала массива ARRAY равен 512, а адрес 513-го элемента этого массива — 1024 или 2000B. Таким образом, адрес фактического параметра SUBR в подпрограмме SUBRGO есть 2000B, оператор CALL SUBR передает управление по этому адресу.

Имеется еще одно, более простое решение задачи, но которое справедливо не для всех реализаций языка. Ниже приведена программа на языке фортран-Дубна, реализующая это решение:

```
01        PROGRAM A
02        M=2000B
03        GO TO M, (1, 2)
04      1 CONTINUE
05      2 CONTINUE
06        STOP
07        END
```

Переменной M, которая используется в операторе перехода по предписанию, присваивается (арифметическим оператором присваивания) адрес перехода. Если полученное значение может быть использовано в операторе перехода по предписанию (представление восьмеричных чисел совпадает с представлением значений, присваиваемых переменным оператором ASSIGN) и в процессе выполнения программ отсутствует контроль значений переменной M (в операторе 03), то оператор перехода по предписанию (оператор 03) передаст управление по адресу 2000B.

Решение задачи 37. Значения выражений, вычисленные двумя способами, совпадают на БЭСМ-6 в 11-ти значащих цифрах, а на ЕС ЭВМ — в шести. Сравнивая между собой значения, вычисленные на БЭСМ-6 и ЕС ЭВМ, отметим, что они совпадают для первого способа вычислений в шести значащих цифрах и для второго — в семи. Для заданных значений A, B и C выражение $S=(A-B)/C$ с точностью до 15 знаков равно $S=0.0460139605936338$. Сравнивая это значение выражения S со значениями, вычисленными на ЭВМ, имеем погрешность при первом способе вычислений $\Delta \simeq 2.9563 \cdot 10^{-9}$ на БЭСМ-6 и $\Delta \simeq 5.94 \cdot 10^{-10}$ на ЕС ЭВМ, а при втором способе вычислений $\Delta \simeq 2.9564 \cdot 10^{-9}$ на БЭСМ-6 и $\Delta \simeq 1.0594 \cdot 10^{-8}$ на ЕС ЭВМ. Таким образом, результаты вычислений по первому способу имеют меньшую ошибку, чем по второму. Ошибка не велика, но надо иметь в виду, что при вычислениях с другими данными она может быть чрезмерно большой. Причина различия полученных результатов в конечности разрядной сетки ЭВМ.

Покажем теперь, что при вычислениях по первому способу всегда получается меньшая ошибка. Для этого оценим максимально возможные ошибки вычислений по обоим способам. Числа A, B и C представимы в машине не точно, а с некоторой ошибкой. Как уже упоминалось (см. п. 3.5), ошибки представления любых чисел в памяти ЭВМ связаны с конечностью разрядной сетки ЭВМ. Эта ошибка не превосходит значения последнего разряда мантиссы и равна $\Delta \leqslant 2^{-40} \simeq 9 \cdot 10^{-13}$ для БЭСМ-6 и $\Delta \leqslant 16^{-5} \simeq 9.5 \cdot 10^{-7}$ для ЕС ЭВМ.

При вычитании (A — B) получаем число D, которое имеет ошибку, складывающуюся из ошибки представления чисел A и B в памяти ЭВМ, ошибки выполнения арифметической операции вычитания и ошибки округления. Ошибка при вычитании двух чисел не превосходит суммы модулей ошибок представления в памяти ЭВМ этих чисел:

$$\Delta_D \leqslant |\Delta_A| + |\Delta_B|.$$

Ошибка при делении может быть определена с помощью формулы [30] (см. п. 3.5) $\Delta_{D/C} \leqslant \frac{1}{c}\delta_A + \frac{1}{c^2}\delta_C$. Ошибка округления зависит от способа округления, реализованного в ЭВМ. В большинстве ЭВМ используется либо способ, который называется *отбрасыванием младших разрядов мантиссы*, либо так называемое *симметричное округление* [4, 12, 30]. Максимально возможные значения ошибок округления для ЭВМ, работающих с двоичными числами, составляют $2 \cdot 2^{-s}$ для способа отбрасывания и 2^{-s} для способа симметричного округления. Для ЭВМ, работающих с шестнадцатеричными числами (ЕС ЭВМ), максимальные ошибки округления составляют $16 \cdot 16^{-e}$ для способа отбрасывания и $8 \cdot 16^{-e}$ для способа симметричного округления. Здесь s — число разрядов в мантиссе, e — число значащих цифр мантиссы.

Суммарная ошибка результата в случае первого способа вычисления $\Delta \leqslant \Delta_D + \Delta_{D/C} + \Delta_\delta$. Подставив в эту формулу значения ошибок, получаем ошибку результата при вычислениях на ЭВМ БЭСМ-6 $\Delta \leqslant 3.6 \cdot 10^{-12}$ и на ЕС ЭВМ $\Delta \leqslant 2.9 \cdot 10^{-6}$.

Суммарная ошибка в случае второго способа вычисления

$$\Delta \leqslant \Delta_{A/C} + \Delta_{\delta_1} + \Delta_{B/C} + \Delta_{\delta_2} + \Delta_{\delta_3}.$$

Здесь $\Delta_{A/C}$, $\Delta_{B/C}$ — ошибки при выполнении арифметической операции деления, Δ_{δ_1}, Δ_{δ_2}, Δ_{δ_3} — ошибки округления после выполнения операций деления и вычитания. Подставив значения ошибок, имеем при вычислениях на БЭСМ-6 $\Delta \leqslant 6.3 \cdot 10^{-12}$ и при вычислениях на ЕС ЭВМ $\Delta \leqslant 4.0 \cdot 10^{-6}$.

Из сравнения максимальных значений оценок ошибок, полученных в результате вычислений заданных выражений, для обоих способов следует, что вычисления по первому способу дают меньшую ошибку.

Р е ш е н и е з а д а ч и 38. Выполнение программы на БЭСМ-6 закончится нормально. Однако в ней содержатся две ошибки:

1) несоответствие количества фактических и формальных параметров,

2) вызов функции как подпрограммы. Наложение двух указанных ошибок не вызывает аварийной ситуации. Как указано в приложении 1, при обращении к подпрограмме указатель стека сдвигается на величину, равную количеству фактических параметров, а при выборке формальных параметров функции стек сдвинется на количество пара-

метров функции в другую сторону. Но значение функции сохраняется в стеке и, вследствие этого, значение указателя стека будет правильным. Поясним сказанное. Подсчитаем заполненность стека. Оператор CALL QUEST (X) добавит к стеку одно слово (адрес параметра X). При выполнении функции стек сначала уменьшится на два слова (выборка адресов двух параметров X и Y), а затем перед возвратом увеличится на одно слово после засылки в стек значения функции. Таким образом, после выполнения оператора 02 содержимое стека будет таким же, как и перед его выполнением. Изменится лишь содержимое сумматора. Поэтому результат выполнения этой программы неправильный, но выполнение программы закончится без аварийной ситуации.

Заметим, что в результате ошибки будет изменено содержимое вершины стека. В большинстве ситуаций это может повлиять на выполнение программы. Например, может неправильно произойти возврат из подпрограммы-функции.

Решение задачи 39. 1. В операторе 08 в идентификаторе IA1 замените 1 на I. В этом случае получится несоответствие классов фактического и формального параметров. В операторе 11 произойдет обращение к функции IA1, которая на самом деле отсутствует.

2. В операторе 11 замените в идентификаторе IA1 1 на I. В этом случае компоновщик не найдет требуемую функцию IAI.

3. В операторе 07 замените 1 на I. Тогда подсчет отрицательных элементов произойдет во внутреннем массиве подпрограммы.

4. В операторе 04 замените 1 на I. Тогда подсчет отрицательных элементов произойдет не в той области памяти, которая определена массивом IA1, а начиная с простой переменной IAI, определенной в программе TASK.

Решение задачи 40. Ошибка вызвана различием в описании переменных X и Y, входящих в общий блок памяти. В главном модуле они описаны как действительные переменные с двойной точностью, а в подпрограмме SUB — как обычные переменные типа REAL (неявное определение типа по имени идентификатора). Под переменную двойной точности отводится два слова памяти. Поэтому переменная X из SUB будет поставлена в соответствие первому слову памяти для переменной X из главного модуля, а переменная Y из SUB соответствует второму слову (рис. 2). Учитывая способ представления чисел с двойной точностью (см. п. 3.5 и [36]), легко понять получившийся результат.

Решение задачи 41. Ошибка заключается в отсутствии начального присваивания значения переменной N. В реализации языка

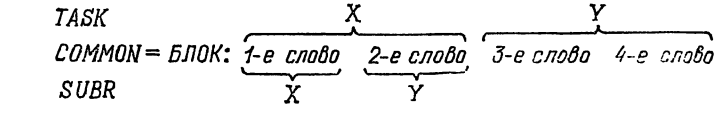

Рис. 2

фортран на БЭСМ-6 всем переменным при загрузке программы присваивается нулевое значение, если явно не указано присваиваемое значение в операторе DATA. Чтобы исправить ошибку, необходимо перед телом цикла вставить оператор N=1, либо в разделе описателей вставить оператор DATA N/1/. При выполнении приведенной программы на других ЭВМ, где не производится присваивания начальных значений по умолчанию, результат может быть совсем другим. Например, аппаратурой ЭВМ может быть зафиксирована ошибка при выполнении операции умножения в операторе N=N*2.

Решение задачи 42. Несоответствие количества фактических и формальных параметров при вызове подпрограммы (операторы 03 и 07) приведет при выполнении программы на БЭСМ-6 к ошибке, которая будет зафиксирована как «останов по считыванию» (см. задачи 9, 23, 25, 38, 49).

В данном случае у подпрограммы SUBR два формальных параметра, а обращение к ней происходит с одним параметром. Оператор CALL SUBR (X) выполняется так: содержимое стека увеличивается на одно слово (засылка в стек адреса фактического параметра X), затем уменьшается на два слова при выполнении оператора SUBROUTINE SUBR (X, Y), в котором происходит выборка адресов двух параметров (см. приложения 1 и 3). На каждом шаге цикла содержимое стека уменьшается на одно слово, т. е. будет уменьшаться содержимое индекс-регистра И15. В БЭСМ-6 есть специальный регистр — «регистр записи/считывания» (ЗПСЧ). ЗПСЧ — это ограничитель записи или считывания. Когда значение индекс-регистра И15 совпадает со значением регистра ЗПСЧ, то происходит аварийная ситуация типа «останов по считыванию». Перед выполнением программы начальное состояние регистра И15 больше значения ЗПСЧ на 3—4 единицы. Следовательно, при выполнении программы, приведенной в условии задачи, на четвертом шаге произойдет аварийная ситуация по считыванию.

Решение задачи 43. Решение этой задачи аналогично решению задачи 37. Ошибка результата произведения складывается из суммы ошибок сомножителей и ошибки округления после выполнения арифметической операции умножения.

Для первого способа вычисления максимальная оценка ошибки результата

$$\Delta \leqslant \Delta_D + \Delta_A + \Delta_\delta .$$

Как и в задаче 37, Δ_D — ошибка результата вычитания чисел. В данном случае при вычитании чисел B и C Δ_A — ошибка представления числа A в памяти ЭВМ и Δ_δ — ошибки округления после выполнения арифметических операций. Подставив значения ошибок, имеем при вычислениях на БЭСМ-6 $\Delta \leqslant 4.5 \cdot 10^{-12}$ и при вычислениях на ЕС ЭВМ $\Delta \leqslant 2.9 \cdot 10^{-6}$.

Для второго способа вычисления максимальная оценка ошибки

$$\Delta \leqslant \Delta_{BA} + \Delta_{CA} + \Delta_{\delta}.$$

Здесь Δ_{BA}, Δ_{CA} — ошибки при выполнении операций умножения, Δ_{δ} — ошибки округления после выполнения арифметических операций. Подставив значения ошибок, получаем $\Delta \leqslant 5.4 \cdot 10^{-12}$ на БЭСМ-6 и $\Delta \leqslant 4.0 \cdot 10^{-6}$ на ЕС ЭВМ.

Таким образом, оценка максимальных ошибок при вычислении заданными способами подтверждает, что вычисления по первому способу дают меньшую ошибку. Хотя разница в значениях приведенных здесь ошибок вычислений по обоим способам невелика, всегда есть опасность, что при многократных вычислениях подобных выражений может накапливаться весьма значительная ошибка. В конечном итоге могут быть получены совершенно неправильные результаты. Это следует учитывать при организации вычислений в программах.

Решение задачи 44. Программу, которая удовлетворяет условию задачи, можно написать так:

```
01      PROGRAM TASK
02      COMMON /BLOCK/ J
03      CALL SUBR(J)
04      STOP
05      END
```

Решение построено на зависимости фактического параметра и переменной общего блока памяти. Полезно провести сравнение с задачей 26. Подумайте, почему второе решение задачи 26 в данном случае неприменимо.

Решение задачи 45. Пример решения задачи:

```
01      PROGRAM TASK
02      EXTERNAL SUBR
03      CALL SUBR1(SUBR)
04      STOP
05      END
   C----------------------------------------
06      SUBROUTINE SUBR1(ARRAY)
07      DIMENSION ARRAY(65)
08      CALL SUBRGO(ARRAY(65))
09      RETURN
10      END
   C----------------------------------------
11      SUBROUTINE SUBRGO(SUBR)
12      CALL SUBR
13      RETURN
14      END
```

При решении этой задачи дважды использовано несоответствие классов фактических и формальных параметров: в операторах 03 и 06 и в операторах 08 и 11. Решение задачи аналогично решению задачи 36.

Решение задачи 46. Ключом к решению является оператор

```
1 CONTINUE
```

Этот пример опровергает представление об операторе CONTINUE как о пустом операторе. В данном случае оператор CONTINUE является составной частью цикла — оператором, завершающем цикл. В языке фортран-Дубна завершающий оператор при трансляции дополняется командами перехода к следующему шагу цикла. В приведенном примере оператор CONTINUE завершает одновременно два вложенных цикла. Нужно еще учесть тот факт, что при загрузке программы на БЭСМ-6 переменным присваивается нулевое значение. Поэтому перед выполнением программы параметр внутреннего цикла N=0.

В данном случае оператор CONTINUE эквивалентен следующей последовательности операторов:

```
*1      N=N+1
        GO TO *1
*2      K=K+1
        GO TO *3
*4      Пустой оператор
```

Со звездочки начинаются метки, «вставляемые» транслятором. Здесь *1 и *3 — метки начала внутреннего и внешнего циклов соответственно. Метки *2 и *4 используются для выхода соответственно из внутреннего и внешнего циклов.

При переходе на метку 1 после выполнения оператора IF в случае $M(K,1) \leqslant 0$ N присваивается значение $N=0+1=1$ и управление затем передается на начало внутреннего цикла. Таким образом, обойти внутренний цикл не удается и программа суммирует элементы всех строк матрицы. Правильное решение состоит в использовании для вложенного цикла своего завершающего оператора. Выполнение этой программы на ЕС ЭВМ также приводит к ошибке.

Решение задачи 47. Результаты вычислений на БЭСМ-6 приведены ниже:

```
X=A*B/C    9.999999999989+14
Y=A*(B/C)  1.000000000000+15
Z=A/C*B    переполнение
```

Как видно, во всех трех случаях получились различные результаты. Переполнение в последнем случае вызвано тем, что промежуточный результат операции деления

```
A/C=10**15/10**(-15)=10**30
```

превосходит максимальное число, представимое в БЭСМ-6, равное приблизительно 10^{19}. Обратите внимание на различие результата в первом и во втором случаях.

Эта задача еще раз (см. п. 3.5) иллюстрирует тот факт, что порядок выполнения операций может иметь существенное значение при вычислениях на ЭВМ. Следует отметить, что на ЕС ЭВМ в третьем случае не произойдет переполнения, так как диапазон представления чисел здесь значительно больше по сравнению с БЭСМ-6. При выполнении программы на ЕС ЭВМ для получения аналогичной ситуации необходимо взять иные значения переменных.

Решение задачи 48. Решением может быть следующая программа:

```
01        PROGRAM TASK
02        INTEGER M(12␣000)
03        EXTERNAL COPY
04        CALL COPY(COPY,M,M)
05        END
      C-----------------------------------------
06        SUBROUTINE COPY(M,L,SUB)
07        INTEGER M(200),L(200)
08        PRINT 2
09        DO 1 K=1,200
10      1 L(K)=M(K)
11        CALL SUB(L,L(201),L(201))
12        RETURN
13      2 FORMAT(' SUBROUTINE COPY')
14        END
```

В операторах 09—10 происходит копирование массива M в массив L. Оператором 11 выполняется передача управления на скопированную программу.

Решение задачи 49. Причина ошибки состоит в том, что QUEST является функцией, а обращение к ней было сделано, как к подпрограмме. Это привело к затиранию содержимого участка памяти, выделенного для формальных параметров (см. п. 3.4.2 и приложение 1). Обращение и возврат из функции оформляется иначе, чем к подпрограмме. При выполнении оператора CALL QUEST(X, Y) адреса параметров X и Y будут записаны в стек, а затем произойдет передача управления функции QUEST. При выполнении функции производится считывание адресов фактических параметров из стека и засылка их в рабочую память функции QUEST, заполнение в стеке адреса возврата из QUEST, запоминание регистров, вычисление функции QUEST, восстановление использованных регистров, считывание из стека адреса возврата из QUEST, засылка в стек значения функции QUEST из слова *VALUE и выполнение возврата в модуль TASK (см. приложение 1). От-

личие в оформлении вызова подпрограммы и функции вызовет затирание стека, и в какой-то момент выполнения программы произойдет потеря управления.

Решение задачи 50. Пример программы, решающей задачу:

```
01      PROGRAM TASK
02      CALL SUBR1(X)
03      STOP
04      END
   C-----------------------------------------
05      SUBROUTINE SUBR1(X)
06      EXTERNAL TASK
07      CALL SET(X,TASK,513)
08      RETURN
09      END
   C-----------------------------------------
10      SUBROUTINE SET(X,ARRAY,N)
11      DIMENSION ARRAY(1)
12      VAR=ARRAY(N)
13      RETURN
14      END
```

При решении задачи использован эффект несоответствия классов фактических и формальных параметров в операторах 07 и 10. Решение аналогично решению задачи 19.

Решение задачи 51. Причиной ошибки явилась опечатка, допущенная в операторе 08. При описании массива IO произошла замена буквы «О» на цифру«0» (нуль). В результате этого массив IO, являющийся формальным параметром подпрограммы SUBR, оказался неописанным и появился другой массив I0. В силу того, что, как уже говорилось в п. 3.1, обращение к функции и к элементу массива синтаксически неразличимы в фортране, то массив идентифицируется только при наличии явного описания соответствующего идентификатора как массива. Это привело к тому, что идентификатор IO воспринимается транслятором как имя функции, так как в 11-м операторе 1 I=I+IO(J) он встретился в арифметическом выражении и при этом не описан как массив. При выполнении этого оператора произойдет передача управления на массив IO. Это приводит к непредсказуемым результатам.

Решение задачи 52. Несоответствие количества фактических и формальных параметров при вызове подпрограммы (операторы 03, 07) приведет к затиранию памяти и в конечном счете к ошибке (решение справедливо для БЭСМ-6, см. решения задач 9, 23, 25, 38, 42, 49). Пояснение: при выполнении оператора CALL SUBR(X, Y) параметры X и Y засылаются в стек, после этого передается управление на подпрограмму SUBR. Выполнение подпрограммы начинается со считывания адресов параметров из стека и пересылки их в рабочую

память подпрограммы SUBR. Поскольку в заголовке программы объявлен только один параметр, то из стека будет выбран адрес только одного фактического параметра, который и будет адресом формального параметра X. Это и приведет к ошибке при выполнении программы.

Решение задачи 53. Для того чтобы понять такое поведение программы, необходимо обратить внимание на EQUIVALENCE-оператор в главном модуле. Результатом действия этого оператора является перекрытие в памяти массивов B и C: последние десять элементов массива B является одновременно и элементами массива C (рис. 3). Таким образом, массивы B и C становятся связанными — изменение одного может привести к изменению другого. Этот факт совершенно не

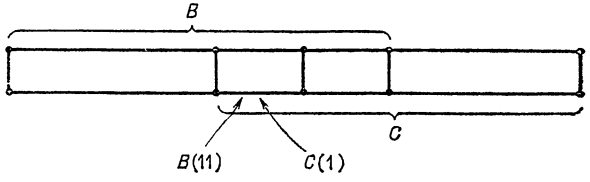

Рис. 3

учитывается в подпрограмме COPY, которая написана в предположении, что массивы M и N не пересекаются. Поэтому при копировании элемента B(1) в C(1) происходит затирание элемента B(11) массива B и т. д.

Решение задачи 54. Результаты вычислений дают для обеих ЭВМ совпадение в четвертой, пятой значащих цифрах. Суммирование убывающей геометрической прогрессии в обратном порядке дает меньшую погрешность, чем суммирование в прямом порядке. Это очевидно, так как суммирование чисел в порядке возрастания дает меньшую ошибку округления. Наиболее точное значение (ошибка меньше $5.0 \cdot 10^{-14}$) получено при вычислении суммы убывающей геометрической прогрессии по формуле для вычисления суммы убывающей геометрической прогрессии. Объяснение полученных результатов аналогично объяснению, приведенному в задачах 3 и 6.

Решение задачи 55. Пример решения задачи:

```
01      SUBROUTINE SUBR(J)
02      J=8
03      RETURN
04      END
```

Решение задачи аналогично решению задачи 7. Разница состоит лишь в том, что в данном случае целая константа 4 из главного модуля TASK затирается целой константой 8. Это приводит к увеличению числа повторений цикла в модуле TASK. Приведенное решение имеет место при условии, сформулированном в решении задачи 7.

Решение задачи 56. Пример решения задачи:

```
01      PROGRAM TASK
02      COMMON /BLOCK/ ARRAY(100)
03      CALL SUBRGO(ARRAY(10))
04      STOP
05      END
C-----------------------------------------
06      SUBROUTINE SUBRGO(SUBR)
07      CALL SUBR
08      RETURN
09      END
```

При решении этой задачи использован эффект несоответствия классов фактического и формального параметров в операторах 03 и 06. Решение аналогично решению задач 36, 45.

Решение задачи 57.

При решении задачи следует помнить, что в операторе DATA не производится преобразование типов засылаемых констант. Так, действительной переменной *B*, которая записана в операторе DATA B/15000/ будет присвоено значение целой константы 15000. В восьмеричной записи переменная B будет иметь вид:

B=6400000000035230B

Согласно правилам выполнения арифметической операции деления, делитель должен быть нормализованным числом. В противном случае происходит прерывание выполнения программы по ошибке «деление на нуль» (см. приложение 2). В данном случае делитель B — ненормализованое число, поэтому при выполнении оператора 06 программы имеет место ошибка «деление на нуль».

ГЛАВА 5

О СТИЛЕ ПРОГРАММИРОВАНИЯ

Форме дай щедрую дань
Временем: важен в поэме
Стиль, отвечающий теме.
Стих, как монету, чекань
Строго, отчетливо, честно,
Правилу следуй упорно:
Чтобы словам было тесно,
Мыслям — просторно.

Н. Некрасов

Рассмотренные до сих пор примеры программ и содержащихся в них ошибок носили модельный характер — программы составлены таким образом, чтобы наиболее ярко проявить характер ошибки. Ситуации, с которыми сталкивается программист в реальной практике, несравненно сложнее и запутанней. Приведенные в этой книге задачи показывают лишь некоторую часть всех проблем, возникающих при отладке программ. Рассмотрим теперь некоторые подходы к решению этих проблем. Начнем обсуждение со стиля программирования.

Трудно дать исчерпывающее определение понятия «стиль программирования». Попытаемся охарактеризовать это понятие, рассмотрев его с различных точек зрения. Главный тезис состоит в том, что *стиль программирования — это стиль мышления, проявляющийся в умении отображать алгоритм решения задачи на конкретный язык программирования.*

Не существует совокупности правил написания программ, следуя которым вы сможете создавать качественные и не содержащие ошибок программы. Стиль программирования не сводится к хорошему знанию конкретного языка программирования, знанию его возможностей и правил записи программ, хотя он все это и предполагает. Скорее эти знания помогут отличить программу, написанную в хорошем стиле, от программы, написанной в плохом стиле. Многие достаточно хорошо знают какой-либо иностранный язык, чтобы без особых затруднений читать специальную или художественную литературу на этом языке,

Но лишь немногие могут с такой же легкостью написать на иностранном языке даже небольшую статью. Таким образом, от знания языка до владения языком лежит «дистанция огромного размера». Но даже овладев языком программирования в совершенстве, мы лишь незначительно приблизимся к овладению стилем программирования. Чего же не хватает еще? Каждый из нас владеет русским языком, усвоив школьные курсы морфологии, орфографии, синтаксиса, пунктуации и т. п., но это еще не гарантирует, что мы можем писать так же, как А. С. Пушкин, Ф. М. Достоевский, Л. Н. Толстой или даже как менее известные писатели.

Программирование — это прежде всего форма творческого процесса [10, 44], в котором можно выделить два основных этапа:

— построение воображаемой модели будущей программы и

— воплощение этой модели в конкретную программу с использованием различных языковых средств.

Модель программы, создаваемая на первом этапе, имеет общий характер, выражая лишь идею решения задачи. На втором этапе происходит полная конкретизация модели. При этом могут быть обнаружены ошибки и неувязки в принятой модели, что потребует частичного или полного пересмотра модели. Именно этот переход от мысли к слову и вызывает основные трудности при программировании. «Связь между пониманием и написанием остается для программирования такой же проблематичной, какой она всегда была для любой иной формы письменного изложения» [10]. «Мысль всегда представляет собой нечто целое, значительно большее по своему протяжению и объему, чем отдельное слово... Процесс перехода от мысли к речи представляет собой чрезвычайно сложный процесс расчленения мысли и ее воссоздания в словах» (*Л. С. Выготский*).

Очень часто корень зла кроется в неумении построить адекватную и ясную модель решаемой задачи, отделить главное от второстепенного, в неспособности учесть все факты, принять правильное решение, когда имеется большой набор альтернатив, короче говоря — просто в неумении четко мыслить [2, 5]. Модель получается поверхностной, а ее анализ — неглубоким и фрагментарным, не охватывающим всю задачу в целом. Именно плохо проработанная модель и порождает многие проблемы при программировании. Следовательно, главное, чему следует уделять внимание,— это развитию мышления, его стиля, в области решения задач с помощью ЭВМ.

К сожалению, взгляд на стиль программирования как на стиль мышления еще далеко не всеми и не до конца осознан. Очень часто стиль программирования сводится к технологии программирования. За последние годы мы пережили несколько вспышек увлечения модными методологиями и технологиями программирования, такими как структурное программирование, разработка программ сверху вниз и и т. п. [2, 3, 10, 13, 17, 26]. Но в главном все они ориентируются на

внешние факторы, характеризующие программу. Кстати сказать, жертвой структурного программирования чуть было не стал фортран с его крайне неструктурными операторами. Но очевидно, что плохо разработанная программа, записанная по правилам структурного программирования, так и останется плохой программой. В этом плане переход от фортрана к таким языкам структурного программирования, как паскаль и ада, мало что даст.

Мы привыкли к понятию «математическая культура» и хорошо понимаем, что за этим стоит. Точно так же имеет смысл говорить о программистской культуре, лежащей в основе стиля программирования.

Эта книга не претендует на полное изложение того, что называется хорошим стилем программирования, да это, пожалуй, и невозможно сделать вообще. Более того, мы не хотим, чтобы у читателя создалось впечатление, что безукоризненное следование рекомендациям, которые будут приведены далее, гарантирует отличное качество программного продукта. Эти рекомендации могут лишь в какой-то степени помочь избежать появления некоторых ошибок в вашей программе и облегчат процесс ее разработки и отладки.

Чтобы более наглядно представить проблемы, связанные с отладкой программ, воспользуемся аналогией с медициной. Медицина — одна из древнейших наук — первая столкнулась с задачей диагностирования. Выявление отклонений от нормы в организме человека и установление причин, вызвавших эти отклонения,— краеугольный камень научной медицины. Не поставив точный диагноз, нельзя провести правильное лечение, а значит, нельзя и помочь больному. Несмотря на большой научный и технический прогресс в области медицинской диагностики, проблемы, стоящие здесь, далеки от своего решения. И на то есть целый ряд объективных и принципиальных причин. Ну, хотя бы, что такое норма и что такое патология? Где кончаются отклонения, обусловленные особенностью развития индивида, и где начинается патология, недопустимая для конкретного человека? Даже установление этой границы не всегда является простым делом. Где и как следует искать отклонения от нормы в организме человека? Если все-таки удалось выявить некоторые отклонения, то как на их основании определить причины этих отклонений — заболевание? Ведь одни и те же отклонения могут быть вызваны целым рядом независимых причин, а одно и то же заболевание в каждом конкретном случае может проявиться по-своему. Основой диагностирования в медицине служит модель «практически здорового» человека, стоящая за термином «норма». Сравнение именно с этой моделью и позволяет выявлять и диагностировать заболевания. Медицина немало усилий прикладывает к делу совершенствования этой модели.

Отладка программ имеет много общего с медицинской диагностикой, хотя есть и существенные отличия. Главное из них — это то, что создавая новую программу, программист должен создавать каждый раз

и новую модель «нормы» для этой программы. Аналогично техническим средствам медицинской диагностики на службе у программиста имеется достаточно богатый арсенал аппаратных и программных средств выявления ошибок. Как и в медицине, здесь можно выделить некоторые классы «симптомов» — форм проявления ошибок. Несмотря на это, отладка остается самым трудоемким этапом в процессе получения готового программного продукта. Рассмотрим некоторые причины, осложняющие отладку программ.

1. При отладке вычислительных программ или достаточно сложных программ, использующих большой набор обрабатываемых данных, иногда трудно вообще установить факт наличия ошибок в программе, так как программист лишь приблизительно представляет, какой результат должна давать программа. Например, интегрируя систему дифференциальных уравнений, мы, как правило, лишь качественно представляем картину изменения переменных и при этом не можем определить правильность последних значащих цифр полученного результата. Такое положение вещей заставляет иногда на стадии отладки прибегать к дублированию получения результата другим методом. В нашем примере это можно сделать, проинтегрировав систему уравнений с помощью другой программы интегрирования, и затем сравнить полученные результаты. Правда, при этом возникает вопрос: как быть в случае расхождения результатов? Ведь ошибочным может оказаться не первоначальный, а как раз контрольный результат.

2. В программе, как правило, содержится не одна, а несколько ошибок. Поэтому программист наблюдает эффект не одной ошибки, а результат взаимодействия нескольких ошибок.

3. Трудно, а иногда и просто невозможно разработать достаточно полную систему тестов, гарантирующих обнаружение всех ошибок в программе.

4. Одни и те же признаки ошибки (формы проявления ошибок) могут быть обусловлены различными причинами.

5. Некоторые ошибки не проявляются сами по себе, а лишь приводят к возникновению других ошибок (так называемые наведенные ошибки), которые и наблюдает программист. Например, обнуляя значения элементов массива и по ошибке выйдя за его границы, можно присвоить нулевое значение некоторой переменной, при использовании которой в арифметическом выражении будет зафиксировано деление на нуль, хотя исходно этой переменной было присвоено ненулевое значение и она нигде не изменялась.

6. Некоторые ошибки нельзя выявить, разбивая программу на части и отлаживая эти части по отдельности. Ошибка возникает лишь при взаимодействии этих частей. Таким образом, стратегия «разделяй и властвуй» не всегда оказывается применимой.

7. Иногда внесение каких-либо изменений в программу с целью локализации ошибки (например, промежуточная печать данных, трас-

сировка и т. п.) приводит к исчезновению проявлявшихся до этого признаков ошибки или к их изменению. Получается своеобразный заколдованный круг, когда всякая попытка выявить ошибку лишь маскирует ее, не давая никакой информации. Эта ситуация схожа с ситуацией, имеющей место в физике микромира,— использование какого-либо прибора для наблюдения процесса полностью изменяет этот процесс.

8. Иногда, изменяя программу методом проб и ошибок, можно устранить ошибку, т. е. она перестает как-либо проявлять себя. При этом остается абсолютно непонятным, в чем заключалась ошибка.

9. В ходе отладки программист нередко допускает просчет, необоснованно принимая некоторые предположения о возможных источниках ошибок. Например, используя стандартную библиотечную подпрограмму, он полностью уверен в ее безошибочности. Такие неверные установки, как правило, либо заводят программиста в логический тупик, либо программист впустую тратит время, пытаясь обнаружить ошибку там, где ее нет.

10. Не всегда имеет место повторяемость ошибки от запуска программы к запуску, даже если в программу и данные не вносились изменения. Наиболее часто это возникает при наличии в программе переменных, которым не присвоено начальное значение. В этом случае начальное значение этой переменной случайным образом зависит от содержимого соответствующего ей слова памяти в момент загрузки программы. В зависимости от этого значения возможно возникновение ошибочных ситуаций.

11. При разработке большого программного комплекса отдельные части программы создаются разными исполнителями, что значительно затрудняет согласование интерфейсов между частями.

Отладка программы — это прежде всего эксперимент, а не наблюдение за поведением программы. Различие между этими двумя понятиями удачно и точно охарактеризовал знаменитый русский физиолог И. П. Павлов: «Наблюдение собирает то, что ему предлагает природа, опыт же берет у природы то, что хочет». И как всякий эксперимент, отладку нужно уметь проводить. Очень важно при этом делать правильные выводы на основании данных, полученных из эксперимента. То, насколько при этом можно ошибиться, наглядно демонстрирует следующая шутливая история, взятая из замечательной книги Э. де Боно [5].

«Некий школьник предложил интересную гипотезу: он утверждал, что органы слуха у пауков находятся на ногах, и взялся доказать это.

Положив пойманного паука на стол, он крикнул: «Бегом». Паук побежал. Мальчик еще раз повторил свой приказ. Паук снова побежал. Затем юный экспериментатор оторвал пауку ноги и, снова положив его на стол, скомандовал: «Бегом». Но на сей раз паук остался неподвижен. «Вот видите,— заявил торжествующий мальчик,— стоило пауку оторвать ноги, как он сразу оглох».

Подводя итог сказанному, приведем некоторые рекомендации, которые, как мы надеемся, при осознанном и неформальном их использовании помогут вам выработать свой стиль программирования. Большинство рекомендаций имеют достаточно общий характер и могут быть использованы при программировании не только на фортране, но и на многих других языках программирования. Эти рекомендации не являются исчерпывающими. Много подобных советов читатель может почерпнуть из книг по разработке программ и структурному программированию [7, 9, 20, 22, 25, 28, 42].

1. В основе алгоритма решения задачи лежит математическая модель. Не нужно жалеть времени на разработку и изучение свойств этой модели. Это поможет лучше понять задачу и найти наиболее естественный путь ее решения.

2. При записи алгоритма придерживайтесь одного главного принципа: программа должна быть максимально простой и понятной. Это относится как к содержательной стороне, так и к форме записи программы. Стремление к ясности поможет вам найти наиболее естественные средства для выражения алгоритма. При этом сами собой решатся многие вопросы. Например, как быть с оператором GO TO? Если в конкретной ситуации использование этого оператора позволяет добиться наибольшей ясности, то используйте его. Очень часто слепое следование положениям структурного программирования уводит процесс программирования от основной цели, сводя его к максимальной структуризации программы и борьбе с оператором GO TO.

3. Максимально используйте «изобразительные» возможности языка программирования.

4. Относитесь с должным вниманием и аккуратностью к написанию даже очень простых частей программы. Помните, что программа — это единый организм, работоспособность которого определяется качеством всех ее частей, а не каких-то отдельных компонент. Просчет в каком-то одном месте может привести к неудаче в целом.

5. Не пытайтесь сразу написать эффективную программу. Это может привести к противоположному результату. Помните о следующем эмпирическом факте: 75 % времени выполнения программы приходятся на 25% ее текста. Лишь тестирование первого варианта вашей программы сможет выявить эти 25%, которые действительно требуют тщательной проработки. Экономия на мелочах нередко приводит я проигрышу в целом.

6. Помните об отладке с самого начала создания программы. Для этого активно используйте отладочную печать, вставляя соответствующие операторы еще при написании программы. Во всех местах, где происходит ветвление процесса вычислений, необходимо распечатывать данные, определяющие выбор варианта. В местах слияния ветвей решения следует печатать маркер этого места и информацию о том, по какой из ветвей прошло решение.

7. Не забывайте о надлежащей организации операций ввода-вывода данных. Не жалейте усилий на разработку средств, обеспечивающих наглядность представления вводимых и выводимых данных. Все вводимые данные должны распечатываться и проверяться на корректность.

Печать данных так же, как и представление вводимых данных, должна быть содержательной, отражающей структуру данных и их интерпретацию, а не быть хаотичной грудой цифр. Используйте такие формы отображения, как таблицы и графики.

8. Если вы хотите достигнуть определенных высот в программировании, то процесс создания программы не должен заканчиваться для вас получением работающей программы. Не спешите расстаться с вашим детищем, принесшим вам столько хлопот и мучений. Еще раз внимательно просмотрите программу. Постарайтесь извлечь максимум пользы на будущее. Оцените идеи и методы, использованные вами, с точки зрения их применимости в других ситуациях и пытайтесь выработать шаблоны, обобщающие эти идеи и методы. По мере того как число таких шаблонов будет увеличиваться, будет расти и ваше мастерство, умение оценивать все особенности конкретной задачи и пользоваться наиболее адекватными средствами для их отражения. Сказанное не следует понимать как призыв к шаблонному мышлению в программировании. Скорее наоборот. Применение того или иного шаблона требует глубокого предварительного анализа имеющейся ситуации с целью определения, какой именно шаблон можно применить в данной ситуации. И чем больше шаблонов находится в вашем распоряжении, тем более детальным должен быть этот анализ. Результатом такого анализа может быть рождение новой оригинальной идеи или метода. Таким образом, использование шаблонов освободит вас от изобретения велосипеда в тех ситуациях, когда можно добраться до цели в роскошной гоночной машине, и позволит сосредоточить основное внимание на решении новых для вас проблем.

9. Помните, что наилучшей документацией для любой программы является сама эта программа. Поэтому программа должна содержать в явном виде исчерпывающую информацию, представленную в виде комментариев.

10. Чтобы снизить погрешности результатов при выполнении вычислений с действительными числами, следует:

— избегать вычитания близких чисел,

— избегать деления больших по модулю чисел на малые числа, особенно, если последние имеют невысокую точность,

— сложение (вычитание) длинной последовательности чисел начинать с меньших чисел,

— стремиться уменьшить число операций,

— использовать алгоритмы, для которых известны оценки ошибок,

— не использовать сравнение на равенство действительных чисел в операторе IF.

11. Программисты в своих программах не всегда явно присваивают начальные значения используемым переменным, полагаясь на то, что транслятор или загрузчик присвоит им вполне определенные значения. Это может привести к неверной работе программы. Чтобы избежать такой ошибки, следует явным образом присваивать начальные значения всем используемым в программе переменным.

12. При программировании арифметических выражений во избежание возможных неоднозначностей в последовательности выполнения операций целесообразно использовать скобки.

13. При использовании в записи идентификатора цифр помещайте их только в конце идентификатора, так как цифры 0, 1, 2, 3, 4, 5 в середине идентификатора можно воспринять как буквы O, I, Z, З, Ч, S.

14. В тексте программы старайтесь явно не употреблять константы, за исключением быть может нуля и единицы. Для констант лучше вводить имена, которые и использовать в программе. Это приводит к лучшей «читабельности» программы и к уменьшению числа возможных ошибок.

15. Следует быть осторожным при использовании выражений, включающих операции умножения и деления с целыми числами. При этом нужно всегда помнить, что в фортране результатом деления целых чисел является целое число. Например, если K и N — целые переменные и N=2*(K/2), то K=N только тогда, когда K — четное число.

16. Избегайте употреблять смешанные арифметические выражения, содержащие переменные разных типов. Пользуйтесь функциями FLOAT и IFIX для преобразования значений переменных одного типа в значения другого типа данных.

ПРИЛОЖЕНИЕ 1

РЕАЛИЗАЦИЯ ОПЕРАТОРОВ УПРАВЛЕНИЯ ТРАНСЛЯТОРОМ ЯЗЫКА ФОРТРАН-ДУБНА НА БЭСМ-6

При описании реализации операторов используется автокод мадлен [26, 33].

1. Реализация оператора вычисляемого перехода.

GO TO (M1,M2, ... ,MN),K

Транслятор генерирует таблицу, содержащую команды перехода на соответствующие метки:

L+0:	,WTC, K	*модифицирование адреса следующей команды содержимым переменной* K
	,UJ,*	*передача управления на таблицу меток*
L+1:	,UJ,M1 ,NTR,18	*передача управления на метку* M1
L+2:	,UJ,M2 ,NTR,18	*передача управления на метку* M2
. . . . .		
L+N:	,UJ,MN	*передача управления на метку* MN

Здесь L — адрес, начиная с которого располагается код, соответствующий оператору вычисляемого перехода. Если метка MN находится непосредственно после оператора вычисляемого перехода, то команда по адресу L+N не генерируется транслятором.

Как видно из приведенного фрагмента, реализующего оператор вычисляемого перехода, при K=0 произойдет зацикливание, а при K=N+1 управление будет передано на следующий оператор, если метка MK не расположена непосредственно после оператора вычисляемого перехода.

2. Реализация оператора ASSIGN.

ASSIGN M1 TO J

Оператор реализуется тремя командами с использованием 14-го индексного регистра и сумматора:

```
14,VTM,*M1      засылка адреса метки в индекс-регистр
  ,ITS,   14    считывание содержимого индексного регистра
                на сумматор
  ,STX,   J     запись адреса по адресу переменной J
```

3. Реализация оператора перехода по предписанию.

GO TO K,(M1,M2, ... ,MN)

Оператор ASSIGN помещает в младшие 15 разрядов целой переменной K адрес метки перехода, а оператор перехода по предписанию передает управление по этому адресу.

В случае отсутствия предварительного выполнения оператора ASSIGN или использования идентификатора K в левой части арифметического выражения оператор перехода по предписанию интерпретирует младшие 15 разрядов как адрес (хотя его там нет) и передает управление по этому адресу. В этом случае последствия выполнения оператора непредсказуемы.

```
,WTC, K     модификация адреса
,UJ, 0      переход
```

4. Реализация оператора цикла. В диалекте языка фортран-Дубна употребление выражений для параметров цикла недопустимо,

```
   DO M ПЦ=H,K,Ш
    . . . . . .
   Тело цикла
    . . . . . .
M  . . . . . .
```

Оператор реализуется следующим образом:

```
      ,XTA,H          присваивание начального значения
                      параметру цикла
      ,ATX,ПЦ
      ,UJ  ,*T        безусловная передача управления
                      на проверку
*W:   ,BSS ,0
      . . . . . .
      Тело цикла
      . . . . . .
*M:   ,XTA,ПЦ         изменение параметра цикла
      ,A+X,Ш
      ,ATX ,ПЦ
*T:   ,X—A,K          проверка на окончание цикла
      ,UZA ,*W
```

Обратите внимание, что при H>K тело цикла не выполняется ни разу.

5. Реализация арифметического оператора IF.

IF (AB) M1, M2, M3

Оператор реализуется следующей последовательностью команд:

```
вычисление арифметического выражения AB
,ASN,57
,UZA,*M2        переход при значении AB=0
,ARX,
,UZA,*M1        переход при значении AB<0
,UJ  ,*M3       переход при значении AB>0
```

При совпадении меток возможны варианты.

6. Реализация логического оператора IF.

IF (L) S

где L — логическое выражение, S — оператор. Оператор реализуется следующим образом:

```
                вычисление логического выражения L
              , UZA,*L0000     переход при значении L=.TRUE.
              , UJ  ,*L1       переход при значении L=.FALSE.
*L0000   :    , BSS,
C ОПЕРАТОР  S
*L1      :    , BSS,
```

7. Реализация оператора вызова подпрограммы и вызова функции.

CALL SUB (A1,A2, ... , AN)

Сначала вычисляются все арифметические выражения, соответствующие фактическим параметрам. Значения выражений сохраняются в области рабочих переменных. В подпрограмму или функцию передаются адреса этих рабочих переменных или адреса простых переменных — фактических параметров. Передача параметров подпрограмме организуется через аппаратно реализованный стек (магазин). Оператору CALL соответствует следующая последовательность команд:

```
14,VTM,A1       передача адреса первого параметра
  ,ITS,14
14,VTM,A2       передача адреса второго параметра
  ,ITS,14
 . . . . . .
14,VTM,AN       передача адреса последнего параметра
  ,ITS,14
  ,CALL,SUB     переход к выполнению подпрограммы
```

Оператор обращения к функции:

P=FUN(A1,A2, ... ,AN)

транслируется в следующую последовательность команд:

```
14,VTM,A1          передача адреса первого параметра
  ,ITA,14
14,VTM,A2          передача адреса второго параметра
  ,ITS,14
.........
14,VTM,AN          передача адреса последнего параметра
  ,ITS,14
  ,CALL,FUN        переход к выполнению функции
  ,ATX,P
```

Следует обратить внимание — передача адреса первого фактического параметра функции отличается от способа передачи адресов остальных параметров. Для первого параметра не производится сдвига указателя стека.

8. Реализация операторов SUBROUTINE и FUNCTION. Оператор

SUBROUTINE SUBRUT(A1,...,AN)

реализуются следующим образом:

```
SUBRUT:   ,NAME,                заголовок программы
         14,VTM  ,SUBRUT        адрес тела программы
           ,UJ   ,*BEGIN        передается через И14
SUBRUT**: ,BSS  ,
          . . . . .
           тело программы
          . . . . .
*BEGIN:    ,BSS  ,
          7,MTJ  ,8             сохранение базы вызывающей
                                программы
         14,MTJ  ,10            сохранение адреса начала
                                программы
          7,VTM  ,*C            установка базы программы
           ,STX  ,AN            запоминание адресов
                                фактических параметров в
                                рабочей области
          . . . . . .
           ,STX  ,A1
           ,ITS  ,13            сохранение в стеке адреса
                                возврата
           ,ITS  ,ИР1           сохранение используемых
                                рабочих регистров (К =
                                1,...,6), если есть
                                необходимость
           ,ITS  ,ИРК
```

```
        ,ITS  ,8          сохранение базы вызываемой
                          программы
        ,ITS  ,           сдвиг указателя магазина
        ,NTP  ,3
        ,NTP  ,2
       10,UJ  ,           переход к выполнению
                          программы на метку
                          SUBRUT**
```

Оператор

FUNCTION FUNC(A1,A2, ... ,AN)

реализуется аналогично.

9. Реализация оператора RETURN. Оператор RETURN в подпрограммах и функциях оформляется следующим образом:

```
        ,UJ   ,*END       переход на блок выхода из модуля
*END:   ,BSS  ,
        ,STI  ,
        ,STI  ,7          восстановление базы
        ,STI  ,ИРК        восстановление сохраненных
        . . . . .         регистров ИР1, ... , ИР6
        ,STI  ,ИР1
        ,STI  ,8          восстановление адреса возврата (ИР8)
```

для функций добавляется команда засылки в стек значения

```
        ,UTC,*VALUE
        ,XTS,
       8,UJ  ,            возврат в модуль, из которого
                          произошел вызов
```

*VALUE — переменная, содержащая значение функции.

ПРИЛОЖЕНИЕ 2

АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ БЭСМ-6 И ОСОБЕННОСТИ ИХ ВЫПОЛНЕНИЯ

Система команд БЭСМ-6 включает в себя следующие операции над числами с плавающей точкой [21, 27, 36] (в скобках указана мнемоника команды на автокоде мадлен):

— сложение (A+X),
— вычитание (A—X),
— обратное вычитание (X—A),
— вычитание модулей чисел (AMX),
— изменение знака числа (AVX),
— умножение (A*X),
— деление (A/X),
— сложение порядков (E+X),
— вычитание порядков (E—X),
— корректировка порядка сложением (E+N),
— корректировка порядка вычитанием (E — N).

Рассмотрим перечисленные команды и некоторые особенности их выполнения.

1. Арифметическое сложение (мнемокод A+X). Число, находящееся на сумматоре, складывается с числом, содержащемся в слове оперативной памяти по исполнительному адресу. Результат остается на сумматоре и в 1—40-м разрядах регистра младших разрядов. Содержимое 41—48 разрядов регистра младших разрядов не изменяется. Операция может производиться как с нормализованными, так и с ненормализованными числами. Результат операции нормализуется и округляется, если нормализация влево и округление не блокированы.

2. Арифметическое вычитание (мнемокод A—X). Из числа, находящегося на сумматоре, вычитается число, содержащееся в слове оперативной памяти по исполнительному адресу. Дальнейшее выполнение команды аналогично выполнению команды арифметического сложения.

3. Обратное вычитание (мнемокод X—A). Из числа, содержащегося в слове оперативной памяти по исполнительному адресу, вычитается число, находящееся на сумматоре. Дальнейшее выполнение команды

аналогично выполнению команд арифметического сложения и вычитания.

4. Вычитание модулей (мнемокод АМХ). Формируются модуль числа, находящегося на сумматоре, и модуль числа, содержащегося в слове оперативной памяти по исполнительному адресу. Затем из модуля числа в сумматоре вычитается модуль числа по исполнительному адресу. Дальнейшее выполнение команды аналогично выполнению предыдущих трех команд.

5. Изменение знака (мнемокод АVХ). По этой команде производится изменение знака числа, содержащегося на сумматоре, в зависимости от знака числа, находящегося в слове оперативной памяти по исполнительному адресу. Если число по исполнительному адресу отрицательное, то знак числа, находящегося на сумматоре, изменяется. Если число по исполнительному адресу положительное, то знак числа на сумматоре не изменяется. 1—48-й разряды регистра младших разрядов гасятся. Если нет блокировки нормализации, то после выполнения операции производится нормализация числа.

6. Арифметическое умножение (мнемокод А*Х). Число, содержащееся на сумматоре, умножается на число, находящееся в слове оперативной памяти по исполнительному адресу. Результат остается на сумматоре и в 1—40-м разрядах регистра младших разрядов. Операция может производиться как с нормализованными, так и с ненормализованными числами. Результат операции нормализуется и округляется, если нормализация влево и округление не блокированы.

7. Арифметическое деление (мнемокод А/Х). По этой команде число, содержащееся на сумматоре, делится на число, находящееся в слове оперативной памяти по исполнительному адресу. Результат выполнения команды остается на сумматоре. Делитель должен быть обязательно нормализованным числом, иначе при выполнении команды произойдет внутреннее прерывание с признаком «деление на нуль», а на сумматоре остается неверный результат.

8. Сложение порядков (мнемокод Е+Х). При выполнении этой команды к коду числа, содержащегося на сумматоре, прибавляется код порядка числа, находящегося в слове оперативной памяти по исполнительному адресу, и из кода порядка результата вычитается число 64 (100В), так как при представлении чисел принято, что шкала порядков в ЭВМ сдвинута на 64 (100В), так что машинный порядок числа больше фактического на 64. Машинный порядок числа изменяется в пределах от 0 до +127. Нулевому порядку числа соответствует машинный порядок +64 (100В), максимальному порядку числа 63 соответствует машинный порядок 177В, минимальному порядку —64 соответствует 000В. Из-за того что машинный порядок сдвинут на 64 единицы, при сложении порядков порядок результата превышает фактический машинный порядок на 64 единицы. Для компенсации этого из порядка результата вычитается число 64. Мантисса числа, содержащегося в

сумматоре, сохраняется. Операция может производиться как с нормализованными, так и с ненормализованными числами. Результат операции нормализуется, если нормализация влево не блокирована. Содержимое 1—48-го разрядов регистра младших разрядов гасится.

9. Вычитание порядков (мнемокод Е—Х). При выполнении команды из кода порядка числа, содержащегося на сумматоре, вычитается код порядка числа, находящегося в слове оперативной памяти по исполнительному адресу, и к коду порядка результата прибавляется число 64(100В). Дальнейшее выполнение команды аналогично выполнению команды сложения порядков.

В операциях над числами (арифметическое сложение, вычитание и умножение) мантисса результата состоит из 80-ти разрядов, причем младшие 40 разрядов располагаются в регистре младших разрядов, а старшие — в 1—40-м разрядах сумматора. При нормализации сдвигаются все 80 разрядов мантиссы результата.

Так как при выполнении арифметических операций используется 80-разрядное представление числа, а мантисса числа в памяти имеет лишь 40 разрядов, то на результат вычисления оказывают влияние операции считывания числа из памяти на сумматор и записи числа в память с сумматора. Поэтому для лучшего понимания процесса вычисления в ЭВМ арифметических выражений рассмотрим выполнение этих операций.

10. Считывание числа (мнемокод ХТА). При выполнении этой команды 1—48-й разряды содержимого слова оперативной памяти по исполнительному адресу считываются в 1—48-й разряды сумматора. Код на регистре младших разрядов не изменяется.

11. Запись числа (мнемокод АТХ). При выполнении команды 1—48-й разряды содержимого сумматора записываются в оперативную память по исполнительному адресу. Содержимое сумматора и регистра младших разрядов сохраняется.

12. Считывание и магазинное обращение (запись) (мнемокод XTS). По этой команде содержимое 1—48-го разрядов сумматора записываются в оперативную память по адресу, определяемому счетчиком магазина (адрес первого свободного слова магазина), счетчик магазина увеличивается на единицу (новый адрес первого свободного слова магазина), затем на сумматор считывается содержимое слова оперативной памяти по исполнительному адресу. Содержимое регистра младших разрядов сохраняется.

13. Запись и магазинное обращение (считывание) (мнемокод STX). По этой команде содержимое 1—48-го разрядов сумматора записывается в оперативную память по исполнительному адресу, затем счетчик магазина уменьшается на единицу (адрес последнего выполненного слова магазина), содержимое слова по этому адресу считывается в 1—48-й разряды сумматора. Содержимое регистра младших разрядов сохраняется.

Выполнение арифметических операций на БЭСМ-6 имеет следующие особенности:

— При выполнении арифметических операций используется модифицированный дополнительный код. При выравнивании порядков сдвигаемое число попадает в регистр младших разрядов, начиная с 40-го разряда, и продолжает в нем сдвигаться. Максимальное количество сдвигов, равное 127, будет при сложении чисел с двоичными порядками $+63$ и -64.

— После выполнения операций, если необходимо, производится нормализация вправо. При этом последний разряд результата попадает в регистр младших разрядов, сдвигая его содержимое вправо на один разряд. Затем, в зависимости от состояния признаков блокировки нормализации и округления, производится нормализация и округление.

— При выполнении операции деления используется специальный метод, при котором остаток от деления не определяется. Если число делится нацело, то результат будет точным, если нет, то допускается ошибка в младшем разряде. Делитель должен быть нормализованным. Если это условие не выполнено, то происходит прерывание работы программы. Делимое может быть как нормализованным, так и ненормализованным. Округление после операции деления не производится.

Необходимо отметить, что арифметические операции могут выполняться в одном из следующих режимов:

— с нормализацией и округлением,

— с нормализацией и с блокировкой округления,

— с блокировкой нормализации и с округлением,

— с блокировкой нормализации и с блокировкой округления. Стандартным режимом в мониторной системе Дубна является режим с нормализацией и блокировкой округления.

Нормализация результата влево производится в том случае, если результат операции над мантиссами меньше 0.5 для положительных чисел и меньше или равен 0.5 для отрицательных чисел. Мантисса сдвигается влево на столько разрядов, сколько необходимо для того, чтобы мантисса результата стала больше или равна 0.5 для положительных чисел и больше 0.5 для отрицательных чисел. И затем из порядка результата вычитается число, равное количеству сдвигов. При нормализации результата влево может возникнуть отрицательное переполнение. В этом случае нормализация прекращается, а сумматор и регистр младших разрядов гасятся.

Округление результата в арифметических операциях сложения, вычитания, умножения производится логическим сложением единицы с младшим разрядом сумматора мантиссы в следующих случаях:

1) если нормализация не нужна, а в младших разрядах результата есть хотя бы одна единица,

2) после нормализации вправо, если в младших разрядах есть хотя бы одна единица,

3) после нормализации влево, если до нормализации была хотя бы одна единица в младших разрядах результата и при нормализации влево из младших разрядов в старшие не перешла ни одна единица.

Исключение составляет операция деления, которая не требует округления результата, так как при выбранном алгоритме ее выполнения частное равновероятно может получиться как с избытком, так и с недостатком.

ПРИЛОЖЕНИЕ 3

ИСПОЛЬЗОВАНИЕ ИНФОРМАЦИИ, ВЫДАВАЕМОЙ ОС ДИСПАК, ПРИ ОТЛАДКЕ ПРОГРАММЫ НА ФОРТРАНЕ

При возникновении в программе ошибки, фиксируемой ОС, печатается сообщение об ошибке и содержимое специальных регистров БЭСМ-6 (ОС фиксирует далеко не все ошибки при выполнении программы). Эта информация вместе со знанием того, как используются регистры ЭВМ программой, позволяет во многих случаях довольно легко обнаружить причину и место ошибки в программе.

Далее дано описание информации, выдаваемой ОС, и приводится методика поиска ошибок [6, 26, 33].

КОМАНДА ПРЕРЫВАНИЯ — содержимое слова, содержащего команду, при выполнении которой зафиксирована ошибка, печатается в формате команд.

ИСП. АДР.— печатается исполнительный адрес команды, при выполнении которой зафиксирована ошибка.

СУММАТОР — печатается содержимое сумматора в формате десятичного числа с плавающей точкой и в восьмеричном формате.

РМР — печатается содержимое регистра младших разрядов в восьмеричном формате.

АП — адрес прерывания. Это адрес либо той команды, которая вызвала аварийную ситуацию, либо адрес одной из семи следующих за ней команд.

Э — адрес последнего выполнившегося в программе экстракода (экстракод — специальная команда, выполняющая обращение к стандартным функциям, например SIN, COS, SQRT и др.).

РК — режим выполнения команды. Он показывает режим работы центрального процессора в момент возникновения ошибки. Смысл значений этого регистра такой же, как и у операнда в поле адреса команды NTR языка мадлен. Единица в соответствующем разряде РК означает следующее:

разряд: 1 — блокировка нормализации,
2 — блокировка округления,

3 — логическая группа,
4 — группа умножения,
5 — группа сложения,
6 — блокировка прерывания по переполнению.

Если, например, РК=3, то в момент прерывания программа работала в режиме блокировки округления и блокировки нормализации.

И1, И2, ..., И6 — содержимое первых шести индексных регистров. В них могут содержаться текущие значения тех индексов, с которыми прерванная программа обращалась к массивам. Например, если ошибка произошла в подпрограмме

```
      SUBROUTINE PRIMER
      DIMENSION B (100)
      DO 1 K=1, 100
    1 B(K)=0.
      RETURN
      END
```

то в И1 может стоять текущее значение переменной К.

И7 — так называемый базовый регистр, он указывает адрес базы — адрес первой константы или рабочей ячейки прерванной подпрограммы. Для подпрограммы, написанной на фортране, адрес базы всегда больше, чем адрес последнего оператора этой подпрограммы, который определяется по листингу относительных адресов, полученному при трансляции.

И10 — адрес последнего выполненного возврата (оператор RETURN выполняется таким образом, что адрес, по которому происходит возврат, оказывается в И10) или база той подпрограммы, откуда произошло обращение к прерванной подпрограмме. Так, например, если прервана подпрограмма PRIMER, а И10 указывает на тело подпрограммы PRIM, то очень вероятно, что вызов PRIMER произошел из PRIM, причем модуль PRIMER не обращался ни к каким другим фортранным подпрограммам.

И11 — рабочий регистр, который участвует при организации начала работы фортранных подпрограмм. Обычно И11=0.

И12 — адрес входа в подпрограмму. В начале работы фортранных подпрограмм в этот регистр заносится либо адрес второй ячейки подпрограммы, если обращение было сделано к основному входу, либо адрес входа, к которому было сделано обращение.

И13, И14 — в программах на фортране не используются.

И15 — адрес возврата. При выполнении оператора CALL, обращение к функции, оператору-функции и т. п. В этот регистр заносится адрес возврата, по которому можно определить место, из которого произошло обращение. Если И15=И10, то это означает, что возврат произошел, причем других обращений от момента вызова до прерывания к каким-либо подпрограммам не было.

```
ФОРТРАН
  (16.07.73)
            PROGRAM E1
            INTEGER M(10)
            DATA M/1,2,3,4,5,6,7,8,9,10/
            ISUM=0
     2      CALL SUBSUM(ISUM,M)
     3      PRINT 1,ISUM,M
          1 FORMAT(' ISUM=',I3/10I3)
            END
                    ВЫЗЫВАЕМЫЕ ФУНКЦИИ И ПОДПРОГРАММЫ
     SUBSUM
                          ТАБЛИЦА ОТНОСИТЕЛЬНЫХ АДРЕСОВ
        1    2    3    4    5    6    7    8    9    10
000   :00001:00002:00007

ФОРТРАН
  (16.07.73)
          SUBROUTINE SUBSUM(ISUM,M)

          DO 1 K=1,10
   2    1 ISUM=ISUM+M(K)

   3      RETURN
          END

                          ТАБЛИЦА ОТНОСИТЕЛЬНЫХ АДРЕСОВ
       1    2    3    4    5    6    7    8    9    10

000   :00001:00003:00015
```

```
*EXECUTE
 E1          01000   FT*722  E 01640   SUBSUM      02632   PT5BNP   E 03472   REVCAPD     04303
 PROGRAM   E 01000   FT*642  E 01640   BCDBEC*     02667   TTPPINT  E 03472   FSTREC*   C 04330
 BCDWRIT*    01057   FT*002  E 01640   FT2*      E 03174   BCDPUN     03503   FMPTAPE     04331
 FT*621    E 01057   FD*642  E 01641   KONVI*    E 03207   EBCDPUN  E 03527   RASPAK8     05107
 BCDENC*   E 01063   FD*722  E 01641   PRINT80     03303   ISOTCOSY   03712   RASPAK12    05125
 FT*611    E 01063   FC*722  E 01641   PRINT8    E 03303   ISOTCOSI E 03763   ZAPAK12     05140
 FT*571    E 01065   FC*642  E 01641   SUBPERR*    03471   SWRITE     04003   *ICHECK*  C 05155
 FT*561    E 01070   FC*002  E 01641   PRINT82   E 03472   SREAD    E 04004   ERR*        05173
 NEXTLET*  E 01075   IOEND*  E 01646   BCDPUN2   E 03472   SRWEND*  E 04102   SDEC        05316
 RK*       E 01112   FT*643  E 01646   PT8PUN    E 03472   LUNMUN  C 04116   SDEC*     E 05316
 WSY*      E 01363   FT*723  E 01646   PT5PUN    E 03472   OCTTDEC     04117   СВОБОДНО    05356
 IOCONT*     01574   FT*003  E 01646   BINPUN    E 03472   CARD2    C 04152
 IOAC*     E 01625   STOP*      02607   PT8BNP    E 03472   ISOICTTR   04177
I/O—ERROR 206 DETECTED IN BCDWRIT* CALLED FROM 02637B LUN 00
FORMAT                              0

V
I/O—ERROR 215 DETECTED IN BCDWRIT* CALLED FROM 02637B LUN 00
SPECIFICATION              FORMAT

                                    0

V
**********************************************************************
                КОМАНДА ПРЕРЫВАНИЯ ИСП. АДР.
КОНЕЦ ЗАДАЧИ    00 074 0000 00 22    00000      00000
 АП     Э       РК  И1     И2     И3     И4     И5     И6     И7     И10
02622  02622 006  32062  17404  00000  17404  17415  37001  02660  02464
          СУММАТОР                          РМР
+000000000000    +00   0000000000000000     0000000000040000
  И11    И12    И13    И14    И15    И16    И17    КРА  ЗПСЧ    И21
 00000  02633  01714  01711  02640  00000  53406  00000  53400  02003
**********************************************************************
```

И16 — портится при обращении к экстракодам, но может содержать адрес последнего фактического параметра в последнем выполненном оператор еобращения к фортранной подпрограмме (вызов подпрограммы CALL, вызов функции, вызов оператора-функции).

И17 — регистр счетчик магазина, его нормальное значение должно быть больше значения регистра ЗПСЧ. Если И17=ЗПСЧ, то возможен останов по считыванию. Если И17 меньше ЗПСЧ, то это означает, что программа (например, затерев сама себя) испортила И17.

ЗПСЧ — регистр адреса останова по записи или по считыванию. Если перед загрузкой программы не было расширения памяти (*CALL FICMEMORY), то ЗПСЧ=53400.

Сказанное о возможных состояниях регистров может не соответствовать действительности в следующих случаях:

1) если останов произошел в какой-либо стандартной или системной программе,

2) если программа теряет управление,

3) в случае употребления подпрограмм, написанных не на фортране.

Пример. Рассмотрим на примере конкретной программы (см. задачу 17 и ее решение) методику использования служебной информации для поиска ошибок. На с. 134, 135 приведен полный листинг программы, содержащий: тексты главной программы и подпрограммы с таблицами относительных адресов операторов программы, таблицу загрузки программы в память, которая следует после управляющей директивы *EXECUTE, диагностическое сообщение об ошибке и информацию, печатаемую операционной системой.

Как видно из листинга, при исполнении программы произошла ошибка ввода-вывода. По виду оператора FORMAT (он напечатан в строке, следующей после диагностического сообщения об ошибке), который при этом использовался, ясно, что он носит случайный характер и такая ситуация могла возникнуть при потере управления, когда программа начинает случайным образом «гулять» по памяти, тем более, что по адресу 2637 (все адреса даются в восьмеричном виде) никаких обращений к операциям ввода-вывода нет. Анализируя выдаваемую операционной системой информацию, можно заключить следующее.

1. Базовый регистр И7=2660 принадлежит подпрограмме SUBSUM: 2632<2660<2667,

2. Регистр И12=2633 — адрес второго слова подпрограммы SUBSUM, начинающейся с адреса 2632: 2633=2632+1.

3. Обращение к системной подпрограмме BCDWRIT* произошло по адресу 2637, что указывает на оператор

```
1 INSUM=INSUM+M(K)
```

подпрограммы SUBSUM, которому соответствует диапазон адресов от 2635 до 2646 включительно.

Регистр И15=2640 указывает, что возврат должен осуществиться на команду, находящуюся в диапазоне адресов оператора

1 INSUM=INSUM+M(K)

так как 2635<2640<2646.

Таким образом, все подозрения относительно виновника ошибки падают на оператор

1 INSUM=INSUM+M(K)

при выполнении которого произошло обращение к подпрограмме BCDWRIT*. Просматривая информацию, относящуюся к переменным, входящим в этот оператор, обнаруживаем, что массив M не описан в подпрограмме SUBSUM. В результате этого обращение к элементу массива воспринялось как обращение к одноименной функции. В результате управление было передано по адресу массива M, что и привело к «блужданию» программы по памяти к подпрограмме BCDWRIT*. Приведенное ниже автокодное расширение подпрограммы SUBSUM подтверждает сказанное. Обратите внимание на то, что идентификатор M, воспринимаемый в подпрограмме SUBSUM как имя функции, не печатается в списке имен подпрограмм и функций, вызываемых из SUBSUM. Это объясняется тем, что M является формальным параметром подпрограммы SUBSUM, а следовательно, фактическое имя может быть произвольным.

АВТОКОД MADLEN

```
                E1        :  ,NAME  ,
                FT*571    :  ,SUBP  ,
                FT*002    :  ,SUBP  ,
                FT*003    :  ,SUBP  ,
                STOP*     :  ,SUBP  ,
                SUBSUM    :  ,SUBP  ,
                *C        :  ,EQU   ,  *+32
0000            PROGRAM   :  ,ENTRY,
0000  0001                   14,VTM   ,  E1
 —    0025                     ,UJ    ,  *BEGIN
0001            E1        :  ,BSS   ,
              C*****НАЧАЛО ОПЕРАТОРА 1
0001                         ,XTA   ,=64000000 00000000
 —                           ,ATX   ,  ISUM
0002                         ,NTR   ,  3
              C*****НАЧАЛО ОПЕРАТОРА 2
 —    0043                   14,VTM   ,  ISUM
0003                         ,ITS   ,  14
 —    0045                   14,VTM   ,  M
0004                         ,ITS   ,  14
 —                           ,NTR   ,  18
0005                         13,VJM   ,  SUBSUM
 —
0006                         ,NTR   ,  3
 —                           ,NTR   ,  18
              C*****НАЧАЛО ОПЕРАТОРА 3
```

```
0007  0035                14,VTM  , /1
 —                          ,ITS  , 14
0010                       13,VJM  , FT*571
0011  0043                14,VTM  , ISUM
 —                          ,ITS  , 14
0012                       13,VJM  , FT*002
 —
0013                          ,NTR  , 3
 —                            ,XTA  ,=64000000 00000001
0014                          ,ATX  , IMPL*1
 —                            ,NTR  , 18
0015          *W0001    :     ,BSS  ,
0015  0044                    ,WTC  , IMPL*1
 —                          1,VTM  ,
0016  0044                   1,UTC  , M—1
 —                         14,VTM  ,
0017                          ,ITS  , 14
 —                         13,VJM  , FT*002
0020                          ,XTA  , IMPL*1
 —                            ,NTR  , 3
0021                          ,A+X  ,=64000000 00000001
 —                            ,ATX  , IMPL*1
0022                          ,X—A  ,=64000000 00000012
 —                            ,NTR  , 18
0023  0015                    ,UZA  , *W0001
 —                         13,VJM  , FT*003
0024                       13,VJM  , STOP*
 —
0025          *BEGIN    :     ,BSS  ,
0025                         7,MTJ  , 8
 —                          14,MTJ  , 10
0026  0040                   7,VTM  , *C
 —                            ,ITS  , 13
0027                          ,ITS  , 1
 —                            ,ITS  , 8
0030                          ,ITS  ,
 —                            ,NTR  , 3
0031                          ,NTR  , 2
 —                          10,UJ   ,
0032          *END      :     ,BSS  ,
0032                          ,STI  ,
 —                            ,STI  , 7
0033                          ,STI  , 1
 —                            ,STI  , 8
0034                          8,UJ   ,
 —
0035          /1        :     ,OCT  , 12023440 22251525
0036                          ,OCT  , 23236447 13044463
0037                          ,OCT  , 13630460 22231451
0043          ISUM      :     ,BSS  , 1
0044          IMPL*1    :     ,BSS  , 1
0045          M         :     ,BSS  , 10
                              ,DATA ,
0043          *DATA     :     ,BSS  ,
0043                          ,OCT  , 64000000 00000001
0044                          ,OCT  , 64000000 00000002
```

```
0045                        ,OCT   , 64000000 00000003
0046                        ,OCT   , 64000000 00000004
0047                        ,OCT   , 64000000 00000005
0050                        ,OCT   , 64000000 00000006
0051                        ,OCT   , 64000000 00000007
0052                        ,OCT   , 64000000 00000010
0053                        ,OCT   , 64000000 00000011
0054                        ,OCT   , 64000000 00000012
                          10,SET   , *DATA
0055                       1,      , M
                           ,END    , E1
СТРУКТУРА ПОДПРОГРАММЫ:
          ЗАГОЛОВОК  00001
        ОБОЗНАЧЕНИЯ  00007
                 —   00000
                 —   00000
   ИДЕНТИФИКАТОРЫ  00005
          КОМАНДЫ  00035
      ГРУППА BSS  00014
       КОНСТАНТЫ  00006
           ДАННЫЕ  00012
        РАССЫЛКИ  00001
ЧИСЛО ПЕРФ. 0089       ЧИСЛО ОШИБ  ОПЕРАТОРОВ 0000
**************************************************************
АВТОКОД MADLEN
          SUBSUM   :  ,NAME,
          *C       :  ,EQU  , *+22
0000  0001             14,VTM  , SUBSUM
 —    0016               ,UJ   , *BEGIN
0001      SUBSUM   :  ,BSS  ,
          C*****НАЧАЛО ОПЕРАТОРА 1
0001                     ,XTA  ,=64000000 00000001
 —                       ,ATX  , K
0002                     ,NTR  , 3
 —                       ,NTR  , 18
0003      *W0001   :  ,BSS  ,
          C*****НАЧАЛО ОПЕРАТОРА 2
0003      *1       :  ,BSS  ,
0003  0032            ;14,VTM  , K
 —                       ,ITA  , 14
0004                     ,AOX  ,=40000000 00000000
 —    0034               ,WTC  , M
0005                    13,VJM  ,
 —
0006                     ,ATX  , *ERROO
 —    0033               ,WTC  , ISUM
0007                     ,XTA  ,
 —                       ,NTR  , 3
0010                     ,A+X  , *ERROO
 —    0033               ,WTC  , ISUM
0011                     ,ATX  ,
 —                       ,XTA  , K
0012                     ,NTR  , 3
 —                       ,A+X  ,=64000000 00000001
0013                     ,ATX  , K
 —                       ,X—A  ,=64000000 00000012
```

```
0014                        ,NTR  ,  18
 —    0003                  ,UZA  ,  *W0001
            C*****НАЧАЛО ОПЕРАТОРА 3
0015  0024                  ,UJ   ,  *END
 —
0016          *BEGIN   :    ,BSS  ,
0016                        7,MTJ  ,  8
 —                         14,MTJ  ,  10
0017  0026                  7,VTM  ,  *C
 —                          ,STX  ,  M
0020                        ,STX  ,  ISUM
 —                          ,ITS  ,  13
0021                        ,ITS  ,  8
 —                          ,ITS  ,
0022                        ,NTR  ,  3
 —                          ,NTR  ,  2
0023                       10,UJ   ,
 —
0024          *END     :    ,BSS  ,
0024                        ,STI  ,
 —                          ,STI  ,  7
0025                        ,STI  ,  8
 —                         8,UJ   ,
0031          *ERROO   :    ,BSS  ,  1
0032          K        :    ,BSS  ,  1
0033          ISUM     :    ,BSS  ,  1
0034          M        :    ,BSS  ,  1
                            ,END  ,  SUBSUM
СТРУКТУРА ПОДПРОГРАММЫ:
        ЗАГОЛОВОК  00001
     ОБОЗНАЧЕНИЯ   00000
              —    00000
              —    00000
ИДЕНТИФИКАТОРЫ     00000
         КОМАНДЫ   00026
      ГРУППА BSS   00004
      КОНСТАНТЫ    00003
          ДАННЫЕ   00000
       РАССЫЛКИ    00000
ЧИСЛО ПЕРФ. 0056   ЧИСЛО ОШИБ. ОПЕРАТОРОВ 0000
```

СПИСОК ЛИТЕРАТУРЫ

1. Айнберг В. Д., Геронимус Ю. В. Основы программирования для единой системы ЭВМ.— 2-е изд., испр. и доп.— М.: Машиностроение, 1985.
2. Акофф Р. Искусство решения проблем.— М.: Мир, 1982.
3. Баяковский Ю. М., Вьюкова Н. И., Галатенко В. А., Михайлова Т. Н., Морозова Л. Б., Ходулев А. Б., Штаркман Вик. С. Расширенный фортран — форекс (руководство для пользователя).— М.: ИПМ им. М. В. Келдыша АН СССР, 1983.
4. Березин И. С., Жидков Н. П. Методы вычислений. Т. 1.— М.: Наука, 1966.
5. де Боно Э. Рождение новой идеи. О нешаблонном мышлении.— М.: Прогресс, 1976.
6. Боровин Г. К., Кугушев Е. И., Ярошевский В. С. Ошибки-ловушки при программировании на фортране. Инструкция по их выявлению и устранению.— М.: ИПМ им. М. В. Келдыша АН СССР, 1982.
7. Бутаков Е. А. Методы создания качественного программного обеспечения ЭВМ.— М.: Энергоатомиздат, 1984.
8. Бухтияров А. М., Маликова Ю. П., Фролов Г. Д. Практикум по программированию на фортране.— М.: Наука, 1983.
9. Ван Тассел Д. Стиль, разработка, эффективность, отладка и испытание программ.— М.: Мир, 1981.
10. Вейценбаум Дж. Возможности вычислительных машин и человеческий разум. От суждения к вычислениям.— М.: Радио и связь, 1982.
11. Вирт Н. Систематическое программирование. Введение.—М.: Мир, 1977.
12. Воеводин В. В. Вычислительные основы линейной алгебры.— М.: Наука, 1977.
13. Горелик А. М., Ушкова В. Л., Шура-Бура М. Р. Мобильность программ на фортране.— М.: Финансы и статистика, 1984.
14. Грис Д. Конструирование компиляторов для цифровых вычислительных машин.— М.: Мир, 1975.
15. Грис Д. Наука программирования.— М.: Мир, 1984.
16. Грунд Ф. Программирование на языке фортран IV.— М.: Мир, 1976.
17. Дрейфус М., Ганглоф К. Практика программирования на фортране, Упражнения с комментариями.— М.: Мир, 1978.

18. Ерофеев В. И., Меркушов Ю. П., Першиков В. И., Соколов А. П. Средства отладки программ в ОС ЕС ЭВМ.— М.: Статистика, 1979.
19. Жаблон К., Симон Ж.-К. Применение ЭВМ для численного моделирования в физике.— М.: Наука, 1983.
20. Зиглер К. Методы проектирования программных систем.— М.: Мир, 1985.
21. Инструкция по программированию на БЭСМ-6.—М.: ИТМ и ВТ АН СССР, 1967.
22. Йодан Э. Структурное проектирование и конструирование программ.— М.: Мир, 1979.
23. Карпов В. Я. Алгоритмический язык фортран.— М.: Наука, 1976.
24. Катцан Г. Язык фортран 77.— М.: Мир 1982.
25. Керниган Б., Плодже р Ф. Элементы стиля программирования.— М.: Радио и связь, 1984.
26. Королев Л. Н. Структуры ЭВМ и их математическое обеспечение.— 2-е изд.— М.: Наука, 1978.
27. Мазный Г. Л. Программирование на БЭСМ-6 в системе «Дубна».— М.: Наука, 1978.
28. Майерс Г. Надежность программного обеспечения.— М.: Мир, 1980.
29. Майерс Г. Архитектура современных ЭВМ.— М.: Мир, 1985.
30. Мак-Кракен Д., Дорн У. Численные методы и программирование на фортране.— М.: Мир, 1977.
31. Меткалф М. Оптимизация в фортране.— М.: Мир, 1985.
32. Пратт Т. Языки программирования: разработка и реализация.— М.: Мир, 1977.
33. Принципы работы системы IBM/370.— М.: Мир, 1975.
34. Радд. У. Программирование на языке ассемблер и вычислительные системы IBM/360 и 370.— М.: Мир, 1979.
35. Риндфлайш Д. Отладка программ в системах 360/370 на основе дампов памяти операционной системы.— М.: Машиностроение, 1982.
36. Салтыков А. И., Макаренко Г. И. Программирование на языке фортран.— 2-е изд.— М.: Наука, 1984.
37. Салтыков А. И., Семашко Г. Л. Программирование для всех.— 2-е изд.— М.: Наука, 1986.
38. Самарский А. А. Введение в численные методы.— М.: Наука, 1987.
39. Таненбаум Э. Многоуровневая организация ЭВМ.— М.: Мир, 1979.
40. Тербер К. Дж. Архитектура высокопроизводительных вычислительных систем.— М.: Наука, 1985.
41. Форсайт Дж., Мальколь м М., Моулер Е. Машинные методы математических вычислений.— М.: Мир, 1980.
42. Хьюз Дж., Мичтом Дж. Структурный подход к программированию.— М.: Мир, 1980.
43. Хьюз Ч., Пфлигер Ч., Роуз Л. Методы программирования: Курс на основе фортрана.— М.: Мир, 1981.
44. Шнейдерман Б. Психология программирования.— М.: Радио и связь, 1984.

ББК 22.18
Б83
УДК 519.6

Боровин Г. К., Комаров М. М., Ярошевский В. С. **Ошибки-ловушки при программировании на фортране**/Под ред. Ю. М. Баяковского — М.: Наука. Гл. ред. физ.-мат. лит., 1987.— 144 с. — (Библиотечка программиста).

Основное содержание книги составляют задачи, цель которых — поиск ошибок в приведенных программах на фортране. Задачи охватывают широкий спектр ошибок, относящихся практически ко всем аспектам языка фортран: синтаксис языка, типы данных, управляющие операторы, модульная организация программ, представление данных и арифметические операции, операторы ввода-вывода. Приводятся подробные решения. Детально рассмотрены источники возникновения ошибок. Обсуждается стиль программирования как главный фактор, влияющий на качество программы.

Книга рассчитана как на начинающих, так и на опытных программистов. Многие вопросы представляют интерес при программировании на других языках.

Рис. 3. Библиогр. 44 назв.

Рецензент

член-корреспондент АН СССР *Л. Н. Королев*

Б $\frac{1702070000-113}{053(02)-87}$ 7-87

Геннадий Константинович Боровин
Михаил Михайлович Комаров
Виктор Семенович Ярошевский

ОШИБКИ-ЛОВУШКИ ПРИ ПРОГРАММИРОВАНИИ НА ФОРТРАНЕ

Серия «Библиотечка программиста», вып. 50

Редактор *О. И. Сухова*
Художественный редактор *Г. М. Коровина*
Технический редактор *В. Н. Кондакова*
Корректоры *Е. Ю. Рычагова, О. М. Березина*

ИБ № 32285

Сдано в набор 15.10.86. Подписано к печати 21 04 87. Т-10331. Формат 84×108/32. Бумага тип. № 3. Гарнитура литературная. Печать высокая. Усл. печ. л. 7 56. Усл. кр.-отт. 7,77. Уч.-изд. л. 8,8. Тираж 116 000 экз Заказ № 3248. Цена 55 коп.

Ордена Трудового Красного Знамени издательство «Наука»
Главная редакция физико-математической литературы
117071 Москва В-71, Ленинский проспект, 15

Ордена Октябрьской Революции и ордена Трудового Красного Знамени МПО «Первая Образцовая типография» имени А, А. Жданова Союзполиграфпрома при Государственном комитете СССР по делам издательств, полиграфии и книжной торговли 113054 Москва М-54, Валовая, 28

Отпечатано во 2-й типографии издательства «Наука» 121099 Москва Г-99, Шубинский пер., 6 Зак 745