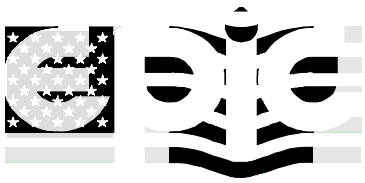


РУКОВОДСТВО ПО GAMS*

(РУССКАЯ ВЕРСИЯ)

Авторы:
Антони Брук
Дэвид Кендрик
Александр Меераус
Рамеш Раман

Перевод:
Ольга Тихонова
Андрей Савицкий
Дейн МакКинни

Август 1999

Подготовлено:
Центральноазиатской миссией
Агентства США по международному развитию

Environmental Policy and Institutional Strengthening Indefinite Quantity Contract (EPIQ)
Partners: International Resources Group, Winrock International, and Harvard Institute for International Development

Subcontractors: PADCO; Management Systems International; and Development Alternatives, Inc.

Collaborating Institutions: Center for Naval Analysis Corporation; Conservation International; KBN Engineering and Applied Sciences, Inc.; Keller-Bliesner Engineering; Resource Management International, Inc.; Tellus Institute; Urban Institute; and World Resources Institute.

* Оригинал документа на английском языке разработан GAMS Corporation

Task Order No. 813
Contract No. PCE-I-00-96-00002-00

РУКОВОДСТВО ПО GAMS*
(РУССКАЯ ВЕРСИЯ)

Авторы:
Антони Брук
Дэвид Кендрик
Александр Меераус
Рамеш Раман

Перевод:
Ольга Тихонова
Андрей Савицкий
Дейн МакКинни

Август 1999

Подготовлено:
Центральноазиатской миссией
Агентства США по международному развитию

Environmental Policy and Institutional Strengthening Indefinite Quantity Contract (EPIQ)
Partners: International Resources Group, Winrock International, and Harvard Institute for International Development

Subcontractors: PADCO; Management Systems International; and Development Alternatives, Inc.

Collaborating Institutions: Center for Naval Analysis Corporation; Conservation International; KBN Engineering and Applied Sciences, Inc.; Keller-Bliesner Engineering; Resource Management International, Inc.; Tellus Institute; Urban Institute; and World Resources Institute.

* Оригинал документа на английском языке разработан GAMS Corporation

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 1

1.1. ОБОСНОВАНИЕ

С развитием алгоритмизации и компьютеризации в 1950-х и 1960-х годах был достигнут значительный прогресс в решении больших математических задач.

Однако в 70-х годах широта применения компьютерной техники оказалась ниже, чем ожидалось, потому что действия связанные с решением математических задач составляли только малую часть общих решаемых проблем в области моделирования. Большая часть времени требовалась на то, чтобы развивать модель, включающую в себя подготовку данных, трансформацию подготовленных данных и отчет о готовности данных. Каждая модель, требовала много часов анализа и программного времени, только для того, чтобы правильно организовать данные и написать программы, которые должны были бы впоследствии трансформировать эти данные в формы, требуемые математическими программирующими программами-оптимизаторами. Кроме того, огромную трудность представляло решение проблемы выявления и уменьшения количества ошибок, потому что программы, которые выполняли операции с данными, были доступны только специалистам, которые писали их, и не были доступны специалистам, которые их анализировали и были ответственны за проект.

GAMS был задуман как средство способное изменить эту ситуацию и в частности чтобы:

- обеспечить высокий уровень языка для компактного представления больших и комплексных моделей;
- позволить относительно просто и безопасно делать изменения в описаниях модели;
- предоставлять в простых формах однозначные операторы алгебраических связей;
- позволить выполнять описание модели, которое не зависит от алгоритмов решения.

1.2. ОСНОВНЫЕ ОСОБЕННОСТИ GAMS

Некоторые особенности GAMS будут разъяснены в следующих подразделах.

1.2.1. ОСНОВНЫЕ ПРИНЦИПЫ

Основная идея GAMS заключалась в том, чтобы объединить идеи, вытекающие из теории реляционной базы данных и математического программирования.

Кроме того была сделана попытка соединить эти идеи таким образом, чтобы удовлетворить потребности стратегического моделирования.

Теория реляционной базы данных обеспечивает структурную композицию (интегрированную систему для ПЭВМ типа IBM PC, объединяющую различные виды информации с использованием правил иерархии) для развития организации основных данных и развития трансформационных возможностей.

Математическое моделирование обеспечивает путь описания проблемы и различные методы ее решения.

При разработки системы GAMS придерживались следующих принципов:

- Пользователю должны были быть доступны все существующие алгоритмические методы, при этом новая система не должна была изменить представление пользователя о моделировании. Ввод новых методов или новая реализация существующих методов должны были осуществляться без внесения изменений в существующие модели. Линейные, нелинейные, смешанные целочисленные, смешанные целочисленные нелинейные оптимизации и смешанные дополнительные задачи могли бы подключаться без ущерба структурной конструкции создаваемых моделей.

- Оптимизационная задача должна была иметь структуру независимую от данных, которые она использует. Это разделение логики и данных позволяет увеличивать задачу в размерах таким образом, что это не приводило к увеличению в комплексности представления основной части модели построенной на логике.

- Использование реляционных данных модели требует, чтобы распределение ресурсов компьютера было автоматизированным. Это означает, что большие и комплексные модели могут быть составлены таким образом, чтобы пользователь не заботился о таких деталях, как размеры массивов или объем памяти.

1.2.2 ДОКУМЕНТАЦИЯ

Модель GAMS представлена в форме, легко читаемой людьми и компьютером. Это означает, что программа GAMS является также и документацией модели и отдельное описание модели, требуемое в прошлом (которое было неприятной необходимостью, которое необходимо было поддерживать и которое редко было современным), больше составлять не требуется. Кроме того, GAMS задуман таким образом, чтобы соединить в себе следующие свойства, которые отвечают требованиям пользователя к документации:

- Модель GAMS записывается в сжатой форме, но в ней полностью использована элегантность математического представления задач.

- Трансформация всех данных определяется сжато и в алгебраической форме. Это означает, что все данные могут быть введены в самой элементарной форме, что все трансформации выполняются в структуре модели, и что в информации вывода имеется возможность проверки трансформации данных.

- Комментарий (поясняющий текст) сделан как часть определения всех символов и воспроизводится каждый раз, когда соответствующие символам значения выводятся на печать.

- Вся информация, которая необходима, чтобы понять модель, имеется в одном документе.

Конечно, чтобы полностью понять ценность всех свойств модели, необходимо знать некоторые разделы науки, но GAMS позволяет - сделать модель более доступной, более понятной, более поддающейся проверке, более заслуживающей доверие .

1.2.3. МОБИЛЬНОСТЬ

Система GAMS задумана таким образом, чтобы она могла быть работоспособна на различных типах компьютеров без изменения оригинального текста. Модель, сделанная на персональной ЭВМ класса 386, затем может быть использована на машине класса 586. Один разработчик может составить модель, но в дальнейшем она может быть использована другим пользователем, который работает вдали от разработчика.

В противоположность предыдущим подходам, необходимо перемещать только один файл - файл содержащий модель GAMS. Модель содержит в себе все необходимые данные и логические описания для того, чтобы получить решение.

1.2.4. ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС

Требования к возможности использования на различных компьютерах также относятся и к вопросу о пользовательском интерфейсе.

Основа системы GAMS файл-ориентированая, поэтому не существует какого-то особого редактора или графических подпрограмм ввода и вывода сопровождающих вычислительный процесс. Чем загружать пользователя необходимостью изучать еще один редактор, GAMS предпочел открытую архитектуру, в которой каждый пользователь может использовать по его выбору любой редактор и процессор. Эта основа пользовательского интерфейса, которая облегчает интеграцию GAMS с различными подпрограммами и будущей пользовательской средой.

1.2.5. БИБЛИОТЕКА МОДЕЛИ

Когда архитекторы задумывают новое здание, они развивают новую структуру, но используют идеи и технологии, которые были апробированы предшественниками. Это справедливо и в других областях: некоторые элементы идей из предыдущих проектов служат источником идей для новых разработок.

Мы (имеются в виду создатели GAMS) собрали модели, составленные на ранних стадиях разработок GAMS, и используем их в библиотеке примеров. Часть из них являются стандартными учебными примерами и могут быть использованы в вопросах формулирования задач и иллюстрации возможностей GAMS. Остальные - это модели, которые интересны с точки зрения методов и данных, которые они используют. Все модели, представленные в библиотеке, и описаны в открытой литературе. В настоящее время комплект моделей включен в систему GAMS и базу данных таким образом, чтобы помогать пользователю использовать примеры так, чтобы осветить интересующие его разделы, секторы или темы.

Чтобы описать особенности в различных главах, использован синтаксис Backus-Naur из системы обозначения (BNF), где:

[] означает, что заключенная в них конструкция является необязательной;

{} означает, что заключенная в них конструкция может быть повторена ноль или большее число раз;

| означает, что имеется оператор, выраженный через аргументы с обоих сторон символа.

1.3. ОРГАНИЗАЦИЯ КНИГИ

Некоторые пособия к программным средствам являются как бы справочниками: они в деталях описывают каждую команду. Другие пособия обучают пользователя шаг за шагом на малом числе примеров. Эта книга использует оба приема. Текст разделен на три части. Первая часть (разделы 1 и 2) являются вступительной (предварительной). Раздел 2 является кратким учебным пособием, где для обучения проводится только один пример с некоторыми деталями - небольшая модель о транспорте, прочитав который пользователь сможет быстро понять возможности GAMS. Следующая часть (разделы 3-14) составляет основу книги. Компоненты языка GAMS представлены в определенном порядке с поясняющими примерами, которые часто взяты из библиотеки примеров.

Все модели, взятые из библиотеки, указаны в квадратных скобках (например, [TRNSPORT]). Некоторые специализированные материалы были намеренно опущены, потому что основной целью GAMS было сделать его доступным наиболее широкой аудитории, особенно для тех, кто не имеет значительного опыта в использовании компьютера или математических программирующих системах. Считается, что пользователь имеет представление о численных методах и математическом моделировании.

Третья часть (разделы 15-16) представляет собой специальные рассуждения способные принести пользу при достаточной уровне освоения GAMS и могут быть изучены при необходимости. Пользователи больших, комплексных и дорогих моделей найдут здесь много полезного материала. Темы включают долларовое условие и особенности программирования текущего контроля, такие как **loop** и **if-else.**

УЧЕБНОЕ ПОСОБИЕ GAMS 2

РИЧАРДА Е. РОЗЕНХАЛА

2.1. ВВЕДЕНИЕ

Вводная часть этой книги посвящена детальному изучению небольшой и простой оптимизационной задачи, на примере которой показано, как в ней использовать GAMS при формулировании, решении и анализе. Она была написана Ричардом Е. Розентхалом из Naval Postgraduated School г.Монтерей, Калифорния. Пример небольшой, но полностью раскрывает GAMS и его особенности. В разделе сделано много ссылок к другим частям книги, но они только указывают, где можно ознакомиться с информацией более детально; представленный здесь материал может читаться эффективно и без этих ссылок.

Рассматриваемый пример "оптимизация транспортных перевозок" - задача линейного программирования, которая исторически используется в качестве "лабораторного животного" в развитии технологии оптимизации. [Смотри, например, Dantzig (1963)]. Это хороший выбор для иллюстрации возможностей языка алгебраического моделирования GAMS, потому что задачи "оптимизации транспортных перевозок" (при этом не важно какой большой пример вы рассматриваете) обладают простой, описываемой алгебраически структурой. Вы увидите, что если бы была рассмотрена более сложная задача "оптимизации транспортных перевозок", то и в этом случае почти все операторы в исходном файле GAMS, которые здесь представлен остался бы почти без изменений.

При постановке задачи "оптимизации транспортных перевозок" считается известным:

- запасы товаров на нескольких заводах;
- требование нескольких потребителей на отдельные виды товаров;
- стоимость перевозки единицы товара потребителям.

Вопрос экономиста: сколько, каких видов товаров, откуда и каким потребителям транспортировать, чтобы суммарная стоимость перевозки была минимальной?

Алгебраически эта задача обычно представляется в следующем виде:
Индексы:

i = plants (*заводы*)
j = markets (*потребители*)

Заданные значения:

a_i **= supply of commodity of plants i (in cases)** *наличие товара на заводе i*

b_i **= demand for commodity at market j (cases)** *требования на товар потребителя j*

Примечание: В моделях GAMS используются только латинские буквы. Перевод на русскй язык в представленной моделе дан для облегчения восприятия структуры модели. В реальных моделях коментарии на русском языке недопустимы.

c_{ij} **= cost per unit shipment between plant i and market j ($/case)**
стоимость каждой поездки между местом нахождения товара и потребителем j ($/ящик)

Расчетные переменные:

x_{ij} = amount of commodity to ship from plan i to market j (cases)

количество товара для перевозки от месторасположения завода i до потребителя j (ящики)

где **$x_{ij} \geq 0$**, для всех **i, j**

Ограничения:

Observe supply limit at plant i : $\sum_j x_{ij} \leq a_i$, for all **i** (cases)

Требование на лимит запаса завода i

Satisfy demand at market j : $\sum_i x_{ij} \geq b_j$, for all **j** (cases)

Удовлетворение требований потребителей j

Целевая функция:

Minimize $\sum_i \sum_j c_{ij} x_{ij}$ **($K)**

Минимизировать

Этот простой пример выявляет использование прошлого практического опыта, накопленного в моделировании, который общепризнан хорошим базисом для обучения. И использование этого опыта соответствовало замыслу GAMS.

Во первых, все объекты модели идентифицируются (или группируются) по типам.

Во вторых, последовательность объектов выбирается таким образом, что нет ссылки на символ прежде, чем он будет определен.

В третьих, определяют единицы измерения всех объектов.

В четвертых, единицы измерения выбираются таким образом, чтобы разработчик работал с небольшими по абсолютной величине числами (Символ $K означает здесь тысяча долларов).

Имена типов могут отличаться у разных разработчиков модели. Например, экономисты используют термин "exogenous variable (экзогенная переменная)" и "endogenous variable (эндогенная переменная)" для "given date (заданных данных)" и "decision variable (расчетных переменных)", соответственно. В GAMS установлена следующая терминология: индексы называются **sets**, данные называются **parameter**, расчетные переменные называются **variables**, ограничение целостности, уравнения, неравенства и целевая функция называются **equations**.

Представление в GAMS задачи "оптимизации транспортных перевозок" близко к ее алгебраическому представлению. Однако, самое важное отличие заключается в том, что версия GAMS может быть прочитана и выполнена компьютером.

В нашем примере задачи "оптимизации транспортных перевозок", имеется два консервных завода и три потребителя с нижеприведенными данными (этот пример взят из Dantzig, 1963)

	расстояние от завода по потребителя			Запасы
	Markets (потребители)			
Plants(*заводы)*	**New York**	**Chicago**	**Toreka**	
Seatle	2.5	1.7	1.8	350
San Diego	2.5	1.8	1.4	600
demands(*требование*)	325	300	275	

Расстояние от завода до потребителя дано в тысячах миль, стоимость транспортирования груза $90.00 каждый ящик на каждую тысячу миль.

Для GAMS представляют эту задачу в следующем виде:

```
     Sets

     i  canning plants        /Seattle,  San-Diego/
        консервные заводы
     j  markets               /New-York, Chicago, Toreka/;
        потребители

     Parameters

     a(i)    capaciti of plant  i in cases
             запасы завода i в ящиках
                  /Seattle     350
                   San-Diego   600

     b(j)    demand at market j in cases
             требование потребителя j в ящиках

                /New-York       325
                 Chicago        300
                 Toreka         275    /;

Table d(i,j)  distance in thousands of miles
               расстояние в тысячах миль

                New-York    Chicago     Toreka
Seattle           2.5         1.7         1.8
San-Diego         2.5         1.8         1.4   ;


Scalar  f  freight in dollars per case per thousand miles  /90/;
            фрахт в долларах каждого ящика на каждую тасячу миль

Parameter  c(i,j)  transport cost in 1000s of dollars per case;
                   стоимость транспорта в тыс. долларов каждого ящика

           c(i,j) = f*d(i,j)/1000;

Variables

    x(i,j)  shipment quantities in case
            количество погрузки в ящиках

    z       total transportation costs in 1000s dollars;
            общая стоимость транспорта в тыс. долларов

Positive variable x;

Equations

     cost         define objective function
                  устанавливает целевую функцию

     supply(i)    observe supply limit at plant i
                  соблюдение лимита запаса на заводе  i

     demand(j)    satisfy    demand   at  market j;
```

удовлетворение требований потребителя j

```
cost..      z  =e= sum((i,j), c(i,j)*x(i,j))  ;

supply(i)..    sum(j, x(i,j))   =l=    a(i) ;

demand(j)..    sum(i, x(i,j))   =g=    b(j) ;

Model transport /all/;

solve transport using  lp minimizing z;
display x.l, x,m ;
```

Если вы представите файл, содержащий вышеприведенные операторы, как файл ввода в программу GAMS, то будет скомпилирована и выполнена модель задачи "оптимизация транспортных перевозок". Детали ввода зависят от того, что запрашивает GAMS на различных типах компьютеров, но самый простой ("без лишних сложностей") способ вызвать GAMS - ввести слово **gams** после имени файла ввода (input file). Вы увидите текущую по монитору информацию, описывающую процесс, который выполняет GAMS, включая имя файла, в который начал записываться вывод (output). Если вводный файл не содержал ошибок, то после выполнения GAMSом тестирования вводного файла, внизу появится оптимальное решение нашей задачи "оптимизации транспортных перевозок" в виде следующей записи:

	New-York	**Chicago**	**Toreka**
Seattle	**50.000**	**300.000**	
San-Diego	**275.000**		**275.000**

Также будут получены значения маргинальных стоимостей (симплексных множителей)

	Chicago	**Toreka**
Seattle		**0.036**
San-Diego	**0.009**	

Например, полученные результаты расчетов говорят, что оптимальным решением является: ничего не отправлять из Чикаго в Toreka,
но если вы все таки настаиваете на отправке товаров из Чикаго в Toreka, то отправка одного ящика обойдется дополнительно 0.36$K (или 36$) к оптимальной стоимости [1].

2.2. СТРУКТУРА МОДЕЛИ GAMS

Теперь мы рассмотрим основные компоненты модели GAMS со ссылками к вышеприведенному примеру.

[1] Возникает вопрос, можно ли проверить, является ли полученное решение оптимальным для предложенных данных.
Если вы отправите один ящик из Сиетла в Toreka, тогда, чтобы обеспечить баланс запас/потребности, вы должны отправить на один ящик меньше из Сан-Диего в Toreka, и на один ящик больше из Сан-Диего в Нью-Йорк, и на один ящик меньше из Сиетла в Нью-Йорк. Нетто увеличения стоимости транспортировки груза составит +1800 - 1400+2500-2500=400 миль, стоимость которой составляет 36$ по данным расценкам .

Базовые компоненты GAMS:

Ввод	**Вывод**
• **Sets** (множества) объявление (декларация) присвоение членства • **Data** (данные) (**Parameters, Tables, Scalars**) объявление (декларация) присвоение значения • **Variables** (переменные) объявление (декларация) присвоение типа • Присвоение границ и/или начальных значений (необязательная операция) • **Equations** (уравнения) объявление (декларация) определение • Операторы **Model** и **Solve** • Оператор **Display** (необязательная операция)	• Эхо-принт • Справочная карта • Список уравнений • Состояние отчета • Результаты

Также имеются необязательные вводные компоненты, такие как редакторский контроль за "плохими" данными и требования, чтобы сведения, выдаваемые в отчетах о результатах выполнения модели максимально соответствовали нашим потребностям. Другими наиболее характерными необязательными в применении возможности GAMS являются - сохранение и восстановление старых моделей, создание и рассмотрение нескольких моделей в одной выполняемой модели и т.д.

Но в данном учебном пособии будут рассмотрены только базисные компоненты.

Прежде чем более детально рассмотреть базисные компоненты, следует определить несколько общих положений:

1. Модель GAMS - это набор операторов, написанных на языке GAMS. Внутренне правило управления последовательностью операторов является причиной того, что нельзя ссылаться на модель прежде, чем она будет продекларирована и воспринята компилятором как существующая.

2. Типографические операторы GAMS могут быть представлены почти в любом стиле, который только доступен и удобен пользователю. Допускается использование нескольких строк для одного оператора, вложенные пустые строки и несколько операторов в одной строке. Вы получите хорошее представление и том, что допускается при записи операторов из рассматриваемого примера, но полный перечень правил приводится в разделе 3.

3. Начинающему пользователю GAMS настоятельно рекомендуется оканчивать каждый оператор символом точка с запятой (;), как показано в нашем примере.

4. Компилятор GAMS не делает различий между заглавными и прописными буквами, поэтому можно свободно использовать и те и другие.

5. Документация (имеется ввиду комментарии), которая формируется в результате выполнения математической модели, является решающим фактором в ее полезности. Когда документация вписана в модель, а не написана отдельно после выполнения расчетов, она более полезна и обычно более правильно записана а значит и понята. Существует два способа внесения документации в модель GAMS.

   Первый - компилятором GAMS игнорируется любая линия, которая начинается со звездочки в позиции 1.
   Второй - возможно более важный, текст документации может быть помещен внутри особых операторов GAMS. Все строчные слова в модели "оптимизации транспортных перевозок" являются примером второго способа составления документации.

6. Как можно видеть из вышеприведенной записи компонентов ввода, создание объектов GAMS включает два этапа: объявление (декларацию) и присваивание или определение. "Декларация" означает объявление о существовании чего-либо и этому компоненту дается имя. "Присваивание" или "определение" означает , что этому компоненту дается отдельное конкретное значение или форма. Например, в случае работы с уравнениями, декларация и определение выполняется при помощи разных операторов GAMS. Однако для всех других объектов GAMS существует набор обязательных требований о том как выполнять декларацию и присваивание, а именно - в тех же самых операторах или отдельно.

7. Имена, даваемые объектам модели, всегда должны начинаться с буквы, а затем могут содержать еще до 9 букв или цифр в различном порядке.

2.3. МНОЖЕСТВА (SETS)

Set переводится с английского как множество.

Sets (*множества*) являются основным конструктивным блоком в модели GAMS.Если модель представить в алгебраическом виде, то множества будут соответствовать индексам. Вышеприведенный пример "оптимизация транспортных перевозок" включает в себя только один оператор **Set**.

```
Sets

     i  canning plants        /Seattle,  San-Diego/
     j  markets               /New-York, Chicago, Toreka/;
```

Действие этого оператора очевидно. Продекларировано два множества и им даны имена **i** и **j**. Кроме того присвоены этим множествам следующие конкретные значения-элементы:

```
i = {Seattle, San-Diego}
j = {New York, Chicago, Toreka}
```

Необходимо учитывать типографические различия между форматом (способом записи) в GAMS и традиционным форматом принятом в математике, который используется для записи элементов множества. GAMS для записи множества использует наклонную черту **"/"** , а не фигурные кавычки просто потому, что не все клавиатуры компьютеров имеют фигурные скобки. В GAMS имена,

состоящие из нескольких слов (например, New York) недопустимы, их нужно записывать через дефис (New-York).

Подчеркнутые слова представленные в операторе **Set** есть "текст коментариев". Текст не является обязательным при написании оператора. Он существует только для внутренней документации, не оказывает влияния на выполнение расчетов в модели. Компилятор GAMS не делает попыток интерпретировать текст, но он сохраняет текст и воспроизводит его на различных этапах выполнения программы.

Не обязательно создавать все множества используя только один оператор. Можно создавать каждое множество или группы множеств с помощью отдельных операторов, как это показано ниже:

```
Set     i canning plants     /Seattle ,  San-Diego         /  ;
Set     j markets            /New-York,  Chicago  , Toreka /  ;
```

Пустые промежутки в строке и пустые линии (интервалы), также как использование заглавных и строчных букв выбираются по усмотрению пользователя.

Каждый пользователь GAMS имеет склонность развивать индивидуальную стилистику при написании моделей. (Использовать **set** или **sets** можно по усмотрению пользователя. Использовать **set** в операторе, который выполняет одну декларацию, и **sets** в операторе, который выполняет несколько деклараций - это хороший английский язык, но GAMS трактует единственное и множественное число как синонимы).

Когда элементы множества представляют собой последовательность, удобно использовать при присвоении членства элементам множества свойство GAMS, которое выражается с помощью звездочки '*****'.
Например, операторы GAMS

```
set  t   time periods  /1991*2000/;
set  m   machines      /mach1*mach24/;
```

означают следующее

t = {1991,1992,1993,...,2000}
m = {mach1,mach2,...,mach24}

Необходимо помнить, что элементы множества **set** являются и сохраняются как буквенные характеристики ряда, поэтому, например, элементы множества **t** не являются числами.

Другое удобное свойство GAMS связано с оператором **alias**, который используется для того, чтобы дать ранее продекларированому множеству другое имя. В следующем примере показан вид оператора **alias**

```
alias (t, tp);
```

в математической системе обозначения множество **t** имеет еще одно имя **tp**. Это полезно использовать в моделях, в которых имеется взаимосвязь между элементами одного и того же множества.

В вышеприведенных операторах множества **i, j, t** и **m** – примеры статических множеств, т.е. они присваиваются пользователем непосредственно их членам **set** и не изменяются. GAMS обладает возможностью создавать динамические множества, в которых члены множества получаются при выполнении над **set** -теоретических и логических операций. Динамические множества рассмотрены в разделе 11. Другое ценное свойство GAMS - возможность работы с многомерными множествами, которые рассмотрены в разделе 4.

2.4. ДАННЫЕ (DATA)

В моделе GAMS, рассматривающей задачу "оптимизации транспортных перевозок", демонстрируются все три возможных фундаментальных формата ввода данных:

- списки;
- таблицы;
- непосредственное присвоение.

В следующих трех подразделах по очереди будут рассмотрены каждый из этих трех форматов.

2.4.1. ДАННЫЕ, ВВОДИМЫЕ СПИСКОМ

Первый формат иллюстрируется в примере первым оператором **parameter**, который мы повторяем ниже

```
Parameters

     a(i)    capaciti of plant  i in cases
                  /Seattle     350
                   San-Diego   600

     b(j)    demand at market j in cases
                /New-York       325
                 Chicago        300
                 Toreka         275  /;
```

При использовании этого оператора вы получите некоторые новые знания. Конечно, они очевидны, но все-таки стоит проанализировать их в деталях. Операторы декларируют существование двух параметров, дают им имена **a** и **b** и декларируют, соответственно, их "домены" **i** и **j** (здесь "домен" - множество или кортеж[1] множества, через которые определяются **parameters** (параметры), **variable** (переменные) или **equations** (уравнения). Этот оператор также содержит коментирующий текст для каждого параметра и присваивает значения **a(i)** и **b(j)** для каждого элемента **i** и **j**.

Если вы хотите, то можно разделить один оператор на два отдельных. Ниже приведен пример, как это сделать

```
Parameter
     a(i)    capaciti of plant  i in cases
             /Seattle     350, San-Diego   600 /;


Parameter
     b(j)    demand at market j in cases
                /New-York       325
                 Chicago        300
                 Toreka         275  /;
```

[1] Кортеж, N-ка. - Упорядоченный набор из N элементов.

Ниже приведены несколько правил, которые необходимо знать при использовании форматов **Parameter** .

1. Список элементов домена и соответствующие им значения параметра могут быть представлены почти любым способом, который вам нравится. Только необходимо придерживаться следующих правил записи

-вводимый список должен быть заключен между двумя наклонными чертами (**/**);
-пары элемент-значение разделяются запятыми [вышеприведенный параметр **a(i)**] или каждая пара занимает отдельную строку [параметр **b(j)**].

2. В GAMS символ точка с запятой (**;**), разделяющий список элемент-значение, домен и предшествующий ему текст отсутствует. Это сделано вследствии однотипности операторов которые используются для декларации и присвоения. (Список элемент-значение отдельно на разных строках не интерпретируется GAMSом и результатом будет сообщение об ошибке).

3. Компилятор GAMS обладает необыкновенно ценным свойством, называемым "проверка домена", что позволяет проверить, является ли в действительности элемент каждого домена в списке членом соответствующего множества. Например, если Вы записали правильно **'Seattle'** в операторе, декларирующим **Set i**, но записал **'Seatle'** в последующем списке элемент-значение, то компилятор GAMS выдаст сообщение об ошибке, что элемент **'Seatle'** не относится к множеству **i**.

4. Ноль - значение по умолчанию для всех параметров. Поэтому в список элемент-значение необходимо включать только ненулевые значения. Они могут быть включены в любой последовательности.

5. Скаляр рассматривается как параметр, не имеющий домена. Он может быть продекларирован и присвоен при помощи оператора **Scalar**, содержащий "вырожденный" список и только с одним значением, как это показано ниже операторе модели-примере "оптимизация транспортных перевозок":

Scalar f freight in dollars per case per thousand miles /90/;
фрахт в долларах каждого ящика на каждую тасячу миль

Домен двух и более мерного параметра также может быть введен списочным форматом. Это очень удобно для ввода разбросанных (имеющих мало ненулевых) и супер-разбросанных (имеющих мало отличных от нуля) массивов. Подробно об этом описано в разделе 5.

2.4.2. ДАННЫЕ, ВВОДИМЫЕ В ФОРМЕ ТАБЛИЦ

Программисты, составляющие модели для решения практических задач, заметили, что очень часто в больших моделях вводные данные состоят большого числа соответствующих небольших таблиц чисел. Поэтому при составлении модели очень удобно иметь табличный формат ввода данных. Пример двухмерной таблицы (или матрицы) представлен в модели "оптимизации транспортных перевозок":

Table d(i,j) distance in thousands if miles

	New-York	Chicago	Toreka
Seattle	**2.5**	**1.7**	**1.8**
San-Diego	**2.5**	**1.8**	**1.4**

Действие этого оператора заключается в том, что декларируется параметр **d** и определяются домены, как множество последовательных пар в Декартовом произведении **i** и **j**. В этом операторе также задаются значения **d**, они расположены под соответствующими заголовками. Если в таблице введено пустое пространство, то оно интерпретируется как ноль.

Как и в списочном формате, GAMS выполнит проверку домена на соответствие членам объявленного множества и имен колонок и строк в таблице. Форматы ввода данных с большим числом колонок, чем можно разместить на одной строке и с вводом более чем двухмерных таблиц, освещены в разделе 5.

2.4.3. ДАННЫЕ, ВВОДИМЫЕ НЕПОСРЕДСТВЕННЫМ ПРИСВОЕНИЕМ

Метод ввода данных в формате "непосредственное присвоение" отличается от методов ввода данных в формате список или таблица в том, что метод непосредственного присвоения делит две задачи (декларация параметра и присвоение параметра) между двумя отдельными операторами. Модель "оптимизации транспортных перевозок" содержит в себе следующий пример этого метода:

```
Parameter  c(i.j)  transport cost in 1000s of dollars per case;
           c(i,j) = f*d(i,j)/1000  ;
```

В конце первой линии обязательно должен быть поставлен символ точка с запятой **';'**. Без этого компилятор GAMS воспримет обе линии как часть одного оператора. (GAMS будет не в состоянии различить действительную интерпретацию записи, и вы получите краткое, но полезное сообщение об ошибке.)

Эффект от первого из вышеприведенных операторов заключается в том, что декларируется параметр **c**, определяется домен **(i,j**) и выполняется комментирующий текст. Второй оператор присваивает для **c(i,j)** значение произведения параметров **f** и **d(i,j)**. Естественно, и это допустимо в GAMS, все это возможно если Вы уже выполнили присвоение значений параметрам **f** и **d(i,j)** в предыдущих операторах.

Вышеприведенное непосредственное точечное присвоение относится ко всем парам в домене c. Если необходимо выполнить такое присвоение отдельным элементам в домене, нужно имена этих элементов заключить в скобки. Например

```
c('Seattle','New-Yourk') = 0.40;
```

является допустимым оператором присвоения в GAMS.

Присвоение значения параметрам может быть выполнено больше, чем один раз. Каждый оператор присвоения имеет действие немедленно, поэтому после присвоения параметру нового значения предыдущее значение уже не принимается во внимание (в то время как декларация параметра не может быть выполнена более одного раза). Это сделано специально, чтобы не допустить случайного использования одного имени для двух разных понятий.

Правая сторона оператора присвоения может содержать в себе огромное количество вариантов математических выражений и встроенных функций. Если вы знакомы с каким либо языком программирования, таким как FORTRAN или C, вы не будете иметь затруднений в написании операторов присвоения в GAMS. (Однако, нужно помнить, что GAMS имеет некоторые отличия и от языка FORTRAN, и от языка C. Например, в GAMS можно выполнить присвоение значения **c(i,j)** для всех пар **(i,j)** без структуры "делай цикл".)

Список стандартных операций GAMS и дозволенных функций приведен в таблице 6.1.

Ниже дано несколько примеров действительных присвоений. Во всех случаях предполагается, что параметр с левой стороны уже продекларирован, а параметрам с правой стороны уже выполнено присвоение значений в предыдущих операторах(невидимых здесь операторах).

```
csquared  =  sqr(c);
e  =  m*csquared;
w  =  l/lamda;

eoq(i)  =  aqrt( 2*demand(i)*ordcost(i)/holdcost(i));
t(i)  = min(p(i), q(i)/r(i), log(s(i)))  ;
euclidean(i,j)  = sqrt(sqr(xi(i)  - xi(j)  + sqr(x2(i) -x2(j)));
present(i)  = future(j) * exp(-interest*time(j));
```

В непосредственном присвоении также может быть использовано суммирование и умножение операторов, о которых будет рассказано дальше.

2.5. ПЕРЕМЕННЫЕ (VARIABLES)

В моделе GAMS расчетные (или эндогенные) переменные декларируются при помощи оператора **Variables**. Каждой переменной дается имя, соответствующий домен и (необязательно) комментирующий текст. Модель "оптимизация транспортных перевозок" содержит следующий пример оператора **Variables**

```
Variables
     x(i,j)   shipment quantities in cases
     z        total transportation costs in 1000s of dollars ;
```

С помощью оператора **variables** выполняется декларация переменной перевозки для каждой пары (**i,j**).(В разделе 8 будет показано как GAMS может обращаться с типичной в повседневной практике ситуацией, когда перевозки допустимы только некоторого подмножества полного множества пар **(i,j)**.)

Переменная **z** декларируется без домена, потому что она скалярная величина. Каждая оптимизационная модель GAMS содержит в себе одну такую переменную, служащую в качестве величины, которая минимизируется или максимизируется.

После декларации переменной, ей должен быть присвоен тип. Типы переменных, допустимые в GAMS, приведены в таблице.

Тип переменной	**Допустимые пределы изменения переменной**
Free	Минус бесконечность - плюс бесконечность
Positive	Ноль - плюс бесконечность
Negative	Минус бесконечность - ноль
Binary	0 или 1
Integer	0, 1, ..., 100

Переменная, которая служит в качестве величины, которая должна быть оптимизирована, должна быть скалярной и должна иметь тип **Free**. В нашем примере "оптимизации транспортных перевозок" переменной **z** присваивается тип **Free** по умолчанию, а переменная **x(i,j**) определяется как неотрицательная с помощью следующего оператора

```
Positive variable x;
```

Нужно помнить, что домен переменной **x** не должен быть повторен при присвоении типа переменной. Все вводы в домен автоматически имеют такой же тип переменной.

В разделе 2.10 описано, как присваиваются нижние и верхние границы и как инициализируются значения переменных.

2.6. УРАВНЕНИЯ (EQUATIONS)

Мощь алгебраического языка моделирования, таких как GAMS, проявляется наиболее ярко при составлении уравнений и неравенств. Это связано с тем, что группы уравнений и неравенств имеют алгебраические структуры, когда все члены групп вводятся одновременно, а не индивидуально.

2.6.1. ДЕКЛАРАЦИЯ УРАВНЕНИЙ

Уравнения декларируются и определяются при помощи различных операторов. Формат декларации уравнений такой же как и другие вводы в GAMS. Сначала записывается ключевое слово (в нашем случае **Equations**), затем следует имя, домен и запись одного или группы уравнений или неравенств, которые декларируются. Наш пример (модель "оптимизации транспортных перевозок") содержит следующие декларации уравнений:

```
Equations

     cost         define objective function
     supply(i)    observe supply limit at plant i
     demand(j)    satisfy demand at market j;

cost..                      z    =e=    sum((i,j), c(i,j)*x(i,j))  ;
supply(i)..    sum(j, x(i,j))    =l=    a(i)                       ;
demand(j)..    sum(i, x(i,j))    =g=    b(j)                       ;
```

Нужно помнить, что слово **Equation** имеет широкое значение в GAMS, оно включает связи и в уравнениях, и в неравенствах, и уравнение GAMS с одним именем может представлять одно или несколько таких связей. Например, **cost** не имеет домена, так это простое уравнение, а **supply** представляет множество неравенств, определенных через домен **i**.

2.6.2. СУММИРОВАНИЕ (И УМНОЖЕНИЕ) ВЫРАЖЕНИЙ В GAMS

Прежде чем перейти к разделу "Определение уравнений" мы рассмотрим, как в GAMS можно выполнять операцию "сложение выражений". Нужно учитывать, что GAMS разработан для стандартной клавиатуры и чтения ввода "линия за линией", поэтому невозможно (и это было бы неудобно для пользователя) использовать стандартные математические выражения для суммирования.

В GAMS суммирование выражений может быть использовано и для простых, и для комплексных выражений. Этот формат основывается на идее, что суммирование - это оператор с двумя аргументами:

```
Sum(index of summation, summand)
```

Два аргумента разделяются запятой. Если первый аргумент содержит запятую, тогда он должен быть заключен в круглые скобки. Вторым аргументом может быть любое математическое выражение, включая другое суммирование.

Простой пример - задача "оптимизация транспортных перевозок" содержит следующее выражение

```
Sum(j,  x(i,j))
```

что эквивалентно $\sum_j x_{ij}$

Простое, но более комплексное суммирование представлено в следующем примере

```
Sum(i,j),   c(i,j)*x(i,j))
```

что эквивалентно $\sum_i \sum_j c_{ij} x_{ij}$

Последнее выражение может быть записано в другом виде

```
Sum(i,  Sum(j,  c(i,j) * x(i,j)))
```

В разделах 6 и 8 показано, как использовать оператор 'dollar', чтобы установить ограничения на оператор суммирования таким образом, чтобы в суммирование были включены только те элементы **i** и **j**, которые удовлетворяют выбранным условиям.

Умножение в GAMS определяется так же, как и суммирование, только вместо оператора **Sum** используется оператор **Prod**. Например

```
prod(j,  x(i,j))
```

эквивалентно $\prod_j x_{ij}$.

Операторы суммирования и умножения могут быть использованы в операторе непосредственного присвоения для **parameters**.

Например:

```
Scalar  totsupply  total supply over all plants;
totsupply  = sum(i,  b(i)) ;
```

2.6.3. ОПРЕДЕЛЕНИЕ УРАВНЕНИЙ

Определение уравнений по разнообразию его выражения является самым комплексным оператором в GAMS.

Ниже приведена структура записи (синтаксис) определения уравнения:

- имя уравнения, которое определяется;
- домен;
- условие, ограничивающее домен (не обязательно);
- символ '..' ;
- выражение с левой стороны;
- соответствующий оператор: **=l=**, **=e=** или **=g=;**
- выражение с правой стороны.

В нашем примере "оптимизация транспортных перевозок" используются все три оператора

```
cost..         z  =e= sum((i,j), c(i,j)*x(i,j))  ;
```

```
supply(i)..    sum(j, x(i,j))   =l=    a(i) ;
demand(j)..    sum(i, x(i,j))   =g=    b(j) ;
```

Ниже приведены несколько положений, которые необходимо помнить.

- При помощи одного оператора GAMS можно создать множество уравнений. Этот оператор контролируется доменом. Например, в результате определения ограничения **demand** для каждого элемента домена **j** будет составлено частное ограничение, как показано в нижеследующем фрагменте из файла output модели GAMS "оптимизация транспортных перевозок":

```
demand(new-york)..
x(seattle, new-york) + x(san-diego, new-york)  =g= 325;

demand(chicago)..
x(seattle, chicago)  + x(san-diego, chicago)   =g= 300;

demand(toreca)..
x(seattle, toreka)   + x(san-giego, toreca)    =g= 275;
```

Когда мы выполняем определение уравнения мы в вводном файле записываем в алгебраической форме только общее уравнение, а GAMS из общего уравнения составляет все частные уравнения. (При использовании других оптимизационных пакетов вы можете встретить что-то похожее в части составления вводного файла, но вы не встретите таких возможностей в файле output).

- При решении многих практических задач мы сталкиваемся с ситуацией, когда должны быть исключены или же должны отличаться некоторые члены домена уравнения (т.е. часть доменов описыватся другими уравнениями). GAMS без труда может описать задачу с такими требованиями используя оператор 'dollar' или 'such-that', которые в этом разделе не приводятся, но они подробно описаны в разделе 8. Возможность ограничения домена особо полезна, когда необходимо сохранить размеры модели решающей реальные практические задачи, в пределах ее разрешимости.
- Операторы соотношений в структуре уравнений имеют следующие синтаксис и назначение:

  =l= меньше чем или равно;
  =g= больше чем или равно;
  =e= равно.

- Очень важно понимать различие между символами **"="** и **"=e="**. Символ **"="** используется только в непосредственном присвоении, символ **"=e="** используется только в определении уравнения. Эти два контекста совершенно различны. Непосредственное присвоение задает параметру нужное значение прежде, чем вызывается solver. Определение уравнения также описывает требуемые связи, но эти связи не могут быть удовлетворены до тех пор, пока не вызван solver. Это означает, что определение уравнений должно включать в себя переменные и не должно включать в себя непосредственное присвоение.В этом и состоит различие.
- Переменные могут находиться на левой, правой или на обоих сторонах уравнения. Одна и та же переменная может появиться в уравнении несколько раз, как переменные в обычных уравнениях. Процессор GAMS будет автоматически конвертировать уравнение к его эквивалентной стандартной форме (переменные расположены на левой стороне уравнения и не дублируются) прежде, чем будет вызван solver.
- Определение уравнения может быть размещено в любом месте вводного файла GAMS, при условии, что само уравнение и все переменные и параметры, которые присутствуют в уравнении, предварительно были

продекларированы. (После декларирования параметр уже может быть использован в уравнении, а значение параметру можно присвоить или переприсвоить после определения уравнения. Это очень удобно, когда создаются большие модели, но приходиться работать с одним вводным файлом GAMS.) Не обязательно определять сами уравнения в той же самой последовательности, в которой они были продекларированы.

2.7. ЦЕЛЕВАЯ ФУНКЦИЯ

Необходимо напомнить, что GAMS не имеет полностью определенного объекта, называемого "целевой функцией". Чтобы подробно обозначить функцию, которая будет оптимизироваться, необходимо ввести переменную, которая

- будет иметь тип **Free** (т.е. она не будет иметь ограничение в знаке);
- будет скаляром (т. е. она не будет иметь домен);
- будет содержаться в уравнении, которое будет идентифицировано как целевая функция.

2.8. ОПЕРАТОРЫ MODEL И SOLVE

В GAMS **Model** - это очень узкое понятие. Оно означает состав **Equations** (уравнений). При декларации **Model**, также как и всем другим объектам GAMS, должно быть дано имя.

Синтаксис декларации **Model** следующий: за ключевым словом **Model** следует имя модели, затем следует список имен уравнений, заключенный между двумя наклонными вправо черточками. Если в модель включены все ранее определенные уравнения, то можно вместо всего списка уравнений поставить **/all/** В нашем примере имеется только один оператор **Model**

```
model transport /all/;
```

Если бы мы предпочли использовать полный список уравнений вместо слова **all**, тогда оператор был бы записан следующим образом

```
model transport /cost, supply, demand/ ;
```

Так как домены не являются частью имени уравнения, то при декларации **Model** они опущены в списке уравнений.

Когда подмножество существующих уравнений заключает в себе частную генерируемую модель (подмодель), то список использовать не обязательно.

На первый взгляд оператор **Model** может выглядеть чрезмерным излишеством, но для опытного пользователя он очень удобен, так как с его помощью можно составить несколько разных моделей при одном запуске GAMS.

После того как продекларирована модель и присвоены уравнения, можно вызывать Solver (блок, в котором выполняются оптимизационные расчеты). Вызов **solver**а выполняется при помощи оператора **solve**. В нашем примере "оптимизация транспортных перевозок" он записан в следующем виде

```
solve transport using lp minimizing z;
```

Синтаксис оператора **solve** следующий:

1. Ключевое слово **solve**.
2. Имя модели, которая будет решаться.
3. Ключевое слово **using**.

4. Ключевые символы для обозначения процедуры решения,
   которой соответствует задача .

Полный список процедур решения:

- **lp** - линейное программирование;
- **nlp** - нелинейное программирование;
- **mip** - смешанное целочисленное программирование;
- **rmip** - облегченное смешанное целочисленное программирование;
- **minlp** - смешанное целочисленное нелинейное программирование;
- **rminlp** - облегченное смешанное целочисленное нелинейное программирование;
- **mcp** - смешанная дополнительная задача;
- **cns** – жесткая нелинейная система.

5. Ключевое слово **minimizing** или **maximizing**.
6. Имя переменной, которая должна быть оптимизирована.

2.9. ОПЕРАТОР DISPLAY

Оператор **Solve** вызывает выполнение некоторых последовательных действий, в частности:

- генерируется модель;
- подготавливаются данные в соответствующей структуре, чтобы выполнить ввод задачи в solver;
- выполняется solver;
- результаты записывается из solver в файл (output).

Имеется два способа получения оптимального значения основной и/или двойственной переменной:

- можно посмотреть эти значения в выводе Solver (файл output);
- можно запросить GAMS отобразить результаты на дисплее.

Наш пример "оптимизации транспортных перевозок" содержит следующий оператор

```
display  x.l, x.m;
```

который вызывает на экран найденные значения уровня **x.l** и предельные (или сокращенные до минимума) значения стоимости перевозки **x.m** переменных транспортной перевозки **x(i,j).** GAMS автоматически форматирует данные с расчетными величинами в таблицу с соответствующими названиями в шапке таблицы.

2.10. БАЗА ДАННЫХ ".LO, .L, .UP, .M"

GAMS был задуман таким образом, чтобы использовать системы с небольшими базами данных, в которых выполняются записи для переменных и уравнений. В каждой записи переменной внутри компилятора есть четыре области:

.lo = нижняя граница;
.l = уровень или основное значение;
.up = верхняя граница;
.m = маргинальное или двойственное значение.

Синтаксис для их представления:

- имя переменной или уравнения;
- имя области;
- (если необходимо) домен (или элемент домена).

GAMS предоставляет пользователю полный доступ к написанию и прочтению баз данных. Сейчас, вероятно, вы не оцените эти свойства GAMS, но в будущем, когда у вас появиться опыт в составлении моделей, они могут стать очень полезными. Некоторые примеры использования баз данных будет дан ниже.

2.10.1. ПРИСВОЕНИЕ ПЕРЕМЕННЫМ ГРАНИЦ И/ИЛИ НАЧАЛЬНЫХ ЗНАЧЕНИЙ

По умолчанию нижняя и верхняя границы переменных устанавливаются автоматически в соответствии с типом переменных (**free, positive, negative, binary *или* integer**), однако GAMS позволяет пользователю изменять границы переменных.

Некоторые примеры:

```
x.up(i,j)                    =       capacity(i,j)                ;
x.lo(i,j)                    =    10.0                            ;
x.up('seattle', 'new-york') =    1.2*capacity('seattle','new-york');
```

В первом и третьем примере предполагается, что **capacity(i,j**) является параметром, который предварительно был продекларирован, и ему было присвоено значение. Операторы присвоения границ переменным записываются после декларации переменных, но перед оператором **Solve**. Все математические выражения, доступные для непосредственного присвоения, размещаются с правой стороны.

При составлении моделей в нелинейном программировании очень важно помочь solver как можно уже определить диапазон между нижней и верхней границами. Также очень важно определить начальное решение, от которого solver может начать поиск оптимума. Например, в модели инвентаризации, переменными являются **quantity(i).** Кроме этого известно, что оптимальное решение неограниченной версии задачи - это параметр, называемый **eoq(i).** Как приближение для оптимума задачи мы устанавливаем

```
quantity.l(i) = 0.5*eoq(i);
```

(Если ноль находится внутри заданных границ , то по умолчанию начальное значение принимается равным нулю, в противном случае по умолчанию за начальное значение принимается ближайшее к нулю граничное значение.)

Важно помнить, что области **.lo** и **.up** являются полностью подконтрольны пользователю GAMS, в отличиие от областей **.l** и **.m**, которые могут быть инициализированы пользователем, но затем они контролируются solver.

2.10.2. ТРАНСФОРМАЦИЯ И ОТОБРАЖЕНИЕ ОПТИМАЛЬНЫХ ЗНАЧЕНИЙ

(При первом прочтении этот раздел может быть пропущен.)

После того как через оператор solve вызывается оптимизатор (блок модели, который выполняет оптимизационные расчеты) значения основных и двойственных переменных, которые получились в результате оптимизации, помещаются в базу данных (в зоны **.l** и **.m**). Затем результат может быть прочитан, трансформирован и отображен на дисплее при помощи различных операторов GAMS.

Предположим, что в задаче "оптимизации транспортных перевозок" мы хотим узнать в процентах, насколько требования каждого потребителя удовлетворяются каждым заводом. Тогда после оператора **solve** мы должны ввести

```
parameter pctx(i,j)     perc of market j's demand filled by plant i ;
          pctx(i,j) =   100.0*x.l(i,j)/b(j)                         ;
                  display pctx ;
```

Присоединие этих команд к основной задаче "оптимизации транспортных перевозок" даст следующий результат в output:

pctx perc of market j's demand filled by plant i

	new-york	chicago	toreka
seattle	15.385	100.000	
san-diego	84.615		100.000

Например, введя маргинал, мы в сжатой форме выразим "отношение ограничения", которое обычно встречается в смешанных и чистых задачах. Такие модели линейного программирования связаны с определением оптимального количества нескольких видов сырья, имеющегося в наличие, которое необходимо положить в несколько видов готовой продукции. Пусть **y(i,j)** будет количеством тонн продукции, выпускаемой **j**. Предположим, что "отношение ограничения " - нет продукта, который может включать в себя более 25 % одного компонента, которое записывается следующим образом

```
y(i,j)/q(i)  =l=  .25;
```

для всех **i,j**.
Чтобы сохранить линейность модели, ограничение записывается как

```
ratio(i,j)..   y(i,j)  - 0.25*q(i)  =l=  0.0;
```

и эта запись предпочтительнее, чем запись в форме отношения.

Основная проблема заключается в том, что маргинальное значение **ratio.m(i,j)**, выраженное в форме линейного ограничения не имеет существенного (внутреннего) смысла. Самое большее, это может сказать нам, как много можно извлечь пользы при релаксации линейного ограничения к виду

```
y(i.j)  - 0.25*q(j) =l= 1.0;
```

К сожалению здесь это релаксированное ограничение не имеет реального значения. Ограничение, над которым мы хотим выполнить релаксацию, имеет нелинейную форму отношения в ограничении. Например, мы хотели бы знать маргинальную прибыль от изменения отношения ограничения к

```
y(i,j)/q(j)  =l=  .26  ;
```

В действительности мы можем получить желаемые маргиналы, выполнив следующие преобразования в серии маргиналов

```
parameter amr(i,j)     appropriate marginal for ratio constraint ;
          amr(i,j)  =  ratio.m(i,j)*0.01*q.l(j)                   ;
                  display  amr                                    ;
```

Присвоение, выполненное для **amr**, осуществляет доступ к информации из базы данных и для **.m**, и для **.l**. Целью трансформации является возможность показа, что

```
y(i,j)/q(j)  =l=  0.26;
```

является эквивалентом

```
y(i,j)  - 0.25*q(j)  =l=  0.01*q(j);
```

2.11. ВЫВОД GAMS

По умолчанию GAMS создает длинный и информативный вывод (output) (смотри раздел 10). В данном учебном пособии вывод рассматривается частично, и только его главные компоненты:

Эхо принт	или	Эхо принт
Сообщения об ошибках		Справочная карта
Справочная карта		Список уравнений
		Статистика модели
		Состояние отчета
		Отчет о решении

Значительная доля лишних несбывшихся ожиданий вызвана учебниками и пользовательскими пособиями, которые дают читателю ошибочное представление о том, что безупречное использование усовершенствованного программного обеспечения должно быть легким для любого программиста с положительным опытом. GAMS создавался с пониманием того, что даже самый опытный пользователь может допускать ошибки. GAMS пытается выявить ошибки как можно быстрее и минимизировать частоту их появления.

2.11.1. ЭХО-ПРИНТ

Так или иначе ошибки препятствуют началу решения любой оптимизационной задачи.

Первый шаг вывода GAMS - это "эхо-принт" и он заключается в копировании вашего вводного файла. Для дальнейшего использования GAMS сам проставляет с левой стороны "эхо-принт" номера линий (строк). Но наш пример "оптимизация транспортных перевозок" не содержит ошибок, поэтому "эхо-принт" выглядит следующим образом:

```
3
4    Sets
5         i  canning plants        /Seattle,  San-Diego/
6         j  markets               /New-York, Chicago, Toreka/;
7
8    Parameters
9
10        a(i)    capaciti of plant  i in cases
11                     /Seattle     350
12                      San-Diego   600
13
14        b(j)    demand at market j in cases
15                    /New-York       325
16                     Chicago        300
17                     Toreka         275    /;
18
19   Table d(i,j)  distance in thousands of miles
20                   New-York    Chicago     Toreka
21   Seattle           2.5         1.7         1.8
22   San-Diego         2.5         1.8         1.4   ;
23
24   Scalar  f  freight in dollars per case per thousand miles  /90/;
25
26   Parameter  c(i,j)  transport cost in 1000s of dollars per case;
27
28        c(i,j) = f*d(i,j)/1000;
29
30   Variables
31       x(i,j)  shipment quantities in case
```

```
32       z       total transportation costs in 1000s dollars;
33
34   Positive variable x;
35
36   Equations
37        cost         define objective function
38        supply(i)    observe supply limit at plant i
39        demand(j)    satisfy demand at market j;
40
41   cost..      z  =e= sum((i,j), c(i,j)*x(i,j))  ;
42
43   supply(i)..    sum(j, x(i,j))   =l=    a(i) ;
44
45   demand(j)..    sum(i, x(i,j))   =g=    b(j) ;
46
47   Model transport /all/;
48
49   solve transport using  lp minimizing z;
50
51   display x.l, x,m ;
```

Причина, по которой "эхо-принт" начинается с линии номер три, а не с линии номер 1, заключается в том, что вводный файл содержит в себе два оператора "доллар-принт-контроль". Они контролируют распечатку вывода, но так как эти операторы не имеют ничего общего с оптимизационной моделью, то они опущены в "эхо-принт". Запись оператора "доллар-принт-контроль" должна начинаться с первого столбца экрана и выгдядеть как записано ниже.

```
$title a transportation model
$offupper
```

Оператор **$title** служит для того, чтобы последующий текст (в нашем случае "**a transportation model**") был напечатан в верхней строке каждой страницы вывода. Оператор **$offupper** необходим для эхо, которое содержит смешанные буквы - прописные верхнего регистра и строчные нижнего регистра. Более подробная инструкция дана в разделе 10 и в приложении B.

2.11.2. СООБЩЕНИЯ ОБ ОШИБКАХ

Когда в вводном файле компилятор GAMS наталкивается на ошибку, он включает закодированные сообщения об ошибках внутри "эхо-принт" на линии, следующей немедленно после места обнаружения неточности. Эти сообщения всегда начинаются с четырех звездочек (********) и содержат символ **$**, расположенный непосредственно под точкой, где компилятор предполагает наличие ошибка. За символом **$** следует численный код ошибки, описание которой будет дано после "эхо-принт".

Рассмотрим несколько примеров.

Пример 1. Ввод оператора

```
set  q  quarterly time period /spring, sum, fall, wtr/;
```

результатом в "эхо-принт" будет

```
1   set  q  quarterly time period /spring, sum, fall, wtr/;
****                                      $160
```

В этом случае компилятор GAMS выявляет, что что-то неправильно с элементом множества **sum**. Внизу "эхо-принт" мы видим интерпритацию ошибки кода 160:

```
ERROR MESSAGES
160   UNIQUE ELEMENT EXPECTED (ожидаемый уникальный элемент)
```

Проблема заключается в том, что слово **sum** – зарезервированное слово, обозначающее суммирование, поэтому наш элемент должен иметь уникальное (не зарезервированное) имя, предположим **'summer'**. Это обычная ошибка начинающего пользователя. Полный список зарезервированных слов приведен в таблице 3.2.

Пример 2.

Другая обычная ошибка - это пропуск точки с запятой, предшествующей непосредственному присвоению или определению уравнения. Предположим, что в нашем примере "оптимизация транспортных перевозок" перед присвоением **c(i,j)** мы пропустили точку с запятой , как показано ниже

```
Parameter  c(i,j)  transport cost in 1000s of dollars per case
    c(i,j) = f*d(i,j)/1000;
```

Результат вывода (output) будет следующим:

```
17                          c(i,j) = f*d(i,j)/1000;
                                $97    $195$96$194$1

ERROR MESSAGES
1    REAL NUMBER EXPECTED
96   BLANK NEEDED BETWEEN IDENTIFIER AND TEXT
          (- OR - ILLEGAL CHARACTER IN INENTIFIER)
          (- OR - CHECK FOR MISSING ';' ON PREVIOUS LINE)
97   EXPLANATORY TEXT CAN NOT START WITH '$', '=', or '..'
          (-OR-CHECK  FOR MISSING ';' ON PREVIOUS LINE)

194  SYMBOL REDEFINED
195  SYMBOL REDEFINED WITH A DIFFERENT TYPE
```

Такая ситуация, когда на незначительную неточность вы получаете 5 сообщений об ошибке, не будет исключением. Этот пример позволяет определить правило: сконцентрируйтесь на первой ошибке и игнорируйте остальные. Первая выявленная ошибка на линии 17 была с кодом 97, и она указывает, что GAMS думает, что символы на линии 17 являются продолжением комментирующего текста конца линии 16, и не являются непосредственным присвоением, как в принципе желали бы мы. Сообщение об ошибке также уведомляет нас о необходимости проверить предыдущую линию на наличие точки с запятой.

Однако не нужно ожидать, что сообщения об ошибках всегда будут так точно их описывать. Компилятор не может читать ваши мысли. Возможностей компилятора недостаточно, чтобы охватить все ваши замыслы, поэтому необходимо научиться выявлять причины ошибок на основе подсказок, которые даются в выводе GAMS. Например, отсутствие точки с запятой может быть выявлено, если проанализировать ввод **c** по справочной карте (она будет рассмотрена в следующем разделе) и заметить, что для **c** присвоение не выполнялось.

```
SYMBOL     TYPE REFERENCES
C          PARAM DECLARED        15  REF   17
```

Пример 3.

Часто причинами ошибок являются просто орфографические ошибки, и они выявляются раньше, чем они могут внести проблему в решение задачи.

Предположим, что слово **"Seattle"** в таблице записано иначе, чем оно было представлено в декларации множества, поэтому мы получим следующее сообщение об ошибке

```
3
4    Sets
5         i  canning plants        /Seattle,  San-Diego/
6         j  markets               /New-York, Chicago, Toreka/;
7
8    Table d(i,j)  distance in thousands of miles
9                    New-York    Chicago     Toreka
10   Seatle            2.5         1.7         1.8
****    $170
11   San-Diego         2.5         1.8         1.4   ;

    ERROR MESSAGES
    170 DOMAINVIOLATION FOR ELEMENT
```

Пример 4.

Подобно предыдущему примеру, если мы случайно введем **dem(j)** вместо **b(j),** как это требуется с правой стороны ограничения, в результате будет следующее сообщение об ошибке

```
    45  demand(j)..  sum(i, x(i,j))  =g=  dem(j)  ;
                                           $140

    ERROR MESSAGES
    140  UNKNOWN SYMBOL, ENTERED AS PARAMETER
```

Пример 5.

Следующий пример - это пример математической ошибки, которую иногда делают новички модельеры, и которые GAMS адаптирован улавливать. Следующее выражение является математически непоследовательным и, следовательно, не является интерпретируемым оператором

Для всех **i** $\sum_i x_{ij} = 100.$

Это уравнение содержит две ошибки, обе относятся к контролирующим индексам. Индекс **i** является надконтрольным, а индекс **j** - подконтрольным.

Можно видеть, что индекс **i** дает конфликтующую последовательность. Появление в определении количества "для всех **i**" предполагает, что **i** остается фиксированным для каждого уравнения из целой группы. Но появление символа суммирования предполагает изменение. Эти условия не могут быть выполнимыми для обоих случаев. С другой стороны, в любом случае индекс **j** не может быть контролирующим. Таким образом, мы не можем знать, какое из этих возможных значений **j** использовать.

Если мы введем в GAMS уравнение **meaningless** с ошибкой, то обе ошибки будут правильно диагнозцированы.

```
    meaningless(i)..       sum(i,    x(i,j)   =e=  100;
    ****                       $125      $149

    ERROE MESSAGES
    125  SET IS UNDER CONTROL ALREADY   [This refers to set i]
    149  uncontrolled set entered as constant  [This refers to set j]
```

Более подробную информацию о сообщениях об ошибках вы найдете в разделе 10.

Всестороннее выявление ошибок и хорошо написанные сообщения об ошибках оказывают большую помощь в создании моделей.

2.11.3. СПРАВОЧНАЯ КАРТА

Следующий раздел вывода, который, если выявлены ошибки, является последним, представляет собой две "справочные карты", включающие в себя краткое изложение и анализ вводного файла и предназначенные для отладки и составления документации.

Первая справочная карта - это "перекрестная справочная карта" - такая же, как в самых современных компиляторах. Она представляет собой перекрестный справочный список всех объектов модели (**sets, parameters, variables, equation**), расположенных в алфавитном порядке. В списке показан тип каждого объекта и даны закодированные справки о каждом появлении объекта в вводном файле. Перекрестная справочная карта в нашей задаче "оптимизация транспортных перевозок" выглядит следующим образом.

SYMBOL			TYPE	REFERENCES				
A	PERAM	DECLARED	10	DEFINED	11	REF	43	
B	PARAM	DECLARED	14	DEFINED	15	REF	45	
C	PARAM	DECLARED	26	ASSIGNED	28	REF	41	
COST	EQU	DECLARED	37	DEFINED	41	IMPL-ASN	49	
		REF	47					
D	PARAM	DECLARED	19	DEFINED	19	REF	28	
DEMAND	EQU	DECLARED	39	DEFINED	45	IMPL-ASN	49	
		REF	47					
F	PARAM	DECLARED	24	DEFINED	24	REF	28	
I	SET	DECLARED	5	DEFINED	5	REF	10	
		19	26	28	31	38	2*41	
		2*43	45	CONTROL	28	41	43	
		45						
J	SET	DECLARED	6	DEFINED	6	REF	14	
		19	26	28	31	39	2*41	
		43	2*45	CONTROL	28	41	43	
		45						
SUPPLY	EQU	DECLARED	38	DEFINED	43	IMPL-ASN	49	
		REF	47					
TRANSPORT	MODEL	DECLARED	47	DEFINED	47	IMPL-ASN	49	
X	VAR	DECLARED	31	IMPL-ASN	49	REF	34	
		41	43	45	2*51			
Z	VAR	DECLARED	32	IMPL-ASN	49	REF	41	
		49						

Например, перекрестный справочный список говорит нам, что символ **A** является параметром, который был продекларирован на линии номер пять, определен (параметру было присвоено значение) на линии номер 11, и на него имеется ссылка на линии номер 43. Символ **I** имеет более сложный ввод в перекрестном справочном списке. Показано, что множество было продекларировано и определено на линии номер 5. На него имеются ссылки по одному разу на линиях 10, 19, 26, 28, 31, 38, 45 и дважды ссылки на линиях 41 и 43. Множество **I** также использовалось как контролирующий индекс в суммировании, определении уравнения или непосредственном присвоении параметра на линиях 28, 41, 43 и 45.

Для новичка в GAMS детальный анализ перекрестной справочной карты может быть не очень важным. Возможно, наибольшую пользу он или она получат от

справочной карты, выявляя неожиданные объекты, которые были введены в модель из-за ошибок синтаксиса или пунктуации.

Вторая часть справочной карты - это список объектов модели, сгруппированных по типу и записанные с их комментирующими текстами. Список объектов модели в нашем примере "оптимизация транспортных перевозок" следующий:

```
sets

i             canning plants
j             markets

parameters

a             capacity of plant i in cases
b             demand at market j in cases
c             transport cost in 1000s of dollars per case
d             distance in thaousand of miles
f             freight in dollars per case per thousand miles

variables

x             shipment quantities in case
z             total trnsportation costs in 1000s of dollars

equations

cost          define objective function
demand        satisfy demand at market j
supply        observe supply limit at plant i

models

transport
```

2.11.4. СПИСОК УРАВНЕНИЙ

Когда вводный файл уже не содержит компиляционных ошибок (т.е. компиляция пройдена успешно), тогда GAMS может начать генерировать модель. При этом всегда остается вопрос, на который может ответить только разработчик модели - генерирует ли GAMS такую модель, какую предполагалось сделать.

Список уравнений является, возможно, лучшим изобретением при изучении этого сверх важного вопроса.

Список уравнений - это результат оператора **Solve**. Список уравнений демонстрирует нам частные случаи, которые создаются, если текущие значения множеств и параметров подставить в общие алгебраические формулировки модели. Например, общее требование ограничения, задаваемое в водном файле для нашего примера "оптимизация транспортных перевозок" выглядит следующим образом:

```
demand(j)..  sum(i, x(i,j)) =g= b(j) ;
```

в то время как, список уравнений для каждого частного ограничения выглядит следующим образом:

```
--------demand      =g=   satisfy demand at market j

demand(new-york)..
```

```
x(seattle, new-york) + x(san-diego, new-york)  =g=  325;

demand(chicago)..
x(seattle, chicago) + x(san-diego, chicago)  =g=  300;

demand(toreka)..
x(seattle, toreka) + x(san-diego, toreka)  =g=  275;
```

По умолчанию выводится максимум три частных уравнения для каждого общего уравнения. Чтобы изменить количество уравнений, выводимых по умолчанию, в вводном файле перед оператором **solve** необходимо ввести оператор

```
option limrow = r;
```

где **r** - желаемое количество выводимых частных уравнений.

Вывод по умолчанию содержит также раздел, называемый "списком столбца". Он аналогичен списку уравнений, но показывает коэффициенты трех частных переменных для каждой общей переменной. Этот список будет особенно полезным при проверки модели GAMS, которая предварительно была выполнена в формате MPS. Для того, чтобы изменить число частных колонок, подставляемых по умолчанию в каждую общую переменную, вышеприведенная команда должна быть расширена:

```
option limrow  =r,  limcol = c;
```

где **c** - желаемое количество колонок, которое будет подставлено в переменную.
(После того, как модель отлажена, рекомендуем присваивать **limrow** и **limcol 0**. Это позволяет уменьшить количество страниц выводного файла.)

В нелинейных моделях список уравнений GAMS показывает приближения Тейлора первого порядка нелинейных уравнений. Приближения принимаются в качестве начальных значений переменных.

2.11.5. СТАТИСТИКА МОДЕЛИ

Последний раздел вывод, который создает GAMS перед вызовом solver, - это это группа статистических данных о размере модели, как показано ниже для нашей задачи "оптимизация транспортных перевозок".

```
MODEL  STATISTICS

BLOCKS OF EQUATIONS      3       SINGLE EQUATIONS     6
BLOCKS OF VARIABLES      2       SINGLE VARIABLES     7
NON ZERO ELEMENTS       19
```

BLOCK показывает количество общих уравнений и переменных. **SINGLE** отсылает к отдельным строкам и колонкам в частном примере модели, которая начинает генерироваться. Для нелинейных моделей, чтобы описать степень нелинейности задачи, приводится другая статистика.

2.11.6. СОСТОЯНИЕ ОТЧЕТОВ

После того как выполнен solver, GAMS выдает на печать краткое "**Solver summary**" ("заключение по решению"), в котором двумя самыми важными разделами являются **SOLVER STATUS** и **MODEL STATUS**. Для нашей задачи

"оптимизация транспортных перевозок" **solve summary** выглядит следующим образом:

```
               S O L V E     S U M M A R Y

     MODEL      TRANSPORT           OBJECTIVE     Z
     TYPE       LP                  DIRECTION     MINIMIZE
     SOLVER     BDMLP               FROM LINE     49

     ****  SOLVER STATUS       1 NORMAL COMPLETION
     ****  MODEL STATUS        1 OPTIMAL
     ****  OBJECTIVE VALUE               153.6750

     RESOURCE USAGE, LIMIT         0.168       1000.000
     ITTERATION COUNT, LIMIT       2           1000
```

Состоянию отчетов предшествует строка из четырех звездочек "****", как в сообщениях об ошибке, поэтому необходимо выработать привычку изучать все случаи появления строки с "****", где бы они ни появлялись в выводном файле. Требуемое состояние solver – **1 NORMAL COMPLETION**, но возможны 5 других сообщений, которые указывают на различные типы ошибок и неудач. Они подробно рассмотрены в разделе 10.

Существует одинадцать возможных состояний модели, включая обычное состояние завершения линейного программирования (**1 OPTIMAL, 3 UNBOUNDER, 4 INFEASIBLE**), и другие, относящиеся к нелинейному и целочисленному программированию.

В нелинейном программировании состояние, заставляющее продолжить поиск - это состояние **2 LOCALITY OPTIMAL**. Большинство программных средств, которые могут гарантировать решение задач нелинейного программирования - это программы нахождения локального оптимума. Ответственным за анализ выпуклости задачи и требования того, чтобы определить является ли локальный оптимум глобальным оптимумом, является пользователь.

В целочисленном программировании состояние, чтобы продолжать поиск искать является состоянием **8 INTEGER SOLUTION**, которое означает, что вероятное целочисленное решение найдено. Или другими словами, встречает ли решение соответствующее абсолютное оптимальное допустимое отклонение, которое было определено пользователем.

2.11.7. ОТЧЕТЫ О РЕШЕНИИ

Если состояние solver и состояние модели приемлемы, то можно переходить к изучению результатов оптимизации. Результаты расчетов представлены в стандартном формате вывода (output) математического программирования, который имеет дополнительные свойства, заключающиеся в том, что строки и колонки сгруппированы и помечены согласно именам, которые присваиваются только для решенных частных моделей. В этом формате приводятся распечатки каждой строки и колонки дающих значения нижнего предела, уровня, верхнего предела и маргинала.

Строка вывода группируется по блоку общего уравнения, а колонки вывода - по блоку общей переменной. Имена элементов множеств введены в вывод (output) для более легкого чтения. Вывод solver в нашем примере "оптимизация транспортных перевозок" для уравнения **supply(i)** и **demand(j)** и переменной **x(i,j)** показан ниже:

```
---- EQU  SUPPLY     observe supply limit at plant i
```

```
                   LOWER    LEVEL     UPPER         MARGINAL

SEATTLE            -INF     350.000   350.000       EPS
SAN-DIEGO          -INF     350.000   600.000        .


---- EQU DEMAND         satisfy demand  at market j

                   LOWER       LEVEL     UPPER      MARGINAL

NEW-YORK           325.000     325.000   +INF       0.225
CHICAGO            300.000     300.000   +INF       0.153
TOREKA             275.000     275.000   +INF       0.126

---- VAR X      shipment  quantities  in case

                          LOWER    LEVEL      UPPER    MARGINAL

SETTLE   .NEW-YORK          .       50.000    +INF       .
SETTLE   .CHICAGO           .      300.000    +INF       .
SETTLE   .TOREKA            .          .      +INF      0.036
SAN-DIEGO.NEW-YORK          .      275.000    +INF       .
SAN-DIEGO.CHICAGO           .          .      +INF      0.009
SAN-DIEGO.TOREKA            .      275.000    +INF      +INF
```

Одиночные точки в выводе (output) - это нули. Запись **EPS**, сокращенное "эрсилон" - есть очень маленькая, но не равной нулю величина. В нашем случае EPSEps означает вырождение. (Нежесткая переменная ограничения запасов Seattle представлена в базисе нулевым уровнем. Маргинал маркируется как EPS. Такая маркировка предпочтительнее, чем маркировать его как ноль, если необходимо выполнить перезапуск оптимизации, основываясь на старом базисе).

Если результаты solver содержат в себе или какие-то "невозможности", или маргинальные «теневые цены» неправильного знака, тогда эти "нарушающие" корректность объекты, соответственно, выделяются при помощи **INFES** или **NORT**. Если задача завершается не ограниченным продвижением к оптимуму, тогда строки и колонки, соответствующие максимальному радиусу поиска, помечаются **UNBDN (без границ)**.

Заключительная часть отчета Sover - это очень важный "**report summary**" ("краткий отчет"), в котором приведены результаты подсчета общего числа неоптимальных (**NONOPT**), "невозможных" (**INFEASIBLE**), неограниченных (**UNBOUNDED**) строк и колонок. В нашем примере "оптимизация транспортных перевозок" краткий отчет показывает, что все результаты подсчета равны нулю:

```
**** REPORT SUMMARY:       0    NONOPT
                           0    INFEASIBLE
                           0    UNBOUNDED
```

После того, как отчет solver записан, управление возвращается из solver обратно в GAMS. Все уровни и маргиналы, полученные solver, вводятся в базу данных GAMS в области **.l** или **.m**. Затем эти значения могут быть трансформированы и выведены на экран в любом желаемом отчете. Как уже известно, пользователь только перечисляет соответствующие количества, которые будут отображены на экране, а GAMS автоматически форматирует и отмечает соответствующие массивы.
Например, вводный оператор

```
display x.l, x.m;
```

результатом которого будет следующий вывод

```
----     73     VARIABLE   X.L    shipment quantitis in cases

                NEW-YORK       CHICAGO      TOREKA

SEATTLE          50.000         300.000
SAN-DIEGO       275.000                     275.000


----     73     VARIABLE   X.M    shipment quantitis in cases

                 CHICAGO      TOREKA

SEATTLE                       0.036
SAN-DIEGO        0.009
```

Как видно из справочной карты, списка уравнений, отчета о решении и необязательного вывода на экран GAMS сохраняет комментирующий текст и переносит его в вывод (output), чтобы помочь сохранить хорошо документированную модель.

2.12. ЗАКЛЮЧЕНИЕ

Это краткое учебное пособие продемонстрировало возможности GAMS, которые позволяют создавать практические модели оптимизации быстро и эффективно. Следующее обсуждение суммирует преимущества использования языка алгебраического моделирования GAMS по сравнению с матричным генератором или диалоговым Solver.

1. Используя систему обозначения, базирующуюся на алгебраической системе обозначения, можно описать оптимизационную модель для компьютера также просто и легко, как это делает любой другой человек, имеющий математическую подготовку.

2. Так как алгебраическое описание задачи имеет всеобщность, большинство операторов модели GAMS являются снова пригодными к использованию, когда новые решения являются результатом таких же или похожих задач. Это особенно важно в современном мире, где модели постоянно изменяются.

3. Из-за того, что все множества составляются в одном операторе, сохраняется время и уменьшается количество ошибок.

4. Можно сохранить время и уменьшить количество ошибок, если вводить данные в формулу для расчета вместо ввода данных непосредственно.

5. Модель является самодокументирующейся. Так как задача развертывания модели и документирование модели может выполняться одновременно, то составитель модели вынужден быть более аккуратным при составлении документации и выводе данных.

6. Вывод GAMS легко читаем и используем. Отчет решения solver автоматически переформатируется таким образом, чтобы родственные уравнения и переменные группировались вместе в соответствии с метками. А команда **display** позволяет без труда модифицировать и табулировать результаты.

7. Если вы обучаете или изучаете моделирование, вы можете выгадать от требований компилятора GAMS, чтобы каждое уравнение было математически последовательным. Даже если вы являетесь опытным составителем моделей, Вам доступны сотни способов выявления ошибок, что позволяет значительно сократить время отладки модели.

8. С помощью долларового оператора и других передовых возможностей, которые не приведены в этом кратком учебном пособии, можно эффективно выполнять крупномасштабные модели.

Частные применения долларового оператора включают следующее:

- Он может накладывать логическое ограничение на допустимые комбинации индексов для переменных и уравнений, которые были включены в модель. Посредством этого можно выделить лишние строки и колонки и сохранить размер задачи в пределах разрешимости.

- Он может быть использован для составления комплексных суммирований и умножений, которые затем могут быть использованы в уравнениях или настроенных отчетах.

- Он может быть использован для выдачи предостерегающих сообщений или для преждевременного условного завершения при контекстно-специфичных данных редакторов.

ПРОГРАММЫ GAMS 3

3.1. ВВЕДЕНИЕ

Этот раздел позволяет ознакомиться со структурой языка GAMS и его компонентами. Мы еще раз подчеркиваем, что GAMS - это язык программирования и программы, что бы его использовать, должны быть написаны на этом языке. Программа GAMS заносится в дисковый файл, который обычно создается при использовании любого текстового редактора, выбранного пользователем . Когда "работает" GAMS , файл, содержащий программу (вводный файл – файл input), запускается для выполнения. После завершения выполнения файла, появляется результат в выводном файле (файле output), который может быть просмотрен также при помощи текстового редактора. На многих машинах, пока GAMS "прогоняет" через себя модель , на экране появляется несколько сжатых сообщений, информирующих пользователя о ходе выполнения программы и выявлении ошибок. Внимательное изучение выводного файла,с целью адекватной оценки результатов и диагнозцировки ошибок является первейшая обязанность пользователя.

Во время первого или ознакомительного прочтения этот раздел может быть пропущен: обсуждение специальных разделов языка (разделы 4-12) не предполагает знание и понимание этого раздела.

3.2. СТРУКТУРА ПРОГРАММЫ GAMS

Программа GAMS состоит из одного или более операторов, которые определяют структуры данных, начальные значения, преобразование данных, символьные связи (уравнения). Несмотря на то, что последовательность, в которой располагаются операторы, не фиксированна, последовательность, в которой выполняется преобразование данных, важна. Символы должны быть декларированы прежде, чем они будут использоваться, и им должно быть присвоено значение прежде, чем на них будет выполнена ссылка в операторах присвоения. Каждый оператор следует за точкой с запятой, кроме последнего оператора, где точка с запятой необязательна.

3.2.1 ФОРМАТ ВВОДА GAMS

Формат ввода GAMS свободный. Операторы могут быть помещены в любое место на линии, на одной линии может быть расположено несколько операторов или оператор может занимать несколько строк, как показано ниже:

```
Statement;
statement;
statement;  statement;   statement;
    the words that you are now reading is an example of
    very long statement which is stretched over two lines;
```

Перевод дан ниже

оператор;
оператор;
оператор; оператор; оператор;
слова которые ты сейчас читаешь являются
длиным оператором занимающим две строки;

Свободные места и конец строки обычно могут быть свободно использованы между отдельными символами и словами. GAMS не восприимчив к наборной кассе букв, поэтому заглавные и прописные буквы могут быть свободно перемешаны и GAMS воспримет их одинаково (не делая различия между прописными и заглавными буквами). На одной строке можно поместить 120 литер. Для более удобного восприятия текста можно оставлять полностью пустые строки.

Не все строки являются частью языка GAMS. Чтобы указать, что строка не относятся к языку GAMS на первой позиции строки используется один из двух символов: символ звездочка **"*"** или символ доллар **"$"**. Символ звездочка **"*"** в начале строки (в позиции 1) означает, что линия не может быть выполнена и воспринимается как комментарий. Символ доллар **"$"** также в позиции 1 означает, что в остальной части линии помещены параметры компиляции.

3.2.2. КЛАССИФИКАЦИЯ ОПЕРАТОРОВ GASM

Каждый оператор GAMS может быть отнесен к одной из двух групп:
- операторы декларации и определения;
- операторы выполнения.

Оператор декларации описывает класс символа. Часто начальные значения обеспечиваются декларацией, тогда это может быть названо определением. Спецификация символьных связей для уравнений является определением. Операторы выполнения являются инструкциями, чтобы выполнять такие действия, как трансформация данных, решение модели и генерация отчета.

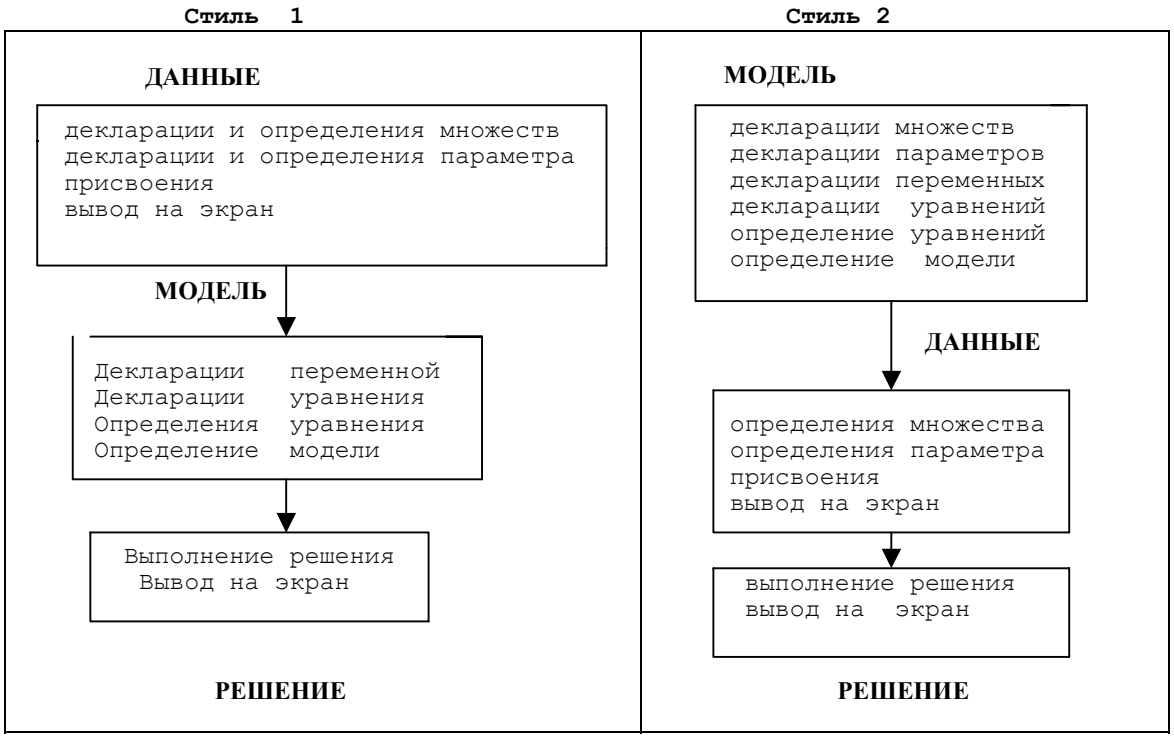
Декларирующие операторы	**Операторы выполнения**
acronym parameter variable set scalar model alias table equation declaration equation definition	option while assignment solve for loop execute repeat display abort

Хотя в программе GAMS имеется большая свобода в выборе последовательности размещения операторов, определенная последовательность все-таки используется. Две наиболее часто используемые структуры программ рассмотрены в следующем подразделе.

3.2.3. ОРГАНИЗАЦИЯ ПРОГРАММ GAMS

Два самых распространенных способа организации программы GAMS приведены ниже. В первом методе сначала размещаются данные, затем модель, а затем операторы решения. В этом способе организации программы множества помещены вначале. Затем данные определяются при помощи операторов **parameter(s), scalar(s) и table(s)**. Следующей определяется модель при помощи операторов декларации переменных и уравнений, определения уравнения и оператора **model**. В заключении модель решается, а результаты выводятся на экран.

Второй стиль придает особое значение модели и помещает ее перед данными. Это особенно удобная последовательность, так как модель может быть решена повторно с другими множествами данных. Здесь имеется разделение между декларацией и определением.



Например, множества и параметры могут быть продекларированы первыми при помощи операторов

```
set  c "crops" ;
parameter   yield  "crops yield" ;
```

и затем определены позже при помощи операторов

```
set c   /wheat, clover, beans /;

parameter yield /wheat  1.5
                clover 6.5
                beans  1.0  /;
```

Первый оператор декларирует, что идентификатор **c** является множеством, а второй определяет элементы множества.

Множества и параметры, используемые в уравнениях, должны быть продекларированы прежде, чем начинается работа с уравнениями; однако они могут быть определены и после спецификаций уравнений, но прежде, чем спецификация уравнения будет использована в операторе **solve**. Это дает программам GAMS значительную организационную гибкость.

3.3. ТИПЫ ДАННЫХ И ОПРЕДЕЛЕНИЙ

В GAMS имеется шесть базисных типов данных и каждый символ или идентификатор должен быть продекларирован, чтобы отнести его к одной из следующих групп:

- множества (sets)
- параметры (parameters)
- переменные (variables)
- уравнения (equations)
- модели (models)

- **Scalars** и **tables** не являются отдельными типами данных. Они являются стенографичным способом, позволяющим удобно декларировать символы, относящиеся к типу parameter. Используется удобный, практичный формат для инициализации численных данных.

Определения имеют обычные характеристики, например:

parameter	**a**	**(i,j)**	**input-output matrix**
data-tipe-keyword	**identifier**	**domain list**	**text**
ключевое слово	*идентификатор*	*список доменов*	*коментарий*

Список доменов и коментарий - это всегда необязательные характеристики.

Другой пример:

```
set                time       time period                    ;
model              turkey     turkish fertilizer model       ;
variables  x,y,z                                             ;
```

В последнем примере несколько идентификаторов (разделенные запятыми) декларируются в одном операторе.

3.4. РАЗДЕЛЫ ЯЗЫКА

Прежде чем продолжить ознакомление с деталями языка, необходимо определить несколько базисных символов и установить правила их выражения и написания в GAMS. Эти базисные символы часто называются лексическими элементами или формой составления блоков языка.

Это:

- **characters**
  знаки
- **reserved words and tokens**
  зарезервированные слова и знаки
- **identifiers (indents)**
  идентификаторы
- **labels**
  метки
- **text**
  текст
- **number**
  число
- **delimiters**

ограничители

- **comments**
  коментарии

Каждый из этих пунктов обсуждается в деталях в следующих подразделах.

Как было указано выше, мы можем использовать любую комбинацию прописных и заглавных букв, потому что GAMS не делает различий между прописными и строчными буквами.

3.4.1. ЛИТЕРЫ

Некоторые литеры в программе GAMS недопустимы из-за того, что они недопустимы или двусмысленны на некоторых машинах. В основном недопустимыми являются все небуквенные и контрольные литеры. Место, где любой литер допустим - блок **"ontext-oftext"**, который рассматривается в разделе 3.4.8. Полный список допустимых литер приведен ниже. Большинство не общепризнанных литер препинания не являются частью языка GAMS, но могут быть свободно использованы в текстах и коментариях.

от **A** до **Z**	алфавит	-	минус
от **a** до **z**	алфавит	**()**	круглые скобки
&	амперсанд	**[]**	квадратные скобки
*****	звездочка	**{}**	фигурные скобки
@	at	%	процент
****	обратная черта	от **0** до **9**	цифры
:	двоеточие	**#**	знак фунта стерлинга
,	запятая	**?**	знак вопроса
$	доллар	**;**	точка с запятой
.	точка	**'**	одиночная кавычка
+	плюс	**/**	прямая черта
"	двойная кавычка	_	подчеркивание
=	равно	**!**	восклицательный знак
<	меньше чем	**^**	диакритический знак над гласной
>	больше чем		

3.4.2. ЗАРЕЗЕРВИРОВАННЫЕ СЛОВА

GAMS подобно другим компьютерным языкам, таким как C или Paskal, использует зарезервированные слова (или их иначе называют ключевыми служебными словами), которые имеют предопределенное значение. Вы не можете использовать их для определения идентификатора или в качестве метки. Полный список зарезервированных слов приведен ниже. Кроме слов в GAMS зарезервировано несколько составных символов из неалфавитных литер.

abort	ge	not	smin	if	lead
acronym	gt	option	sos1	then	lag
acronyms	inf	options	sos2	else	mapval
alias	integer	or	sum	semicont	putclose
all	le	ord	system	semiint	put
and	loop	parameter	table	file	undf
assign	lt	parameters	using	files	errorf
binary	maximizing	positive	variable	putpage	exp
card	minimizimg	prod	veriables	puttl	log
display	model	scalar	xor	free	log10
eps	modelz	scalars	yes	no	normal
eq	na	set	repeat	solve	add

equation equations	ne negative	sets smax	until while	for	power flexibilit y

Зарезервированные неалфавитные символы

=l=	**--**
=g=	**++**
=e=	******
=n=	

3.4.3. ИДЕНТИФИКАТОРЫ

Идентификаторы - это имена, которые мы даем элементам **sets, parameters, variables, models** и так далее. GAMS требует, чтобы идентификатор всегда начинался с буквы, затем следуют буквы или цифры. Длина идентификатора ограничена десятью литерами. Идентификатор может включать в себя только алфавитные литеры (буквы и цифры).

Пример допустимых идентификаторов:

a a15 revenue x0015

Пример недопустимых идентификаторов

15 $rhs oie moi ewoifkoioiiio ueru&wc

Имя, использованное для одного типа данных, не может быть использовано для другого.

3.4.4. МЕТКИ

Метки - элементы множества. Длина метки ограничена десятью литерами. Метки могут быть использованы в двух формах: в кавычках и без кавычек.

<Метки> GAMS во многом схожи с понятием <индекс> в других языках.

Форма без кавычек - это наиболее простая в применении форма, но она имеет ограничения при использовании литер, любая метка без кавычек должна начинаться с буквы или цифры, следующей может быть буква, цифра или одиночный знак **+** или **-**. Примеры меток без кавычек:

Phots-Acid 1986 1952-53 *Фактически метка без кавычек это*
September H2S04 Line-1 *имя множества состоящего из элементов*

В метках с кавычками кавычки используются, чтобы выделить конкретные метки. Метки могут начинаться/включать в себя любой допустимый литер. Можно использовать и двойные, и одиночные кавычки. Закрывающих кавычек должно быть столько же сколько открывающих. Если метки в кавычках заключены в двойные кавычки, то внутри метки могут содержаться одиночные кавычки и наоборот. Большинство опытных пользователей избегают применять метки в кавычках, так как работа с ними может быть утомительной при вводе и вносит сложности при чтении текста модели. Имеется два специальных условия, когда использовать метки в кавычках необходимо. Если кто-то хочет сделать выделение конкретного индекса-метки, тогда он может, например, использовать звездочки в идентификаторе метки и сделать отступ.

Более сложный пример - ключевое слова GAMS может быть использовано как метка, если оно в кавычках. Если кто-то хочет использовать метки, такие

как **no**, **ne** или **sum**, тогда они должны быть в кавычках. Пример меток в кавычках:

'*TOTAL*' 'MATCH' '10%INCR' '12"/FOOT' 'LINE 1'

Метки не имеют численного значения. Метка **'1986'** не имеет численного значения 1986, а метка **'01'** отличается от метки **'1'**.Вот в этом то и причина почему они не называются индексами.

Правила составления идентификаторов и меток показаны ниже:

	идентификатор	**метка без кавычек**	**метка в кавычках**
Количество знаков	10	10	10
Должно начаться с	буквы	буквы и цифры	любой литер
Разрешенные особые Литеры	нет	литеры + или -	любой литер кроме начальной кавычки

3.4.5. ТЕКСТ

Идентификаторы и простые метки могут быть также связаны с сопровождающим текстом. Этот текст является не более чем комментарием, он сохраняется GAMSом и выводится на экран всегда, когда записываются результаты для идентификатора.

Текст может быть в кавычках и без кавычек. Текст в кавычках может включать в себя любые литеры, кроме используемых кавычек. Текст должен быть расположен на одной линии и не может превышать 80 литер в длину. Текст, используемый в форме без кавычек, имеет некоторые ограничения. Текст без кавычек не может начинаться с зарезервированных слов, литер **'..'**, или **'='** и не должен включать в себя литеры точка с запятой **';'**, запятые **','** и наклонную черту **'/'**. Конец строки определяется текстом. Эти требования являются следствием синтаксиса GAMS и это правило у пользователя обычно не вызывает сложности в применении.

Некоторые примеры:

```
this is text
final product shipment (try)
"quoted text containing owerwise illegal characters ; / ,"
' use single qutes ti put a "double" qoute in text'
```

3.4.6. ЧИСЛА

Численные значения вводятся в стиле, подобно тому, как это делается в других компьютерных языках.

Пробелы не могут быть использованы в числах, так как GAMS воспринимает пробелы как разделение чего либо на части.

В GAMS не существует общих различий между реальными и целочисленными типами данных. Вспомните иные языки для сравнения. Если число используется без десятичной точки, оно все равно хранится как реальное число.

GAMS использует обширный диапазон арифметики, который включают специальные символы: для положительной бесконечности - (**INF**), для отрицательной бесконечности - **(-INF**), для неопределенных значений - (**UNDF**), для маленьких величин-эпсилон - (**EPSEps**) и для не доступных - (**NA**). Нельзя ввести число **UNDF**, оно получается в процессе выполнения операций, когда в результате не может получиться правильного ответа, например при делении на

ноль. Все другие специальные символы могут быть введены и использованы, как если бы они были обыкновенными числами.

Следующие примеры показывают различные допустимые способы ввода данных:

```
0        156.70      -135       .0095        1.
2e10     2e+10       15.e+10    .314e5       +1.7
0.0      .0          0.         INF          -INF
EPS      NA
```

Буква **e** означает хорошо известную ученым систему обозначений, позволяющую удобно представлять очень большие и очень маленькие числа. Например:

1e-5 = 1 x 10^{-5} = 0.00001

3.56e6 = 3.56 x 10^{6} = 3560000

GAMS использует меньший диапазон чисел, чем имеется на многих компьютерах. Это сделано специально с целью, чтобы обеспечить нормальную работу программ GAMS на широком диапазоне машин, включая персональные компьютеры. Имеется хорошее общее правило - избегать использовать и создавать в процессе решения числа с абсолютным значением больше чем **1.0e20**.

Вспомним что вводимое число может содержать до **10** цифр на всех машинах, и больше на некоторых.

3.4.7. ОГРАНИЧИТЕЛИ

Как было упомянуто в разделе 3.2.1, операторы разделяются между собой знаком точка с запятой ';'. Однако, если следующий оператор начинается с зарезервированного слова (часто называемого ключевым слово), GAMS не требует использования точки с запятой.

Знаки запятая ',' и наклонная вправо черта '/' используются как разделители в списке данных, что будет рассмотрено в разделах 4 и 5. Запятая завершает элемент данных (как будто встречен неотображаемый символ end-of-line), а наклонная черта завершает список данных.

3.4.8. КОММЕНТАРИИ

Комментарий - это сопроводительный текст, который не выполняется и не сохраняется компьютером. Существует три способа включить комментарий в программу GAMS. Первый уже рассмотрен ранее, это когда строка начинается с литера звездочки '*' на первой позиции. Остальные знаки на линии игнорируются, но распечатываются в файле вывода.

Второй способ заключается в использовании специальных "блочных" разделителей, которые указывают какой раздел должен игнорировать GAMS. На первой позиции строки должен быть знак '**$**'.

Выбор между двумя этими способами зависит от персонального вкуса и удобства.

Ниже дан пример использования блочного комментария.

```
$ontext
    Folowing a $ontext directive in column 1 sll are
    ignored by GAMS but printed on the output file until the
    matching $offtext is encountered, also in column 1.

    This facility is often used to logically remove parts of
    programs that are not used every time, such as statements

    Every $ontext must have a matching in the same file
$offtext
```

Третий стиль позволяет вставлять комментарий внутри строк. Это задается при помощи параметров компилятора - **$inlinecom** или **$eolcom**, как показано в примере:

```
    $eolcom #
    $inlinecom {}

    x = 1; #this is a coment
    y = 2; {this is also a coment}  z = 3 ;
```

3.5. ЗАКЛЮЧЕНИЕ

В данной главе заканчивается обсуждения компонентов языка GAMS. Многие незнакомые термины, используемые в этом разделе, приведены в словаре специальных терминов.

ОПРЕДЕЛЕНИЕ SET 4

4.1. ВВЕДЕНИЕ

Set в переводе с английского означает множество.

Sets - это фундаментальные составляющие блоки в любой модели GAMS. Они позволяют модели быть кратко заявленной и легко читаемой. В этом разделе мы обсудим, как **sets** декларируются и инициализируются. В GAMS имеют место несколько более расширенные понятия, связанные с **set** как, например, присвоение **sets** или команды **lag** и **lead**, но они представлены в книге дальше. Однако, темы, охваченные в этом разделе, будут достаточными, чтобы обеспечить хорошее начало в составлении большинства моделей.

4.2. ПРОСТЫЕ SETS

Множество **S**, которое содержит элементы **a**, **b** и **c** записывается в обычной математической форме следующим образом

S = {a,b,c}

В GAMS из-за ограниченного набора литеров то же самое множество **S** должно быть записано в виде

```
set      S    /a,b,c /
```

Оператор **Set** начинается с ключевого слова **set** (или **sets**). **S** - это имя множества; **a, b** и **c** являются его членами. Это метки, но их часто относят к элементам или к членам.

4.2.1. СИНТАКСИС

В общем случае, синтаксис в GAMS для простых **sets** следующий

```
set  set_name  ["text"]  [/element ["text"] {, element ["text"]} /]
   {,set_name  ["text"]  [/element ["text"] {, element ["text"]} /];
```

В GAMS **set_name** - это внутренне имя множества (также называемое идентификатором). Сопроводительный текст используется для более подробного описания множества или элемента, и располагается непосредственно за описываемым объектом.

4.2.2. ИМЕНА SET

Имя **set** является идентификатором. Идентификатор должен начинаться с буквы, затем следуют буквы или цифры. Он может содержать в себе только алфавитные литеры и может быть длиной до 10 литер. Этого достаточно, чтобы составить необходимые имена, а для обеспечения большей детальности в описании можно использовать сопроводительный текст.

Примеры допустимых идентификаторов:

i i15 countries s0051

а ниже следуют некорректные идентификаторы

25 $currency countriesinafrica food&drink

4.2.3. ЭЛЕМЕНТЫ SET

Небольшое повторение предыдущей главы здесь вынуждено присутствует. Имя каждого элемента **set** может быть длиной до 10 литер. Оно имеет две формы записи: в кавычках и без кавычек. Форма без кавычек более простая в применении, но имеет ограничения на использование литер: любая метка без кавычек должна начинаться с буквы или цифры, следующими могут быть только буквы, цифры одиночные литеры **+** или **-**.

Ниже даны примеры допустимых меток без кавычек

Phos-Asid 1986 1952-53 A
September H2S04 Line-1

В метках с кавычками кавычки используются, чтобы разделить метки. Метки могут начинаться/включать в себя любой допустимый литер. Можно использовать и двойные, и одиночные кавычки. Закрывающих кавычек должно быть столько же сколько открывающих. Если метки в кавычках заключены в двойные кавычки, то внутри метки могут содержаться одиночные и наоборот. Большинство опытных пользователей избегают применять метки в кавычках, так как они могут быть утомительными при вводе и вносят сложности при чтении. Имеется два особых условия, когда использовать метки в кавычках необходимо. Если кто-то хочет сделать выделение метки, тогда он может, например, использовать звездочки в идентификаторе метки и сделать отступ. Более сложный пример - ключевое слова GAMS может быть использовано как метка, если оно заключено в кавычки. Если кто-то хочет использовать метки, такие как **no**, **ne** или **sum**, то эти слова должны быть так же заключены в кавычки. Пример меток в кавычках:

'*total*' 'Match' '10&incr' '12"/foot' 'Line 1'

Метки не имеют численного значения. Метка **'1986'** не имеет численного значения **1986**, а метка **'01'** отличается от метки **'1'**.

Каждый элемент **set** должен быть разделен от других элементов запятой или **end-of-line** (*концом строки*), а от сопроводительного текста - пробелом.

Рассмотрим следующий пример из модели Egiptian fertiliser (*Египетский удобрительный тук*) [FERTS], в которой **set** (*множество*) fertiliser nutrients (*питательный удобрительный тук)* могло быть записанным как

```
set   cq  "nutrients" /N, P205  /;
```

или как

```
set   cq  "nutrients" / N
                        P2O5/;
```

Последовательность, в которой элементы **set** записываются обычно не важна. Однако, если элементы **set** представляют собой, например, периоды времени, тогда удобно ссылаться на "следующий" или "предыдущий" член. Для этого имеются специальные команды, которые будут рассмотрены в разделе 12. Пока достаточно помнить, что до тех пор, пока не используется команда, подразумевающая наличие последовательности, последовательность, в которой

элементы **set** (*множества*) описаны, не относится к делу. Если последовательность описания элементов **set** важна, тогда правила изменяются, а последовательность становится так называемой упорядоченной последовательностью.

4.2.4. СОПРОВОДИТЕЛЬНЫЙ ТЕКСТ

Каждый член или элемент **set** (*множества*) может сопровождаться текстом. Поясняющий текст не может превышать 80 литер в длину и должен располагаться на той же линии, где описаны идентификаторы или метки, к которым он относится.

Например, текст к меткам для **set** (*множества*) "final products" в моделе [SHALE] включает в себя сведения об единицах измерения.

```
set    sf "final products"
          /synclude   "refined crude million barrels)"
           lpg        "liguifiled petroleum gas (mln barrels)"
           ammonia    " ammonia (million tons)"
           coke       "coke (million tons)"
           sulfur     "sulfur (million tons)"
            /;
```

Обратите внимание, что в текст могут быть вставлены пробелы, и он может включать в себя особые литеры, такие как круглые скобки. Однако, на некоторые литеры имеются ограничения. В текст можно включать наклонную вправо черту '/', запятую ',' или точку с запятой ';'только если он заключен в кавычках. Определение **set**, подобное ниже приведенному

```
set prices  prices of commodities in dollars/ounce
            /gold-price, sil-price /;
```

будет причиной ошибки, так как наклонная вправо черта между **dollars** и **ounce** будет указывать на начало декларации множества, и компилятор GAMS будет трактовать **ounce** как имя первого элемента. Затем, наклонная вправо черта перед **gold-price** будет трактоваться, как конец определения **set**, а сам **gold-price** будет трактоваться как новое множество. Однако, эти проблема решается, если заключить поясняющий текст в кавычки. Следующий текст допустим:

```
set prices  "prices of commodities in dollars/ounce"
```

4.2.5 УПОРЯДОЧЕННЫЕ ЭЛЕМЕНТЫ SET (МНОЖЕСТВА)

Литер звездочка "*****" играет особую роль в определениях **set**. Он используется, чтобы облегчить набор упорядоченных элементов **set**, и уменьшить количество ошибок при наборе. Например, в абстрактной модели, где должны быть данные о десятилетнем периоде времени (с 1991 по 2000 годы) вместо того, чтобы набирать десять значений, элементы этого множества могут быть записаны как

```
    set   t   "time"  /1991 * 2000/;
```

Такая запись означает, что **set** (*множество*) **t** включает 10 элементов: 1991, 1992, ..., 2000. GAMS составляет списки меток, проверяя различия между двумя метками. Если только отличительные литеры являются цифрами, и если

число (скажем **L**), сформированное этими цифрами слева меньше, чем справа (**R**), тогда метки составляются для каждого целого числа в последовательности от **L** до **R**. Любое не цифровое различие или другая противоречивость будут восприниматься как ошибки.

Следующий пример иллюстрирует наиболее общую форму определения "со звездочкой"

```
set  g  /a1bc * a20bc/;
```

Обратите внимание, что ниже приведено не тоже самое множество

```
set  g  /a01bc * a20bc/;
```

Хотя каждое из множеств имеет 20 членов, но последнее имеет только 11 совместных членов. Как и последний пример, нижеследующие являются недопустимыми, потому что они противоречат правилам составления списка:

```
set   illegal1  /a20bc * a10bc /;
      illegal2  /a1x1  * a9x9 /;
      illegal3  /a1    *   b9 /;
```

В конце обратите внимание, что элементы **set** (*множества*), часто представленные как метки, могут содержать в себе одиночные литеры + и - так же как буквы и цифры.

4.2.6. ДЕКЛАРАЦИИ ДЛЯ МНОЖЕСТВЕННЫХ SETS (МНОЖЕСТВ)

Ключевое слово **set** (если хотите, можно говорить **sets**: это два эквивалента) не обязательно использовать для каждого множества; лучше использовать его только в начале группы множеств. Часто удобно разместить декларации группы множеств вместе в начале программы. Когда вы поступаете таким образом, то ключевое слово **set** можно использовать только однажды. Если вы предпочитаете смешивать декларацию **set** (*множеств*) с другими операторами, то вы должны использовать оператор **set** для каждой дополнительной группы **sets** (*множеств*).

Нижеприведенные примеры показывают, как два множества могут декларироваться вместе. Обратите внимание, что точка с запятой используются только после того, как продекларировано последнее множество.

```
sets
       s  "Sector "   /manuf
                       argi
                       services
                       govenment  /

       r  "Regions"   /north
                       eastcoast
                       midwest
                       sunbelt    /;
```

4.3 ОПЕРАТОР ALIAS: ДУБЛИРУЮЩИЕ ИМЕНА ДЛЯ МНОЖЕСТВА

Иногда необходимо иметь больше, чем одно имя для одного и того же множества. Например, в моделях input/output каждый предмет потребления может быть использован в производстве всех других предметов потребления, и

необходимо иметь два имени для множества предметов потребления, чтобы описать задачу без неопределенности.

В общей равновесной модели [ORANI] множество предметов потребления записывается

```
set   c   "commodities"  /food,  clothing /;
```

и второе имя для множества присваивается с помощью следующих операторов

```
alias   (c, cp) ;
alias   (cp, c) ;
```

где **cp** - новое множество, которое может использоваться вместо первоначального множества **c.**

Вновь введенное множество может быть использовано, как альтернативное имя для первоначального множества **c**, и всегда будет содержать в себе те же самые элементы, что и первоначальное множество.

Оператор **alias** может быть использован для того, чтобы ввести более чем одно новое имя для первоначального множества.

```
alias (c, cp, cpp, cppp) ;
```

где новые множества **cp, cpp** и **cpp** - есть новые имена для первоначального множества **c**.

Последовательность множеств в операторе **alias** не имеет значения. Но GAMS ставит ограничение: ранее из множеств, имеющих место в операторе **alias,** определено только одно множество. Все другие множества вводятся с помощью оператора **alias.**

Демонстрация использования множества, полученного при помощи оператора **alias,** будет приведена в разделе 6. А пока необходимо запомнить, что они используются тогда, когда множество должно быть представлено более чем одним именем.

4.4. ПОДМНОЖЕСТВА И ПРОВЕРКА ДОМЕНА

Часто необходимо определить множество, все члены которого должны быть членами какого-то большего множества.

Синтаксис следующий:

```
set  set_ident1   (set_ident2) ;
```

где **set_ident1** является подмножеством большего множества **set_ident2.**

Например, мы можем захотеть определить сектора в экономической модели [CHENERY] в следующем стиле

```
set
i      "all sectors"         /linght-ing, food+agr, heavy-ind, services /
t(i)   "traded sectors"      /linght-ing, food+agr, heavy-ind           /
nt     "non-traded sectors"  /services                                  /;
```

Некоторые виды экономической деятельности, например, экспорт и импорт, могут быть логически включены в подмножество всех секторов. Например, чтобы смоделировать торговый баланс мы должны знать, какие сектора (элементы множества) участвуют в торговле (являются traded), и имеется один

явный способ записать их определенно, что и показано в вышеприведенном определении множества **t**. Запись **t(i)** означает, что каждый член множества **t** должен также быть членом множества **i**. GAMS навяжет эту связь, которая называется проверка домена. Очевидно, что в данном случае последовательность декларации важна: членство **i** должно быть известно до того, как декларируется **t** для выполнения проверки. В следующих разделах будет дано больше информации на эту тему. Сейчас важно обратить внимание на то, что проверка домена найдет любые орфографические ошибки, которые могут быть сделаны в членах, основывающих множество **t**. Орфографические ошибки могли бы быть причинами ошибок в модели, если бы они проходили не выявленными.

Допустимо, но неэффективно, определять подмножество без ссылки к более большому множеству, как это выполнено для вышеприведенного множества **nt**. Если бы сервис был записан с ошибками, но они бы не были выявлены, и модель дала бы некорректные результаты. Поэтому мы настаиваем на использовании проверки домена там, где это только возможно. Это выявляет ошибки и позволяет писать модели, которые концептуально чище, потому что логические связи составлены точно.

Здесь завершается обсуждение множеств с простыми элементами. Этой информации достаточно для большинства приложений GAMS, однако, существует множество задач, для которых удобно иметь множества, которые определяются посредством двух или более других множеств.

4.5. МНОГОМЕРНЫЕ МНОЖЕСТВА

Часто необходимо выполнять преобразования между элементами различных множеств. Для этих целей GAMS делает возможным использование многомерных множеств.

GAMS разрешает десятимерные множества.

Следующие два подраздела рассказывают, как выражать преобразования **one-to-one** и **many-to-many** между множествами.

4.5.1. ПРЕОБРАЗОВАНИЯ ONE-TO-ONE

Рассмотрим множество, элементы которого представлены парами:

A = {(b,d), (a,c), (c,e)}

В этом множестве три элемента, и каждый элемент включает в себя пару букв. Этот вид множества удобен во многих типах моделирования. В качестве иллюстрации рассмотрим модель "мировой алюминий" [ALUM], в которой необходимо связать каждую страну, владеющую залежами боксита, с портом, который расположен близко к бокситовым рудникам. Множество стран следующее

```
set    c   "countries"
           / jamaika
             haity
             guyana
             brazil   /;
```

а множество портов следующее

```
set    p   "ports"
           / kingston
```

```
            s-domingo
            georgetown
            belem      /;
```

Затем может быть составлено множество, чтобы связать каждый порт с соответствующей страной,

```
set  ptoc(p, c)  "port to country relationship"
               / kingston    . jamaika
                 s-domingo   . haity
                 georgetown  . guyana
                 belem       . brazil    /;
```

Точка между **kingston** и **jamaika** используется, чтобы создать одну такую пару. Для удобочитаемости вокруг точки можно свободно ставить пробелы. Множество **proc** имеет четыре элемента, а каждый элемент включает в себя пару - порт-страна. Знаки (**p,c**) после имени множества **proc** указывают, что первым членом каждой пары должен быть член множества портов **p**, а вторым должен быть член множества стран **c**. Это второй пример проверки домена. GAMS проверит элементы множества, чтобы гарантировать, что все члены принадлежат соответствующим множествам.

4.5.2. ПРЕОБРАЗОВАНИЕ MANY-TO-MANY

Преобразование **many-to-many** необходимо в определенных случаях. Рассмотрим следующее множество

```
set   i           /  a,   b            /
      j           /  c,   d,    e      /
      ij1(i,j)    /  a.c, a.d          /
      ij2(i,j)    /  a.c, b.c          /
      ij3(i,j)    /  a.c, b.c, a.d, b.d / ;
```

ij1 представляет преобразование **one-to-many**, где один элемент **i** соотносится со многими элементами **j**. **ij2** представляет преобразование **many-to-one**, где много элементов **i** соотносятся **c** одним элементом **j**. **ij3** является самым общим случаем, где много элементов **i** соотносятся со многими элементами **j**. Это множество может быть записано компактно в виде

```
set   i          /  a, b             /
      j          /  c, d, e          /
      ij1(i,j)   /  a.(c,d)          /
      ij2(i,j)   / (a,b).c           /
      ij3(i,j)   / (a,b).(c,d)       / ;
```

Скобки гарантируют запись элементов, и они могут быть раскрыты при составлении пар.

Когда составляются комплексные множества, подобные выше приведенному, важно проверить, что описанное множество существует. Проверка может быть выполнена, используя оператор **display**.

Вышесказанное может быть обобщено на множество с более чем двумя метками в каждом элементе. Математически оно называется 3-кортежное, 4-кортежное или, более обобщенно, n-кортежное.

Этот раздел заканчивается несколькими примерами, которые иллюстрируют определения многомерных элементов множества.

Примеры компактного представления n-кортежных множеств, использующих комбинации точек, скобок и запятых даны в таблице

Конструкция	Результат
(a,b) . c.d	a.b.c , a.b.d
(a,b) . (c,d) . e	a.c.e , b.c.e , a.d.e , b.d.e
(a . 1*3) . c	(a.1 , a.2 , a.3) . c
	или a.1.c , a.2.c. , a.3.c
1*3 . 1*3 . 1*3	1.1.1 , 1.1.2 , 1.1.3 , ... , 3.3.3

Обратите внимание, что звездочка также может быть использована вместе с точкой. При рассмотрении вышеприведенного примера помните, что элементы списка 1*4 являются {1,2,3,4}.

4.6. ЗАКЛЮЧЕНИЕ

В GAMS простые множества включают в себя имя множества и элементы множества. И имя, и элементы могут иметь сопроводительные тексты, поясняющие более детально имя или элементы. Более комплексные множества имеют элементы, которые являются двух и даже n-кортежные. Эти множества с двух и n-кортежами являются идеальными для установления связей между элементами в различных множествах. GAMS также использует возможности проверки домена, чтобы помочь выявить противоречия в метках и орфографические ошибки, сделанные в течении определения связанных множеств.

В этом разделе обсуждение ограничивается тем, что множества, чьи элементы определяются как множество, декларируются. Для многих моделей это все, что необходимо знать о множествах. Далее мы будем обсуждать более сложные понятия, такие как множества, чьи члены изменяются в различных частях модели (присвоение множествам) и другой набор команд, таких как объединение, дополнение и пересечение множеств.

ВВОД ДАННЫХ:

PARAMETERS, SCALARS И TABLES 5

5.1. ВВЕДЕНИЕ

Данная глава посвящена обзору методов декларации и определения числовой информации присутствующей в оптимизационных моделях ввиде констант

Одна из ключевых идей языка GAMS состоит в том, чтобы использовать данные в их основных формах: скалярной, в форме упорядоченной последовательности данных или в форме двух и более мерных таблиц. Основываясь на этих требованиях, в этом разделе представлены три типа декларации и определения данных.

Scalar	(*скаляр*)	Простой (скалярный) ввод данных
Parameter	(*параметр*)	Упорядоченная последовательность данных
Table	(*таблица*)	Данные, представленные в форме таблиц. Должны быть двух или более мерными.

Каждый из этих трех типов данных будет детально рассмотрен в следующих разделах.

Инициализация данных для параметров может быть выполнена только раз; поэтому данные должны модифицироваться при помощи операторов присвоения, что является предметом рассмотрения следующего раздела.

5.2. SCALARS (СКАЛЯРЫ)

Оператор **scalar** используется, чтобы декларировать и (необязательно) инициализировать параметр GAMS, имеющий нулевую размерность. Это возможно в случае, когда нет ассоциативных множеств, и поэтому имеется только один элемент с параметром число.

5.2.1. СИНТАКСИС

В общем случае синтаксис в GAMS для декларации **scalar** следующий

```
scalar(s)    scalar_name   [text]  [/signed_num/]
           {scalar_name   [text]  [/signed_num/] } ;
```

Scalar_name в GAMS - внутренне имя скаляра (также называемое идентификатором). Сопутствующий текст используется, чтобы описать множество или элемент и располагается сразу после описываемого объекта. **Signed_num** - число со знаком, которое присваивается **scalar_name**. Как и все идентификаторы, **scalar_name** должен начинаться с буквы, затем могут следовать буквы или цифры. **Scalar_name** может содержать только алфавитные литеры, он может быть до 10 литер длиной. Сопутствующий текст не может превышать 80 литер и должен располагаться на той же линии, что и описываемый идентификатор или метка.

5.2.2. ИЛЛЮСТРАТИВНЫЙ ПРИМЕР

Пример скалярного определения в GAMS приведен ниже

```
Scalars    rho    "discount rate"                          /.15/
           irr    "internal rate of return"
           life   "financial lifetime of productive units" /20/;
```

Вышеприведенный оператор инициализирует **rho** и **life**, но не инициализирует **irr**. Позже может быть использован другой оператор **scalar** для инициализации **irr** или, чтобы дать значение **irr**, может быть использован (это понятие будет рассмотрено в следующем разделе) оператор присвоения

```
irr = 0.07 ;
```

5.3. PARAMETERS (ПАРАМЕТРЫ)

Несмотря на то, что **parameter** (*параметр*) - это тип данных, который включает в себя понятия **scalars** и **tables**, обсуждение в этом разделе будет сфокусировано на вводе данных посредством именно этого оператора. Упорядоченная последовательность данных может быть считана в GAMS, используя оператор **parameter**.

5.3.1. СИНТАКСИС

В общем случае синтаксис в GAMS для декларации **parameter** следующий

```
parameter(s) param_name [text] [/element [=] sigmed_num
                              { [/element [=] sigmed_num }/]
        {   param_name [text] [/element [=] sigmed_num
                             {[/element [=] sigmed_num }/] } ;
```

в GAMS **param_name** - внутреннее имя **parameter** (*параметра*) (также называемого идентификатором). Сопутствующий текст используется, чтобы описать параметр, и расположен он сразу после описываемого объекта. **Signed_sum** - является числом со знаком и декларируется, чтобы ввести значение, ассоциированное с соответствующим элементом. Как и все идентификаторы, **param_name** должен начинаться с буквы, затем следуют буквы и цифры. **Param_name** может содержать только алфавитные литеры, он может быть до 10 литер длиной. Сопутствующий текст не может превышать 80 литер и должен располагаться на той же линии, что и описываемый идентификатор или метка.

Parameter (*параметер*) может быть проиндексирован через одно или более **sets** (*множеств*) (максимум через 10). Элементы в данных должны принадлежать к **set** (*множеству*), через который индексируется **parameter** (*параметр*).

Значение параметра по умолчанию равно нулю.

Инициализация **parameter** (*параметра*) требует перечень элементов данных, каждый элемент в перечне включает индекс-метку и значение. Наклонная вправо черта должна быть в начале и в конце перечня, запятые должны отделять элементы данных, если они введены более одного на линии. Знак равенства или пробелы могут быть использованы, чтобы разделить индекс-

метку-кортеж от ассоциированного значения. Синтаксис определения **parameter** (*параметр)* может быть таким же, какой использовался для определения **set** (*множества*).

5.3.2. ИЛЛЮСТРАТИВНЫЙ ПРИМЕР

Нижеприведенный фрагмент заимствован из [MEXSS]. Мы также приводим определение **set** (*множества*), так как это сделает этот пример более понятным

```
set   i   "steel plants" /  hylsa      "monterrey"
                            hylsap     "puebla"       /
      j   "markets"      /  mexico-df, monterrey, guadalaja / ;

     parameters dd(j)    distrubution of demand
                          /mexico-df     55
                           guadalaja     15  /;
```

Спецификация [1] проверки домена для **dd** означает, что будет существовать вектор данных ассоциированных с **dd** - по одному числу, соответствующему каждому занесенному в список члену множества **j**. Числа специфицируются вместе с декларацией в формате, очень напоминающем способ, которым мы специфицировали множества: в этом простом случае метки следовали за пробелом разделителем и затем следовало значение. Форматы ввода любого допустимого числа являются допустимыми для значения. По умолчанию значения данных равны нулю. Так как **monterrey** не был включен в список данных, то значение, ассоциированное с **dd("monterrey")** - market share in monterrey (*доля торговли в Монтерее*) - будет равна нулю.

На одной линии можно расположить несколько элементов данных, разделенных между собой запятыми

```
parameter a(i) /seattle = 350,  san-diego = 600/
          b(i) /seattle   2000, san-diego   4500 /;
```

Как и для **sets** (*множеств*), в конце строки запятая необязательна.

5.3.3. ДАННЫЕ ПАРАМЕТРА БОЛЕЕ ВЫСОКИХ РАЗМЕРНОСТЕЙ

Размерность **parameter** (*параметра*) может достигать 10. Инициализация упорядоченного списка данных через оператор **parameter** может быть легко распространена на данные с более высокой размерностью. Индексы-метки, которые появляются на каждой линии в одномерном случае, заменяются меткой-кортежем для более высоких размерностей. Элементы в n-кортеже разделяются точкой, как и в случае многомерных множеств.

Следующий пример иллюстрирует использование данных параметра более высоких размерностей:

```
parameter  salaries (employee, manager, department)
           /anderson   .murphy   .toy         =  6000
            hendry     .smith    .toy         =  9000
            hoffman    .morgan   .cosmetics   =  8000 / ;
```

Все правила использования в записях звездочек и круглых скобок, которые мы рассмотрели при обсуждении **sets** (*множеств*), здесь также действительны. Ниже дан пример, в котором инициализируется очень малая часть данных из всей совокупности данных. GAMS зарегистрирует ошибку, если какие-то комбинации меток (или меток-кортежей) появятся в перечне данных более одного раза.

```
set    row    /row1*row10/
       cod    /cod1*cod10/ ;
```

---

[1] Спецификации задают условия и эффект действия программы, не указывая способа достижения необходимого эффекта.

```
 parameter  a(row, cod)
           /(row1, row4) . cod2*cod7      12
            row10 . cod10                 17
            row1*row7 . cod10             33  /;
```

В данном примере 20 переменных:

- от **row1.cod2** до **row1.cod7** и от **row4.cod2** до **row4.cod7** (12 переменных) приниаются равными 12;
- **row10.cod10** (одна переменная) принимается равным 17;
- от **row1.cod10** до **row7.cod10** (семь переменных) принимаются равными 33.

Необъявленные переменные (80 переменных) принимаются равными нулю.

Этот пример показывает способность GAMS обеспечить для разбросанной структуры данных сжатую инициализацию или определение.

5.4 TABLES (ТАБЛИЦЫ)

В GAMS табулированные данные можно декларировать и инициализировать, используя оператор **table**. Для двух- и более мерных параметров это гарантирует более сжатый и легкий метод ввода данных, чем подход, базирующийся на списке, так как каждая метка появляется только однажды (по крайней мере в небольших таблицах).

5.4.1. СИНТАКСИС

В общем случае в GAMS синтаксис для декларации **table** следующий

```
table  table_name      [text]     EOL

                element              {element}
  element       signed_num        {signed_num}  EOL
{ element       signed_num        {signed_num}  EOL }  ;
```

В GAMS **table_name** - это внутренне имя **table** (*таблицы*) (также называемое идентификатором). Сопутствующий текст используется, чтобы описать множество или элемент и располагается сразу после описываемого объекта. **Signed_sum** - является числом со знаком и декларируется, чтобы ввести значение, ассоциированное с соответствующим **element** (*элементом*).

Оператор **table** является единственным оператором в языке GAMS, который не имеет свободного формата.

Правила составления переменных **TABLE**:

- Взаимное расположение всех объектов в таблице важно. Это единственный оператор, где имеет значение конец строки (EOL). Расположение литер численных объектов в таблице должно перекрываться положением литер заголовков колонок.
- Раздел колонок должен размещаться на одной линии.
- Последовательность чисел со знаком, формирующих строку, должна быть на той же линии.

- Определение элемента строки может размещаться более чем на одной линии.
- Во всей таблице спецификация колонок может появляться только раз.

Правила для формирования простых таблиц не сложны. Компоненты линии заголовка к настоящему моменту близки к последовательности "ключевое слово - идентификатор – перечень домена - текст", перечень домена и текст необязательны. Метки используются сверху и слева таблицы, чтобы сформировать прямоугольную сетку, которая содержит в себе значения данных. Последовательность меток не важна, за исключением, если выполнена спецификация поверки домена, каждая метка которого должна соответствовать метке в ассоциированном множестве. Метки не должны повторяться, но могут быть пропущены, если соответствующие значения равны нулю или не нужны. Чтобы разделить объекты, между метками и данными должно быть по крайней мере по одному пробелу. Пустое пространство вместо объекта означает - с этой комбинацией метки будет ассоциированно значение по умолчанию (ноль).

Обратите также внимание, что в отличие от операторов **set**, **scalar** и **parameter** в операторе **table** может быть декларирован и инициализирован только один идентификатор.

5.4.2. ИЛЛЮСТРАТИВНЫЙ ПРИМЕР

Нижеприведенный пример, заимствованный из [KORPET], предворяется соответствующим определением **set** (*множества*).

```
  set   i  "plants"

           /inchon, ulsan, yosu /;

        m   " productive  units"
           / atmost-dist    "atmospheric distilation unit"
             steam-cr       "steam cracker"
             aromatics      "aromatic unit"
             hydrodeal      "hydrodealkylator" / ;

table   ka(i, m)    "initial cap. of productive unity (100 tons per year) "

                  inchon     ulsan     yosu

atmost-dist       3702       12910     9875
steam-cr                      517      1207
aromatics                     181       148
hydrodeal                     180                ;
```

В вышеприведенном примере обозначения строк взято из **set** (*множества*) **m**, а обозначение колонок из **set** (*множества*) **i**. Обратите внимание, что данные для каждой строки были расположены под соответствующим заголовком колонки.

Если будет неясно, какие цифры относятся к какой колонке, GAMS выдаст сообщение об ошибке и укажет на неопределенный объект.

5.4.3 УДЛИНЕННЫЕ TABLES (ТАБЛИЦЫ)

Если таблица имеет слишком много колонок, чтобы разместить их удобно на одной линии, то колонки, которые не поместились, могут быть продолжены на другой линии. Мы используем пример из 5.4.2, чтобы проиллюстрировать способ удлинения таблицы

```
table ka(i, m) "initial cap. of productive unity (100 tons per year) "

                inchon     ulsan

atmost-dist     3702       12910
steam-cr                    517
aromatics                   181
hydrodeal                   180

    +             yosu

atmost-dist       9875
steam-cr          1207
aromatics          148        ;
```

В удлинении таблицы решающим элементом является знак "**+**" расположенный выше меток строки и слева от меток колонки в продолженной части таблицы. Метки строки продублированы (кроме метки **hydrodeal**, которая была удалена, так как она не имеет ассоциированных данных). Таблицы могут быть продолжены столько раз, сколько необходимо.

5.4.4. TABLES (ТАБЛИЦЫ) БОЛЕЕ ЧЕМ ДВУХМЕРНЫЕ

Размерность таблицы может достигать десяти. Чтобы разделить соседние метки, опять используются точки; они могут быть и в строке, и в колонке. Метка слева строки соответствует первому множеству в перечне домена, а метка справа от каждого заглавия колонки - последнему. Очевидно, что здесь должно быть такое же число меток, ассоциированных с каждым числом в таблице, как **sets** (*множества*) существуют в перечне домена.

От выбранной разбивки будет зависеть размер контролирующих множеств и количество данных, но идеальный выбор должен быть один, который обеспечивает самый интуитивно удовлетворительный способ организации и проверки данных. Большинство людей могут более легко просматривать колонки вниз, чем вдоль строк, но если поместить дополнительные метки в строки, то это значительно уплотнит информацию.

Следующие примеры, заимствованные из [MARKO], иллюстрируют использование более чем двух мерных **tables** (*таблиц*).

```
sets    ci   "  commodities : intermediate  "
             / naphtha    "naphtha"
               dist       "distillate"
               gas-oil    "gas-oil"       /

        cr   "  commodities : crude oils "
              / mid-c     "mid-constinent"
                w-tex     "west-texas"     /

        q    " attributes of intermediate products "
               /density
                sulftur                    / ;
```

```
 table   attrib (ci, cr, q)   blending attributes
                     densty      sulftur

naphtha . mid-c        272         .283
naphtha . w-tex        272         1.48
dist . mid-c           292         .526
dist . w-tex           297         2.83
gas-oil . mid-c        295         .98
gas-oil . w-tex        303         5.05   ;
```

Table (*таблица)* **attrib** также может быть составлена, как показано ниже

```
 table   attrib (ci, cr, q)   blending attributes

         mid-c.densty  w-tex.densty   mid-c.sulftur  w-tex.sulftur

naphtha     272          272              1.48           .283
dist        297          297              2.83           .526
gas-oil     303          303              5.05           .98      ;
```

5.4.5 СОКРАЩЕНИЕ TABLES (ТАБЛИЦ)

Все правила использования в записях звездочек и круглых скобок, которые мы привели при обсуждении множеств, здесь также действительны. Следующий пример показывает, каким образом могут быть сокращены повторяющиеся колонки или строки, если использовать звездочки и записи в круглых скобках. В примере членство **set** (*множества*) не показано, но может быть легко выведено.

```
    table  upgrade (srat, size, tech)
             small.tech1  small.tech2  medium.tech1  medium.tech2
strategy-1     .05            .05          .05           .05
strategy-2     .2             .2           .2            .2
strategy-3     .2             .2           .2            .2
strategy-4                                  .2            .2


table upgradex (srat, size, tech) alternative way of writing table
                                        альтернативный способ записи table
                                                  tech1*tech2
strategy-1                                            .05
strategy-2*strategy-3.(small,medium)                  .2
strategy-4                                            .2        ;

display  attrib,  attribx ;
```

Здесь мы снова сталкиваемся с оператором **display**. Этот оператор используется для того, чтобы записать в файл output (выводной файл) данные, ассоциированные с **upgrade** и **apgradex**.

5.4.6. УДЛИНЕННЫЕ СТРОКИ МЕТОК

В таблицах GAMS можно продолжить строки меток на вторую и даже на третью линию, если мы хотим уменьшить количество колонок. Разрыв должен быть после точки, а оставшаяся часть линии, на которой расположена неполная строка метки-кортежа, должна быть пустой.

Следующий пример, заимствованный из [INDUS], используется для иллюстрации. В действительности эта таблица содержит девять колонок и огромное число строк: здесь воспроизведен только маленький фрагмент таблицы, чтобы показать продолженные строки меток-кортежей.

```
table  yield  (c, t, s, w, z)  crop yield  (metric tons per acre)
                                                    nwfp      pmw
wheat. (bullock, semi-mech). la-plant.
                             (heavy, january)      .385      .338
wheat. (bullock, semi-mech). la-plant.light        .506      .446
wheat. (bullock, semi-mech). la-plant.standart     .592      .524
wheat. (bullock, semi-mech). (qk-harv,standart).
                             (heavy, january)      .439      .387
```

5.5. ЗАКЛЮЧЕНИЕ

В этом разделе были обсуждены декларация и инициализация параметров с помощью операторов **parameter, scalar, table**. В следующем разделе будет рассмотрено, как эти данные могут быть изменены с помощью оператора присвоения.

ОПЕРАЦИИ С ДАННЫМИ ПАРАМЕТРОВ 6

6.1. ВВЕДЕНИЕ

В модели часто требуется, чтобы однажды проинициализированные данные были преобразованы. Первая часть этого раздела посвящена непосредственно операциям с параметрами. Следующая часть раздела посвящена функциям индексного присвоения и индексным операциям.

6.2. ОПЕРАТОР ПРИСВОЕНИЯ

В GAMS оператор присвоения - это фундаментальный оператор управления данными. Он используется при определении или изменении значений, ассоциированных с **sets** (*множествами*), **parameters** (*параметрами*), **variables** (*переменными*) или **equations** (*уравнениями*).

Простое присвоение записывается в стиле, который применяется во многих компьютерных языках. GAMS использует традиционные символы для обозначения сложения (**+**), вычитания (**-**), умножения (*****) и деления (**/**). Как они работают будет проиллюстрировано на примерах в подразделах 6.2.1 - 6.2.6, а более детально описано в подразделе 6.3.

6.2.1 SCALAR (СКАЛЯРНОЕ) ПРИСВОЕНИЕ

Рассмотрим следующую абстрактную последовательность выражений:

```
scalar   x /1.5/;
x = 1.2;
x = x + 2;
```

Scalar (*скаляр*) **x** инициализируется и **x** принимает значение 1.5. Второй оператор изменяет это значение на 1.2, а третий оператор изменяет значение **x** на 3.2. Второй и третий операторы присвоения изменяют первоначальное значение **x** на новое.

Присвоение не может начинаться с зарезервированного слова. Поэтому в качестве делителя перед всеми присвоениями нужно ставить точку с запятой.

6.2.2. ИНДЕКСНОЕ ПРИСВОЕНИЕ

В GAMS создан очень мощный синтаксис для выполнения присвоений под контролем индексов **set**. Эта группа операций обеспечивает одновременное или параллельное присвоение по всему полю индексов для многомерных параметров и

создает выразительный способ определения цифровых значений большого количества декларированных в моделях массивов данных.

Рассмотрим математическую запись:

$$DJ_d = 2.75\ DA_d\ ,\ \forall\ d$$

Она означает, что для каждого индекса из множества **d**, происходит присваивание значение **2.75*DA** к **DJ**. В GAMS это будет записано следующим образом.

```
dj (d) = 2.75* da(d);
```

Данное присвоение есть индексное присвоение, и множество индексов **d** будет обеспечивать направленное присвоение через контролируемое множество **da(d)** к множеству **dj(d)**.

Индексные множества на левой стороне присвоения вместе называются контролирующим доменом присвоения.

Расширение к двум и более контролирующим индексам очевидно. Здесь будет выполнено присвоение для каждой комбинации меток, которая может быть составлена при использовании индексов внутри круглых скобок.

Рассмотрим пример присвоения ко всем элементам 100 данных **a**,

```
set   row        / r-1*r-10 /
      col        / c-1*c-10 /
      sro(row)   / r-7*r-10 /;
parameters   a(row,col),
a(row,col) = 13.2 + r(row)*c(col);
```

Расчет в последнем операторе даст значения для каждой из 100 двухзначных комбинаций меток, которые можно составить из элементов **row** и **col**. Первая комбинация выглядит следующим образом:

```
a('r-1','c-1') = 13.2 + r('r-1')*)c('c-1').
```

6.2.3. ЯВНОЕ ИСПОЛЬЗОВАНИЕ МЕТОК В ПРИСВОЕНИЯХ

В присвоениях часто необходимо использовать метки явно. Это можно выполнить, поставив вокруг меток кавычки (этот способ ранее уже демонстрировался при обсуждении **parameter**).

Рассмотрим следующее присвоение

```
a('r-7','c-4') = -2.36;
```

Этот оператор присваивает постоянное значение одному элемент **a**. Все другие элементы остаются неизменными. Вокруг меток могут быть использованы как одиночные, так и двойные кавычки.

6.2.4 ПРИСВОЕНИЯ ЧЕРЕЗ SUBSET (ПОДМНОЖЕСТВА)

В общем случае, где бы в индексных присвоениях ни встретилось имя set (*множества*), если необходимо сделать присвоение через подмножество вместо целого домена, то может быть использовано подмножество (или даже индекс-метка).

Рассмотрим пример

```
a(sro, 'col-10') - 2.44 - 33 * r(sro);
```

где **sro** было уже утверждено, чтобы быть строгим подмножество **row.**

6.2.5. ИСХОДЫ КОНТРОЛИРУЮЩИХ ИНДЕКСОВ

Число контролирующих индексов с левой стороны знака **"="** должно быть больше или равно числу индексов с правой стороны. С правой стороны не должно быть индексов, которые не указаны в левой стороне, если они не исчезнут вследствие индексных операторов **SUM() , PROD().**

Рассмотрим следующий пример

```
a(row, 'col-2') = 22 -  c(col);
```

Для GAMS такая запись ошибочна, т.к. **col** существует с правой стороны знака **"="**, но его нет на левой стороне.

При определении числа контролирующих индексов каждое **set** (*множество*) считается только раз. Если в контролирующем домене какое-то множество встречается более одного раза, то, чтобы выполнялось условие равенства числа контролирующих индексов числу индексов, второй и более высокие случаи должны быть названы иначе (при помощи оператора **alias**), чем первоначальное множество.

Рассмотрим пример

```
b(row,row) = 7.7 - r(row);        Считается что row 10 штук)
```

Этот оператор имеет только один контролирующий индекс **(row**). Согласно такой записи присвоения будут выполнены только для диагональных объектов **b**, и поэтому будет присвоено только десять значений. Недиагональные элементы зарегистрированы не будут.

Если предусмотреть дополнительное имя и использовать его во второй индексной позиции, тогда мы будем иметь два контролирующих индекса, и GAMS выполнит присвоение для всего декартового произведения, все 100 значений.

```
alias (row,rowp);
b(row,rowp) = 7.7 - r(row) +r(rowp);
```

6.2.6. ИДЕНТИФИКАТОРЫ РАСШИРЕННОГО ДИАПАЗОНА В ПРИСВОЕНИИ

В GAMS идентификаторы "расширенного диапазона" могут быть также используемы в операторе присвоения, как в случае

```
a(row, 'col-10') = inf;          a(row, 'col-10') = -inf;
```

Арифметические действия "расширенного диапазона" будут обсуждены в разделе 6.4.3. Наиболее часто используемые значения - это **NA** в неполной таблице и **INF** для границ переменных.

6.3. ВЫРАЖЕНИЯ

Выражение - это произвольная сложная спецификация для расчета с использованием, где это необходимо для прозрачности записи, круглых скобок. В этом разделе будет продолжено обсуждение присвоения параметров и приведены более детальные выражения, которые могут быть использованы справа от знака равно "=". В подразделах будут показаны все имеющиеся численные возможности стандартных и "расширенных" арифметических действий.

6.3.1. СТАНДАРТНЫЕ АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ

Стандартными арифметическими символами и операциями являются

Операторы	**Описание**
**	экспонента
*, /	умножение, деление
+, -	сложение, вычитание

Операторы записаны в последовательности по старшенству, которая определяет последовательность вычислений, когда в выражениях не используются круглые скобки.

Рассмотрим пример без круглых скобок

```
x = 5 + 4*3**2;
```

Для более наглядной записи необходимо использовать круглые скобки

```
x = 5 + (4 * (3**2));
```

Результат будет одинаков.

Использовать круглые скобки предпочтительнее, чем полагаться на старшинство операторов, так как использование скобок предотвращает ошибки и делает более ясным замысел выражения.

Ограничений на длину выражения не имеется. Оно может занимать любое количество строк: конец строки разрешается выполнять в любом месте, где может быть использован пробел. Пробел может быть использован вокруг идентификаторов, круглых скобок и операторных символов. Пробел недопустим внутри идентификаторов и чисел и является существенным внутри меток в кавычках, используемых, чтобы разделить метки.

xn** рассчитывается в GAMS, как **exp[n*ln(x)].** Достаточно вспомнить аналогию с PASKAL. Если **x** имеет отрицательное или нулевое значение, то действие вычисления **LN()** не определяется, и результатом попытки расчета будет ошибка. Если имеется вероятность появления отрицательного значения для **x** и известно, что **n** целое число, тогда можно пользоваться специальной функцией **power(x,n)** встроенной в компилятор GAMS.

GAMS обладает тремя дополнительными характеристиками, которые позволяют повысить описательные возможности и гибкость расчетных выражений: индексные операции, функции и расширенный диапазон арифметических действий.

6.3.2. ИНДЕКСНЫЕ ОПЕРАЦИИ

В дополнение к простым операциям, описанным в разделе 6.3.1, GAMS предусматривает 4 индексные операции

Операции	Описание
sum	Суммирование через контролирующие индексы
prod	Произведение через контролирующие индексы
smin	Минимальное значение через контролирующие индексы
smax	Максимальное значение через контролирующие индексы

Эти четыре операции выполняются через один или более контролирующих индексов. Синтаксис GAMS для этих операций следующий

```
indexed_op  (  (controlling_indices), expression)
```

Если имеется только один контролирующий индекс, то скобки вокруг контролирующего индекса можно убрать.

Наиболее общей из индексных операций является **sum**, которая используется, чтобы рассчитать сумму через домен set (множества).

Рассмотрим следующий пример, заимствованный для иллюстрации из [ANDEAN]

```
sets   i    plants   / cartagena, callao,      moron    /
       m    product  / nitr-acid, sulf-acid,   amm-sulf / ;

parameters  capacity(i,m)  " capacity in tons per day  "
            totcap(m)      " total capacity by process "

totcap(m) = sum(i, capacity(i,m));
```

Используя обычное математическое представление, это может быть записано следующим образом

$$TOTS_m = \sum_i C_{im}$$

Индекс **i**, через который выполняется суммирование, отделяется от зарезервированного слова **sum** левой круглой скобкой, а от элемента данных **capacity(i,m)** запятой. Для этой операции **i** снова называется контролирующим индексом. Зоной контроля является пара круглых скобок, которые ставятся сразу после ключевого слова **sum**. Нежелательно иметь две независимые индексные операции, контролируемые одним и тем же индексом.

```
R =    sum(i,(prod(i,a(i,i))))
```

Также возможно суммировать одновременно через домены двух и более set (*множеств*); в этом случае необходимо больше круглых скобок. В данных операциях вместо идентификаторов могут быть использованы арифметические выражения.

```
count = sum((i,j), a(i,j));
emp = sum(t, l(t)*m(t));
```

В традиционном математическом выражении это выглядит следующим образом

$$COUNT = \sum_i \sum_j A_{ij} \quad и \quad EMP = \sum_t L_m M_t$$

Операции **smin** и **smax** используется, чтобы найти наибольшее и наименьшее значения через домены индексных множеств или множеств. Индексы для операторов **smin** и **smax** определяются также как и в индексах для оператора **sum**. Рассмотрим следующий пример, где находится наибольший **capacity**.

```
lrgunit = smax((i,m), capacity(i,m));
```

6.3.3. ФУНКЦИИ

GAMS предусматривает использование стандартных функций, таких как возведение в степень, логарифмирование и тригонометрические функции.

Полный перечень функций, которые используются в GAMS, приведен ниже. Применять функции в уравнениях необходимо с особой тщательностью. Эта тема будет подробно раскрыта в разделе 8.

Функции	**Описание**
Errorf(x)	интеграл нормального стандартного распределения от - бесконечности до X
exp(x)	экспонента, e^x
lon(x)	натуральный логарифм
log10(x)	десятичный логарифм
normal(x,y)	случайное число нормально распределенное со средним X и стандартным отклонением Y

Специальная функция **mapval()** введена для того, чтобы присвоить числовое значение ряду понятий. Числовые значения функции **mapval** и соответствующие им понятия приведены в таблице.

Специальные символы	**mapval**	**Описание**
Inf	6	Плюс бесконечность. Очень большое положительное число.
-inf	7	Минус бесконечность. Очень большое отрицательное число.
na	5	Не действительное. Используется для отсутствующих данных. Любая операция, когда используется значение NA, в результате даст значение NA.
undf	4	Неопределенная. Результат неопределенной или недопустимой операции Пользователь не может непосредственно установить значение UNDF.
eps	8	Величина очень близкая к нулю, но отличная от нуля.

GAMS определил результаты всех арифметических операций и все значения функций, используя эти специальные значения. С результатами действия функции **mapval** можно ознакомиться, прогнав модель [CRAZY] из библиотеки. Как и ожидалось, **1+INF** становится равным **INF**, а **1-EPS** становится равным **1**.

Функция **mapval** должна использоваться в сравнениях, включая в себя расширенный диапазон арифметических действий. Только расширенный диапазон арифметических действий, показанный в вышеприведенной таблице, дает ненулевые значения для **mapval**. Например, **mapval(a)** принимает значение **6** если, **a** - это **INF**. Все регулярные числа в **mapval** дают результат **0**.

В следующей таблице приведены результаты возведения в степень и деления различных вводных параметров

Значение		**Операции**		
a	b	a**b	Возведение в степень	a/b
2	2	4	4	1
-2	2	undf	4	-1
2	2.1	4.28	undf	.952
na	2.5	na	na	na
3	0	1	1	undf
ifn	2	inf	inf	inf
2	inf	undf	undf	0

Избегайте использовать или получать в результате расчетов величины по абсолютному значению больше **1.0E20**. Если число слишком большое, тогда GAMS может рассматривать его как **(UNDF)** и все значения, полученные в результате расчетов модели, где используется эта величина, могут стать непригодными к использованию. Всегда используйте **inf** (или **-inf**) для произвольно больших значений.

Если выполняемая арифметическая операция является недопустимой или имеет неопределенный результат из-за значений аргументов (например, деление на ноль), то выдается сообщение об ошибке, а результат устанавливается неопределенным (**UNDF**). Исходя из этого, **UNDF** трактуется как правильное значение данных, и не запускает дополнительного сообщения об ошибке.

Если обнаружена ошибка, GAMS не будет решать модель, но закончит ее компиляцию с указанием всех обнаруженных ошибок (первичных и порожденных).

ПЕРЕМЕННЫЕ (VARIABLES) 7

7.1. ВВЕДЕНИЕ

Этот раздел охватывает вопросы декларации **variables** (*переменных*) GAMS и операции с ними. Многие понятия, освещенные в разделах 4 и 6, применяются здесь непосредственно.

В GAMS **variables** (*переменные*) - это название того, что экономисты называют "эндогенными переменными", эксперты линейного программирования - "параметрами определяющими критериальную функцию", а профессионалы, работающие в прикладных исследованиях, - "расчетными переменными". Это объекты, чьи значения, как правило, являются неизвестными до тех пор, пока модель не решена. Принципиальное различие между **variables** (*переменными*) GAMS и колонками в традиционной терминологии математического программирования заключается в том, что **variable** (*переменная*) GAMS может ассоциироваться со многими колонками в традиционной формулировке.

7.2. ДЕКЛАРАЦИИ VARIABLE (ПЕРЕМЕННОЙ)

Variable (*переменная*) GAMS, также как и другие идентификаторы, должна быть продекларирована прежде, чем на нее будет выполнена ссылка.

7.2.1. СИНТАКСИС

Декларация переменных выполняется подобно декларации **set** (*множеству*) или **parameter** (*параметру*), в ней разрешается наличие перечня домена и рекомендуется поясняющий текст.
В одном операторе могут быть продекларированы несколько переменных.

[var-type] variable[s] var_name [text] {, var ident}

Var-type - тип переменной (тип переменной указывается не всегда. Детально о стилях декларации переменных будет изложено в разделе 7.2.3.)
Var_name - внутреннее имя переменной (также называемое идентификатором). Идентификатор должен начинаться с буквы, затем следуют буквы или цифры. Он может включать в себя только алфавитные литеры и может быть длиной до 10 литер. Поясняющий текст используется, чтобы описать множества или элемент и располагается сразу после описываемого объекта. Поясняющий текст не может превышать 80 литер в длину и должен располагаться на той же линии, что и идентификатор.

Одно важное различие между декларациями **variable** (*переменной*) и **parameter** (*параметр*) заключается в том, что при декларации переменных не могут инициализироваться значения.

Типичный оператор **variable**, взятый из [RAMSEY], показан ниже для иллюстрации

```
variables   k(t)      capital stok (trillion rupees)
            c(t)      consumption  (trillion rupees per year)
            i(t)      investment   (trillion rupees per year)
            utility   utility measure ;
```

Декларация вышеприведенного массива **k** означает, что ссылки к **k** существуют для всего множества **t**. В ассоциированных задачах математического программирования модель включает группу переменных **k** через объединяющий индекс **t**: то есть приходиться одна переменная **k** для каждого индекса-метки **t** из соответствующего **set** (*множества*). В этом случае можно составлять очень большие модели, используя небольшое число **variables** (*переменных*), объединенных некоторыми **set** (множествами). Не смотря на то, что реальное количество искомых переменных может доходить до тысячи и более, в моделях рекомендуется под разными именами использовать до 50 отдельных групп переменных.

Переменная, свободная в декларации от связей с **set** (*множеством*), является скалярной переменной, т.е. участвует в расчетах вне какого-либо ассоциированного множества.

Поясняющий текст при декларации переменных очень важен, так как этот же самый текст будет аннотировать решение в выводном файле, поэтому он должен быть написан настолько наглядно и понятно, насколько это возможно. Обратите внимание, что в примере декларации применяется слово "**per**" вместо символа "**/**", потому что наклонная вправо черта недопустима в тексте без кавычек.

7.2.2. ТИПЫ ПЕРЕМЕННЫХ

Существует 5 базовых типов переменных.

Ключевое слово	**Нижний предел по умолчанию**	**Верхний предел по умолчанию**	**Описание**
free (default)	-inf	+inf	Переменная принадлежат всему множеству действительных чисел от плюс бесконечности до минус бесконечности. Обе границы могут быть изменены пользователем при объявлении границ.
positive	0	+inf	Переменная принадлежит всему множеству положительных чисел. Верхняя граница может быть изменена пользователем при объявлении границ
negative	-inf	0	Переменная принадлежит всему мно-жеству отрицательных чисел. Нижняя граница может быть изменена пользователем при объявлении границ
binary	0	1	Дискретная переменная, которая может принимать значения только 0 и 1.
integer	0	100	Дискретная величина, которая может принимать целочисленные значения внутри границ. Границы могут быть изменены при объявлении в модели верхнего и нижнего предела

По умолчанию тип переменной принимается автоматически - **free**, он означает, что если тип переменной не особенный, то переменная может принимать все действительные значения от минус бесконечности до плюс бесконечности. Наиболее часто используемые типы переменных - **free** и **positive**. **Positive** описывает переменные, для которых отрицательные значения не имеют смысла (такие как объем, частота или цена).

В GAMS имеется еще четыре дополнительных типа переменных: **sos1, sos2, semicont** и **semiint**. Эти типы следует рассмотреть дополнительно и позже.

7.2.3. СТИЛИ ОБЪЯВЛЕНИЯ ПЕРЕМЕННЫХ

Обычно применяется два способа объявления типа переменных.

В первом способе сначала записывается зарезервированное слово **variables**, затем перечисляются все переменные со спецификациями доменов и поясняющим текстом, а затем переменные группируются по их типам. Пример, приведенный ниже, взят из [MEXSS]. Тип переменной по умолчанию - **free**, поэтому в примере переменные **phi**, **phipsi** будут иметь тип **free**. Обратите внимание, что использование имени переменных взято из первоначального математического представления.

Пример

```
  variables
    u(c,i)       "purchase of domestic meterials "
    v(c,j)       "imports "
    e(c,i)       "exports "
    phi          "total cost "
    phipci       "raw material cost "

positive variables u, v, e;
```

Имена переменных при объявлении их типа (в данном случае в **positive variables**) должны разделяться запятыми.

Идентификатор можно декларировать более одного раза, но вторая и последующие декларации должны давать новую дополнительную информацию, которая не должна противоречить уже введенной информации.

Во втором способе декларации переменных переменные записываются группами в соответствии с их типами. Ниже приведен тот же пример, но записанный вторым способом.

Пример

```
free variables
    phi          "total cost "
    phipci       "raw material cost ";

positive variables
    u(c,i)       "purchase of domestic meterials "
    v(c,j)       "imports "
    e(c,i)       "exports "
```

Выбор между двумя подходами должен базироваться на ясности и прозрачности записи.

7.3. ХАРАКТЕРИСТИКИ ПЕРЕМЕННЫХ

Другое важное отличие между **variable** и **parameter** заключается в том, что для того, чтобы описать различные характеристики переменных может быть использован дополнительный набор ключевых слов.

В GAMS **parameter** (*параметр*) имеет по одному числу, ассоциированному с каждой отдельной комбинацией меток. **Variables** (*переменные*) имеют шесть чисел, ассоциированных с каждой комбинацией меток. Вы были ознакомлены с ними в разделе 2. Они приведены ниже.

Характеристики переменных	**Суффикс с переменных**	**Описание**
Нижняя граница	.lo	Нижняя граница переменной. Устанавливается либо пользователем, либо автоматически через значения по умолчанию.
верхняя граница	.up	Верхняя граница для переменной. Устанавливается либо пользователем, либо автоматически через значения по умолчанию
фиксированное значение	.fx	Фиксированное значение для переменной.
активный уровень	.l	Активный инициаторный уровень для переменной перед обработкой в операторе SOLVE. Также эквивалент для текущего значения переменной, которое получится после выполнения оператора SOLVE.
маргинальное или двойственное значение или теневая цена	.m	Значение для критериальной переменной на границе области решения, которое обеспечивается единичным шагом расчетной переменной в перпендикулярном к этой границе направлении.
масштаб значения ветвящееся	.scale	Это фактор масштабирования на переменную. Запрашивается обычно в нелинейном программировании и детально будет рассмотрен позднее.
приоритетное значение	.prior	Это приоритетное значение переменной. Этот параметр используется только в смешанных моделях работающих с целочисленными переменными и детально будет обсужден позднее.

Пользователь может использовать эти суффиксы, присоединив суффикс к имени переменной. Правила использования этих суффиксов будет рассмотрено в разделе 7.4.

7.3.1. ОСОБЫЕ ТОЧКИ ДЛЯ ПЕРЕМЕННЫХ ВНУТРИ ОБЛАСТИ ОПРЕДЕЛЕНИЯ

Все границы по умолчанию, устанавливаемые во время декларации, могут быть изменены, используя оператор присвоения.

Значения границ для переменных типа **binary** и **integer**, которые устанавливается во время декларации, не могут быть полностью исправлены.

За границы переменных полностью ответственен пользователь. После того, как переменные продекларированы, границы по умолчанию уже присвоены: во многих случаях, особенно в линейном моделировании, границ по умолчанию бывает достаточно. С другой стороны, в нелинейном моделировании границы играют более важную роль. Может быть ситуация, когда необходимо назначить границы, чтобы предотвратить неопределенные операции, такие как деление на ноль. Так же часто необходимо определить "разумную" область решения, что поможет сделать решение задачи нелинейного программирования более эффективным.

Нижняя граница не может быть выше, чем верхняя. Если случайно было сделано такое назначение границ, GAMS выдаст сообщение об ошибке.

7.3.2. ФИКСИРОВАНИЕ ПЕРЕМЕННОЙ

GAMS позволяет установить переменные через суффикс переменной **.fx.** Она является эквивалентом нижней границы и верхней границы и равна фиксированному значению. Фиксированные переменные впоследствии могут быть освобождены изменением верхних и нижних границ.

7.3.3. АКТИВНЫЙ УРОВЕНЬ ПЕРЕМЕННЫХ

GAMS позволяет пользователю фиксировать активный уровень переменных с помощью суффикса переменной **.l**. Эти активные уровни переменных предшествуют оператору **solve** и служат начальным значением для solver. Это исключительно важно для задач нелинейного программирования.

7.4. ПЕРЕМЕННЫЕ В ОПЕРАТОРАХ DISPLAY И ПРИСВОЕНИЯ

GAMS позволяет составителю модели использовать значения, ассоциированные с различными характеристиками каждой переменной в операторах присвоения и **display**. Следующие два подраздела объясняют использование переменных с левой и с правой стороны оператора присвоения. Подраздел 7.4.3 объясняет использование переменной в операторе **display**.

7.4.1. ПРИСВОЕНИЕ ЗНАЧЕНИЙ К ХАРАКТЕРИСТИКАМ ПЕРЕМЕННЫХ

Операторы присвоения оказывают действие на одну характеристику переменной и требуют указать суффикс, чтобы определить, какая из характеристик переменной используется. Перечень индексов записывается после суффикса.

Следующий пример иллюстрирует использование операторов присвоения при установлении верхних границ переменных.

```
x.up(c,i,j) = 1000;    phi.lo = inf;
p.fx('pellets', 'ahmsa', 'mexico-fd') = 200;
c.l(t) = 4*cinit(t);
```

Обратите внимание, в первом операторе индексы, относящиеся к домену **x**, появляются после суффикса. Первое присвоение устанавливает верхние границы на все переменные, ассоциированные с идентификатором **x**. Оператор на второй

линии ограничивает один частный ввод. Оператор на последней линии устанавливает значения уровня переменных, и они равны учетверенному значению параметра **cinit**.

Помните, что в присвоениях порядок записи очень важен. Например, две пары нижеприведенных операторов приведут к различным результатам. В первом случае нижняя граница для **c('1985')** будет равной **0.01**, а во втором - нижняя граница равна **1**.

```
c.fx('1985') =1;   c.lo(t) = 0.01;
c.lo(t) = 0.01;    c.fx('1985') = 1;
```

При назначении границ все работает, как описано в предыдущем разделе, включая различные механизмы описанных здесь индексных операций, долларовых операций, присвоение подмножеств и т.д.

7.4.2. ХАРАКТЕРИСТИКИ ПЕРЕМЕННОЙ В ПРИСВОЕНИИ

Использование характеристик переменных с правой стороны операторов присвоения очень важно с различных точек зрения. Существует два общих применения для генерирования отчетов и генерирования первоначальных значений для переменных, базирующихся на значениях других переменных.

Следующий пример, взятый из [CHENERY] иллюстрирует использование характеристик переменных с правой стороны операторов присвоения

```
scalar  cva  "total value added at current prices "
        rva  "real velue added"
        cli  "cost of living index "

 cva = sum (i, v.l(i)*x.l(i));
 rva = sum (i, p.l(i)*ynot(i))/ sum(i, ynot(i));
 cli = cva/cli;

display cli, cva, rva;
```

Также как и в **parameters** (*параметр*), **variable** (*переменная*) должна иметь ассоциированные с ней числовые значения (не по умолчанию) прежде, чем она будет использоваться в операторе **display** или с правой стороны оператора присвоения.

Переменная получит числовое значение после использования оператора присвоения (переменная слева) или в результате работы оператора SOLVE (работа оператора SOLVE будет рассмотрена в разделе 9).

Суффикс **.fx** является только стенографией для одновременной работы с суффиксами **.lo** и **.up** и поэтому может использоваться только с левой стороны оператора присвоения.

7.4.3. ХАРАКТЕРИСТИКИ ПЕРЕМЕННЫХ, ВЫВОДИМЫХ НА ПЕЧАТЬ

Когда переменные используются в операторе **display**, пользователю необходимо обозначить какое из 6 значений переменных должно быть выведено на печать. Это достигается посредством присоединения соответствующих суффиксов к имени переменной. Никакой информации об объединяющих **set** в **variables** (*переменные*) при этом не включается. Вывод происходит по всему комплексу переменных для всего множества индексов set. На примере, взятом

из [MEXSS], показан, вывод на печать (**to display**) расчетное значение **phi**, расчетное значение и теневая цена (маргинал) переменой **v**.

display phi.l, v.l, v.m;

В выводном файле (output) содержится та же информация, но распечатка показывает, какие именно значения выводятся на печать. При выводе на печать нули, и особенно все строки и колонки, содержащие нули, пресекаются, поэтому состав, выводимых на печать расчетных значений будет отличаться от состава выводимых на печать теневых цен (маргиналов), так как ненулевые значения теневых цен (маргинальных значений) часто ассоциируются с нулевыми активными уровнями.

Действительно, значения по умолчанию пресекаются при выводе на печать в выходном файле (output), так как нулевые вводы не выводятся на печать, а значения параметров и уровней переменных по умолчанию равны нулю. Однако для границ значение по умолчанию может быть ненулевым. Например, значение по умолчанию для верхней границы положительной переменной может быть равна **+inf**, и если записать **display v.up**, то в распечатке будет (**ALL +INF**). Если какая либо граница была изменена (т.е. имеет значение не по умолчанию), тогда на печать будет выведено только измененное значение.

7.5. ЗАКЛЮЧЕНИЕ

Помните, что где бы ни появился параметр в операторе **display** или операторе присвоения, переменная также появится - при условии, что она определяется одним из четырех суффиксов. Место, где имя переменной появляется без суффикса, является декларацией переменной (которая рассматривалась в этом разделе) или определением уравнения (которое будет рассмотрено в следующем разделе).

УРАВНЕНИЯ (EQUATIONS) 8

8.1. ВВЕДЕНИЕ

В GAMS **equation** (*уравнение*) - это ключевое слово, предваряющее список имен уранений, входящих в состав решаемой модели (первый шаг). Затем (второй шаг) приводятся алгебраические выражения уравнений. Как и в **variable** (*переменных*), одно имя **equations** (*уравнения*) автоматически декларирует (первый шаг) и создает (второй шаг) всю группу частных уравнений в соответствии с объединенным составом индексов, определенных в **set** (*множестве*).

8.2. ДЕКЛАРАЦИЯ УРАВНЕНИЙ

В GAMS **equations** (*уравнения*) декларируются прежде, чем их начинают использовать.

8.3. СИНТАКСИС УРАВНЕНИЙ

Декларация **equations** (*уравнений*) сходна с декларацией **set** (*множеств*) или **parameter** (*параметров*) в том, что при декларации допускается и рекомендуется наличие поясняющего текста.

Одним оператором - ключевым словом **equation(s)** можно продекларировать несколько уравнений.

equation(s) eqn_name text {, eqn_name} ;

В GAMS **eqn_name** - внутреннее имя **equation** (*уравнения*) (или идентификатор). Идентификатор может начинаться с буквы, затем следуют буквы и цифры. Он может включать в себя только алфавитные литеры и может быть до 10 литер длиной. Поясняющий текст используется, чтобы описать уравнение или группу уравнений и располагается сразу после имени уравнения. Поясняющий текст не должен превышать 80 литер и должен располагаться на той же линии, что и описываемый идентификатор - имя уравнения.

В поясняющем тексте нельзя использовать зарезервированные слова и имена продекларированных ранее переменных. При декларации **equetions** (*уравнений*) не происходит инициализации каких-либо числовых значений или списков данных, как это допускается в декларациях **parameters** (*параметров*) и **sets** (*множеств*).

8.2.2. ПРИМЕР ДЛЯ ИЛЛЮСТРАЦИИ

Пример взят из [PRODSCH] - задачи управления инвентаризацией и производством. В примере также приведены декларации соответствующих **sets** (*множеств*).

```
sets   q    'quarters'  /summer, fall, winter, spring/
       s    'shifts'    /first, second/;

equations

 cost        'total cost definition'
 invb(q)     'inventory balance'
 slab(q,s)   'shift employment balance'  ;
```

Декларация первого уравнения следует за ключевым словом **equation**. Эта декларация начинается с имени уравнения (в данном случае - **cost**), затем следует поясняющий текст (**'total cost definiton'**). Уравнение **cost** - это скалярное уравнение. В ассоциированных оптимизационных задачах должно быть как минимум одно скалярное уравнение.

Уравнение **slab** декларируется через **set** (*множество*) **q**, включающего 4 элемента, и через **set** (*множество*) **s**, включающего 2 элемента, что дает 8 различных уравнений (по одному для каждой из 8 комбинации элементов **s** и **q**). Случай, когда за счет введения ограничивающих факторов уравнений становиться меньше 8, будет рассмотрен ниже. В данном примере составляются все уравнения (для всех восьми комбинаций) **slab**. Не может составляться больше уравнений, чем число комбинаций элементов множеств, через которые декларируется уравнение.

8.3. ОПРЕДЕЛЕНИЕ УРАВНЕНИЙ

Определения - математическая особенность уравнений в языке GAMS. Следующие подразделы посвящены синтаксису определения уравнений с поясняющими примерами. Остальные подразделы развивают обсуждение о некоторых ключевых компонентах определения уравнения.

8.3.1. СИНТАКСИС

Синтаксис определения уравнений в GAMS следующий:

```
eqn_name(domain_list)..  exprassion eqn_type  expression ;
```

eqn_name - это имя уравнения, которое получено при его декларации. Две точки **'..'** всегда разделяют название уравнения и начало алгебраического выражения. Выражения в определении уравнения можно выполнять в формах, обсужденных в разделе 6. Оно также может быть выполнено, как в переменных. **Eqn_type** – это символ между двумя выражениями, которые формируют уравнение. Имеются следующие типы уравнений и соответствующие им символы:

тип уравнения	описание
=e=	Равенство: правая часть равна левой.
=g=	Больше чем: правая часть больше или равна левой.
=l=	Меньше чем: правая часть меньше или равна левой.
=n=	Нет принудительного соотношения между левой и правой частями. Этот тип уравнений используется редко.

Как и в операторе присвоения, уравнения могут занимать неограниченное число строк. Для более наглядного и удобного восприятия выражений используются пробелы.

Однажды определенные уравнения не могут быть изменены или переопределены. Если такая необходимость возникает, то необходимо выполнить определение нового уравнения с новым именем. Однако возможно изменить значение уравнения путем изменения данных, которые оно использует, или используя регулирующий механизм исключения, встроенный в определение – долларовые операции, которые будут рассмотрены в дальнейшем.

8.3.2. ПРИМЕР

Рассмотрим следующий пример, взятый из [MEXSS]. В пример также включены ассоциированные декларации:

```
variables   phi, phipsi, philam, phipi, phieps;
equations   obj;

 obj..
        phi  =e= phipsi + philam + phipi - phieps;
```

obj - имя уравнения, которое определяется. Символ **=e=** означает, что это равенство. Любые другие виды записи уравнения (см. ниже) являются математическими эквивалентами

```
 obj..  phi - phipsi =e=  philam + phipi - phieps;
 obj..  phi - phipsi - philam =e=  phipi - phieps;
 obj..  phi - phipsi - philam - phipi =e=  phieps;
```

Расположение элементов в уравнении - дело выбора пользователя, поэтому часто выбирается такая конструкция уравнения, которая делает модель легкой в понимании и прозрачной.

8.3.3. СКАЛЯРНЫЕ УРАВНЕНИЯ

В ассоциированных оптимизационных задачах должно быть по крайней мере одно скалярное уравнение. Уравнение, определенное в разделе 8.3.2 является примером скалярного уравнения, которое содержит только скалярные переменные. Однако, в общем случае, скалярные уравнения могут содержать индексные переменные, стоящие внутри индексных операторов - **SUM** и **PROD**. Рассмотрим следующий пример из [CHENERY]

```
   dty..   td =e=  sum(i, y(i));
```

В результате применения индексного оператора **sum** происходит уничтожение зависимости для индекса **i**.

8.3.4. ИНДЕКСНЫЕ УРАВНЕНИЯ

В скалярных уравнениях все ссылки к множеству находятся в пределах зоны действия индексных операций - поэтому в одно уравнение может быть включено много ссылок. Однако GAMS позволяет определять уравнения через домен (управление), что делает возможным компактное представление ограничений по индексным множествам.

Индексное множество, стоящее слева от символа две точки **".."**, называется **"домен определения"** уравнения.

Проверка домена гарантирует, что домен через который определяется уравнение, должен быть множеством или подмножеством множества индексов, через которые декларируется уравнение.

Рассмотрим следующий пример индексного уравнения, означающего, что для каждого члена управляющего (или контролирующего) множества индексов создается отдельное (частное) уравнение.

```
dg(t)..  g(t)  =e=  mew(t) + xsi(t)*m(t);
```

Пусть **t** имеет три члена, тогда необходимо задать три ограничения по **t** для каждой из расчетных переменных **g** и **m**. **Mew** и **xsi** являются параметрами: данные, ассоциированные с ними, используются в составлении частных ограничений. Когда определяется уравнение, эти элементы могут быть только продекларированы, но при решении модели, включающую эти уравнения, им должны быть присвоены численные значения.

Если число индексов с левой стороны '..' более одного, то результат очевиден. Для каждой комбинации индексов, которые можно составить из индексов в круглых скобках, будет образовано одно частное ограничение. Ниже приведены два примера из [AIRCRAFT]

```
bd(j,h)..  b(j,h)  =e= dd(j,h) - y(j,h);
yd(j,h)..  y(j,h)  =l= sum(i, p(i,h)*x(i,j));
```

Домен определения этих уравнений - это декартовое произведение **j** и **h**: будут созданы ограничения для каждой пары индексов-меток, которые могут быть составлены из элементов этих двух **sets** (*множеств*).

8.3.5. ЯВНОЕ ИСПОЛЬЗОВАНИЕ МЕТОК В УРАВНЕНИИ

Очень часто в уравнениях необходимо использовать метки явно. Это выполняется также как в **parameter**, с помощью кавычек. Рассмотрим следующий пример.

```
dz.. tz =e= y('jan') + y('feb') + y('mar') +y('apr');
```

8.4. ВЫРАЖЕНИЯ В ОПРЕДЕЛЕНИИ УРАВНЕНИЯ

Арифметические операторы и функции, которые были описаны в разделе 6, могут быть также использованы в уравнениях.

8.4.1. АРИФМЕТИЧЕСКИЕ ОПЕРАТОРЫ В ОПРЕДЕЛЕНИЯХ УРАВНЕНИЙ

Все способы, которые могут быть использованы при вычислении выражений, могут быть использованы и в **equations** (*уравнениях*).

Рассмотрим пример из [CHENERY], где выполняется возведение в степень с использованием скобок

```
dem(i)..  y(i)   =e=   ynot(i)  *  (pd *  p(i))**thet(i)  ;
```

8.4.2. ФУНКЦИИ В ОПРЕДЕЛЕНИЯХ УРАВНЕНИЙ

Функции, представленные в определении уравнений, могут быть разделены на два типа в соответствии с типами аргументов.

Экзогенные аргументы.
Случай когда аргументы известны. Параметры и характеристики переменных (например, характеристики **.l** или **.m**) используются в качестве аргументов. Численно выражение рассчитывается один раз, в момент создания модели, и допускаются все функции, кроме случайно распределенных функций **uninform** и **normal**.

Эндогенные аргументы.
Случай когда аргументы являются переменными и поэтому неизвестными. В процессе решения модели, функция будет рассчитываться численно много раз в промежуточных точках.
Если в модели имеется какая-либо функция с эндогенными аргументами, то модель нелинейна.
В определениях уравнения запрещено использовать функции **uninform** и **normal**.

Функции с эндогенными аргументами классифицируются следующим образом.

Тип	Функция	Производная	Пример
Однородная	Непрерывная	Непрерывная	exp, sin, log
Неоднородная	Непрерывная	Прерывная	max, min, abs
Прерывная	Прерывная	Прерывная	ceil, sign

Однородная функция используется в соответствии с правилами нелинейного моделирования. Неоднородная функция численных задач может использоваться в случаях, если этого невозможно избежать, но только в специальных моделях, называемых **dnlp**. Однако, использование модели типа **dnlp** является очень сложной задачей и в нелинейных функциях рекомендуется использовать бинарные переменные. Разрывные функции недопустимы для аргументов переменных.

Классификация всех функций дана ниже.

Функции	Описание	Классификация	Экзогенная классификация	Эндогенная классификация
abs	Абсолютное значение	неоднородная	допустимая	DNLP
arctan	арктангенс	однородная	допустимая	NLP
ceil	максимальная цена	однородная	допустимая	недопустима
cos	косинус	прерывная	допустимая	NLP
errorf	функция ошибок	однородная	допустимая	NLP
exp	экспонента	однородная	допустимая	NLP
floor	минимальная цена	прерывная	допустимая	недопустимая
log	натуральный логарифм	однородная	допустимая	NLP
log10	десятичный логарифм	однородная	допустимая	NLP
mapval	функция преобразования	прерывная	допустимая	недрпустимая
max	наибольшее значение	неоднородная	допустимая	DNLP
min	наименьшее значение	неоднородная	допустимая	DNLP
mod	остаток	прерывная	допустимая	недопустимая
normal	нормальная случайная	недопустимая	недопустима	недопустимая
power	целая степень	однородная *	допустимая	NLP
round	округление	прерывная	допустимая	недопустимая
sign	знак (-1 или +1)	прерывная	допустимая	недопустимая
sin	синус	однородная	допустимая	NLP

sgr	возведение в квадрат	однородная	допустимая	NLP
sqrt	квадратный корень	однородная	допустимая	NLP
trunc	усечение	прерывная	допустимая	недопустимая
uniform	равномерно частотное распределение	недопустимая	недопустима	недопустимая

Классификация дается только для первого аргумента. Второй аргумент должен быть целочисленным (**integer**) и обычно является константой.

8.4.3. ПРЕДОТВРАЩЕНИЕ НЕОПРЕДЕЛЕННЫХ ОПЕРАЦИЙ В УРАВНЕНИЯХ (EQUATIONS)

Существуют операции, которые могут быть неопределенными при некоторых значениях аргументов. Например, функция **log** является неопределенной, если значение аргумента равно нулю. Деление на ноль является другим примером. Эта проблема легко решается для экзогенных функций и выражений, и это вызывает большие сложности, когда исходные числа - **variables** (*переменные*). Когда задача решается выражение может вычисляться много раз. Одним из путей предотвращения того, что выражение в какой-либо промежуточной точке станет неопределенным, является выделение опасной зоны в соответствующем ограничении для переменной.

Рассмотрим следующую функцию из модели [RAMSEY], предшествующую появлению в модели ограничивающих переменных

```
c.lo(t) = 0.1;
util..  utility  =e= sum(t, beta(t)*log(c(t)));
```

Ограничение, наложенное на **c(t)**, которое говорит, что **c(t)** отличается от нуля, что позволяет предотвратить неопределенность функции **log**.

8.5. АСПЕКТЫ УПРАВЛЕНИЯ ДАННЫМИ ПЕРЕМЕННЫХ (VARIABLES)

В предыдущем разделе мы имели дело со свойствами алгебраических уравнений. В этом разделе рассматривается другой аспект уравнений, он также служит в качестве данных.

Как и для переменных, с каждой отдельной "меткой-кортежем" (отдельной комбинацией меток) каждого уравнения ассоциируются четыре важнейших значения данных. В практике они используются, в основном, для обозначения цели после оператора SOLVE, и здесь в связи с нехваткой места обсуждение будет дано в очень сжатой форме. Суффиксы, ассоциированные с четырьмя значениями такие же, как в **variables** (*переменных*): **.l, .m, .lo, .up**. Они могут быть использованы в операциях присвоенния (что случается редко), или в выражениях, или выводятся на печать (что является наиболее обычным, особенно для маргинала **.m**). Однако все эти элементы будут доступны для данных действий только в случае предварительного решения модели. Значение характеристик **lo, .l, .up** может быть введено комплексно, но предпочтительнее это делать для каждого частного ограничения, чем для общего символьного уравнения.

После того, как получено решение, уже существует значение, ассоциированное с ранее неизвестным термом[1], расположенным слева, который по определению является **.l**. Значения **.lo** и **.up** для каждого типа уравнений показаны в нижеприведенной таблице в форме констант с правой стороны уравнений (**rhs**) и в форме переменных с левой стороны **(.l**). Связь между **rhs** и **.l** удовлетворяется только в случае, когда ограничение совпадает с точкой решения.

Тип	**.lo**	**.up**	**.l**

=e=	rhs	rhs	Rhs
=l=	-inf	rhs	≤rhs
=g=	rhs	inf	≥rhs
=n=	-inf	inf	any

Значение маргинальной величины **(.m)** определяется для критерия оптимизации и обсуждается в большинстве работ математического программирования. Грубым, но удачным определением маргинала является следующее: маргинал это такое число, при которым целевая функция была бы изменена таким образом, что определяющий комплекс переменных в уравнении был бы передвинут на единицу в направлении перпендикулярном границе.

8.6. ЗАКЛЮЧЕНИЕ

В этом разделе описаны декларация и определение **equations**(*уравнений*).

[1] Терм - элементарный член арифметического или логического выражения, именующий элемент данных или являющийся значением функции или константы.

ОПЕРАТОРЫ MODEL И SOLVE 9

9.1. ВВЕДЕНИЕ

В этом разделе при описании способов определения и решения модели сводятся вместе все понятия, введенные в предыдущих разделах.

9.2. ОПЕРАТОР MODEL

Оператор **model** используется для того, чтобы объединить уравнения в группы и обозначить их таким образом, чтобы они могли быть решены. Самая простая форма оператора **model** использует ключевое слово **all**: в этом случае модель включает в себя все уравнения, продекларированные перед вводом оператора **model**. Для ряда задач это все, что необходимо знать об операторе **model**.

9.2.1. СИНТАКСИС

В основном синтаксис декларации **model** следующий:

```
model(s)  model_name [text] [/all | eqn_name {, eqn_name } /]
      {, model_name [text] [/all | eqn_name {, eqn_name } /]} ;
```

В GAMS **model_name** - внутреннее имя скаляра (также называется идентификатором). Поясняющий текст используется, чтобы описать все модели или каждую модель отдельно, и расположен сразу же за описываемым объектом. **Eqn_name** - имя уравнения, которое декларируется до ввода оператора **model**.

Как и все идентификаторы, **model_name** начинается с буквы и далее может состоять из букв и цифр в различном порядке. Он может включать в себя только алфавитные литеры и может быть длиной до 10 литер. Поясняющий текст не может быть длиной более 80 литер и должен располагаться на той же линии, что и идентификатор или метка, которая его описывает.

Пример определения **model** показан ниже

```
model  transport  "a transportation model"  /all/;
```

Модель называется **transport**, а ключевое слово **all** означает, что в модель входят все известные (продекларированные) уравнения.

В одном операторе **model** может быть продекларировано несколько моделей. Это удобно, когда мы хотим, на базе одних и тех же данных, составить, просмотреть и проанализировать различные варианты моделей.

Рассмотрим следующий пример из [PROLOG], в котором используются различные группы уравнений для составления альтернативных вариантов задачи. Решаются три варианта - линейная, нелинейная и "расходы".

```
model   nortonl  "linear version"       /cb,rc,dfl,bc,obj/
        nortonn  "nolinear version"     /cb,rc,dfn,bc,obj/
        nortone  "expenditure version"  /cb,rc,dfe,bc,obj/ ;
```

где **cb, rc, dfl, bc, obj, dfn, dfe** - имена уравнений.

В дальнейшем мы рассмотрим, как получить решение для каждой из этих трех моделей.

9.2.2. КЛАССИФИКАЦИЯ МОДЕЛЕЙ

С помощью GAMS могут быть решены различные типы задач. Тип модели должен быть известен до ее решения. В этом разделе дано краткое описание типов моделей. GAMS контролирует, какой тип модели существует фактически, и сравнивает его с типом модели, который назвал пользователь. Если обнаруживается несоответствие типов, то выдается сообщение об ошибке. Например, если была выбрана линейная модель, а она содержит нелинейные элементы, то будет сообщение об ошибке с поясняющим описанием.

Очень часто задача может быть решена различными способами (различными типами моделей), и только пользователь решает, какой из способов выбрать. Например, если в модели имеются бинарные и целочисленные переменные, то пользователь может выбрать или модель **MIP**, или модель **RMIP**, он также может выбрать оба способа, решить задачу, проанализировать результаты расчета и затем выбрать наиболее подходящий тип модели.

Типы моделей	**Описание**
LP	Линейное программирование (linnear programming). Модель не содержит нелинейных элементов или дискретных (бинарных или целочисленных) переменных.
NLP	Нелинейное программирование (nonlinear programming). В модели основными нелинейными термами являются только непрерывные функции, но она не содержит дискретных переменных. В таблице из раздела 8.4.2 функции классифицированы как однородные.
DNLP	Нелинейное программирование с функциями, имеющими разрыв в производных (nonlinear programming with discontinuouse derivatives). Это такой же тип модели, как и NLP, но в ней могут иметь место функции, которые в таблице из раздела 8.4.2 классифицированы как "неоднородные". Эта задача решается более сложно, чем обычная задача NLP. Пользователю рекомендуется избегать данного типа модели.
RMIP	Релаксированное смешанное целочисленное программирование (relaxed mixed integer programming). Эта модель может содержать в себе дискретные переменные, но необходимыми условиями к дискретности является релаксированность, означающая, что целочисленные и бинарные переменные могут принимать любое значение в пределах их границ.
MIP	Смешанное целочисленное программирование (mixed integer programming). Оно подобно RMIP, но требования к дискретности жесткие. Дискретные переменные должны принимать целочисленные значения внутри границ.
RMINLP	Релаксированное смешанное целочисленное нелинейное программирование (relaved mixed integer nonlinear programming). Модель может содержать в себе и дискретные переменные, и общие нелинейные термы. Требованиями к дискретности релаксируются. Этот класс задач с точки зрения сложности решения такой же как NLP.
MINLP	Смешанное целочисленное нелинейное программирование (mixed integer nonlinear programming). Характеристика такая же, как для RMINLP, но требования к дискретности жесткие.

MCP	Смешанная дополнительная задача (Vixed Complementary Problem)
CNS	Жестка нелинейная система (Constrained Nonlinear System)

Каждый из этих типов будет обсужден в последующих разделах.

9.2.3. ХАРАКТЕРИСТИКИ МОДЕЛИ

Различные характеристики модели могут быть доступны пользователю через список суффиксов модели. Полный список суффиксов модели приведен ниже.

Суффиксы	**Описание**	**Приведение в исходное состояние по умолчанию**	**По умолчанию**	**Глобальный параметр**
Bratio	Базисные испытания на приеме	Пользователь	0.25	Bratio
Domlim	Максимальное число нарушений домена	Пользователь	0	Domlim
Domusd	Число нарушений домена	solver		
Holdfixed	подстановка фиксированных переменных	Пользователь	0	
0	не подставляется фиксированных переменных			
1	подстановка фиксированных переменных из модели			
iterlim	лимит итераций	Пользователь	1000	Iterlim
iterusd	число использованных итераций	Пользователь		
limcol	число колонок выведенных на печать для каждого блока переменных	Пользователь	3	Limcol
limrow	число строк выведенных на печать для каждого блока уравнений	Пользователь	3	Limrow
modelstat	состояние модели	solver		
1	оптимальная			
2	локально оптимальная			
3	неограниченная			
4	невозможное			
5	локально невозможное			

6	промежуточное невозможное			
7	промежуточное неоптимальное			
8	целочисленное решение			
9	промежуточное целочисленное			
10	целочисленное невозможное			
11	(неиспользуемая)			
12	неизвестная ошибка			
13	не ошибка решения			
numegu	количество одиночных (частных) генерируемых уравнений	solver		
numinfes	количество невозможностей	solver		
numnopt	количество не оптимизаций	solver		
numnz	количество ненулевых вводов в матричных коэффициентах	solver		
numunbnd	количество неограниченных переменных	solver		
numvar	количество одиночных (частных) генерируемых переменных	solver		
optca	абсолютный критерий завершения для MIP	Пользователь	0.0	Optca
optcr	относительный критерия завершения для MIP	Пользователь	0.1	Optcr
optfile		Пользователь	0	
reslim	Лимит времени для решения. Обычно в секундах CPU	Пользователь	1000	Reslim
resusd	запас единиц (в секундах CPU), используемых для решения модели	solver		
scaleopt	масштаб параметра	Пользователь	0	
solprint	решение распечатать замену	Пользователь	1	solprint
solveopt	соединение или перемещение параметров расчетных данных	Пользователь	1	solveopt
solvestat	состояние solver	solver		
1	нормальное завершение			
2	итерация прерывания			
3	ресурс прерываний			
4	завершенное solver			
5	лимит расчетной ошибки			
6	неизвестное			
7	(неиспользуемое)			
8	неуспех ошибочного препроцессора			
9	неудача ошибочного solver			
10				
11	ошибка ошибочного внутреннего solver			
12	ошибка ошибочного постпроцессора			
13	неуспех ошибочной системы			

sysout	подсистема	Пользователь	0	Sysout
workspace	размер работающего массива (в МБ)	Пользователь		Work

Следующий пример иллюстрирует использование суффиксов модели

```
model  transport /all/;
transport.reslim = 60;
```

Эта запись указывает **solver** верхний предел равный 60 секундам для попытки решить оптимизационную задачу.

9.3. ОПЕРАТОР SOLVE

Однажды прокомпилированную через оператор **model** модель можно пытаться решить. Для этого существует оператор **solve**. В зависимости от типа модели GAMS вызывается соответствующий solver.

Важно помнить, что GAMS сам не решает задачу, а направляет решение задачи к одной из решающих программ.

9.3.1. СИНТАКСИС

В общем случае, синтаксис для декларации **solve** следующий

```
solve  model_name using model_type maximizing|minimizing var_name|;
solve  model_name   maximizing|minimizing var_name using model_type ;
```

Model_name - это имя модели, определенное оператором **model**. **Var_name** - имя целевой переменной, которая оптимизируется. **Model_type** - один из типов модели, описанных в подразделе 9.2.2.

Ниже приведен пример оператора **solve**

```
Solve transport using lp minimizing  cost;
```

Solve и **using** - зарезервированные (ключевые) слова, **transport** - имя модели, **lp** - тип модели, **minimizing** - требования к оптимизации, **cost** - целевая переменная. В противоположность **minimizing** существует **maximizing** - оба зарезервированные слова.

Обратите внимание, что целевые переменные используются вместо целевых строк или функций.

Целевая переменная должна быть скаляром и иметь тип **free**. В модели должно быть по крайней мере одно уравнение, включающее такую переменную.

В следующих двух подразделах будет кратко описано, что происходит при выполнении оператора **solve**. Затем будет подробно описано, как выводить результаты расчетов и как их интерпретировать. Далее будет рассмотрено, каким образом составлять программу, когда имеется ряд операторов **solve**. В заключение раздела будут описаны параметры, которые важны при контроле оператора **solve**.

9.3.2. ТРЕБОВАНИ К СОСТАВЛЕНИЮ ОПЕРАТОРА SOLVE

Когда GAMS встречает оператор **solve**, должны уже быть выполнены следующие условия:

- Все символьные уравнения определены и целевая переменная должна быть по крайней мере в одном уравнении.
- Целевая переменная должна быть скаляром и иметь тип **free**.
- Каждое уравнение предназначено для решения определенного класса задач (линейное для **lp**, непрерывная производная для **nlp** и т.д. в соответствии с тем, как было схематизировано ранее).
- В уравнениях все **sets** (*множества*) и **parameters** (*параметры*) имеют присвоенные значения.

9.3.3. ДЕЙСТВИЯ, ЗАПУСКАЕМЫЕ ОПЕРАТОРОМ SOLVE

Оператор **solve** запускает выполнение ряда операций, которые описаны ниже:

- Модель преобразуется в форму, требуемую решающей системой, для дальнейшего использования.

- Выполняется поиск и исправление ошибок в разрабатываемой программе и создаются поясняющие сообщения, которые записываются в файл вывода (output) (например, в список уравнений).

- GAMS проверяет, не имеется ли в задаче непоследовательных границ или неприемлемых значений (например, **NA** или **UNDF**).

- Любая выявленная на этой стадии ошибка является причиной завершения с обоснованием и описанием ошибки, настолько подробно насколько это возможно. Чтобы идентифицировать причину затруднений при решении модели используются соответствующие методы заложенные в GAMS.

- GAMS передает контроль решающей подсистеме и ждет, когда будет решена задача.

- GAMS дает сообщения о состоянии процесса решения задачи и направляет результаты расчетов обратно в базу данных GAMS. Формируются новые значения, которые присваиваются к полям **.l** и **.m** для каждого частного уравнения и переменной в модели. Распечатка решения выполняется по умолчанию (строка за строкой, колонка за колонкой). Любые сложности в процессе решения вызывают поясняющие сообщения, которые выводятся на печать. Ошибки, причинами которых являются запрещенные нелинейные операции, сообщаются на этой стадии.

Выводы, получаемые в результате выполнения каждой из перечисленной операции, включая любое возможное сообщение об ошибках, подробно рассматриваются в следующем разделе.

9.4. ПРОГРАММЫ С НЕСКОЛЬКИМИ ОПЕРАТОРАМИ SOLVE

В одной программе могут обрабатываться несколько операторов **solve**. Если необходимо решить ряд дорогостоящих или трудных моделей, то сначала нужно изучить раздел 14, где ознакомиться с **workfiles**, найти, каким образом можно прерывать, а затем продолжать выполнение программы.

В следующих пяти подразделах приведены различные примеры, где в одном и том же файле необходимо применять несколько операторов **solve.**

9.4.1. НЕСКОЛЬКО МОДЕЛЕЙ

Если имеются различные модели, тогда операторы **solve** могут быть последовательными, как показано ниже. Каждая из моделей (пример из [PROLOG]) включает в себя различные группы уравнений, но так как каждая из них использует одни и те же данные, то все три модели могут быть последовательными

```
solve  nortonl  using  nlp  maximizing   z;
solve  nortonn  using  nlp  maximizing   z;
solve  nortone  using  nlp  maximizing   z;
```

Когда в модели больше, чем один оператор **solve**, GAMS в каждом последующем решении использует как можно больше результатов предыдущих решений.

9.4.2. ВОСПРИИМЧИВОСТЬ (ЧУВСТВИТЕЛЬНОСТЬ) АНАЛИЗА К ВАРИАНТАМ СОСТАВЛЕНИЯ МОДЕЛИ

Группа операторов **solve** может быть использована не только для того, чтобы решить различные модели, но также для выполнения тестов на чувствительность модели. GAMS позволяет выполнять анализ результатов решения задачи, когда в модели изменяются данные или границы переменных, и затем эта же модель решается снова. Коммерческие системы LP позволяют выполнять анализ модели на чувствительность посредством GAMS, однако, можно выполнить более всесторонний анализ, если не ограничиваться типом solver или типом модели. Некоммерческий solver обеспечит получение такой информации.

В модели "Очищение нефти", взятой из [MARCO], имеется пример оценки на чувствительность. При контроле загрязнения одним из ключевых параметров модели "очищение нефти" является верхняя граница содержания **sulfur** в топливном масле, получаемом при очистке. В этом примере верхняя граница содержания **surful** в топливном масле задается очистным заводом, и в первом варианте она была назначена равной 3.5 % от первоначальных данных. Первая модель решается с этим значением. Следующее значение содержания **surful** несколько ниже и равно 3.4 %. Модель решается снова с новым значением содержания **surful**. Заключительное значение устанавливается равным 5 % и модель с этим значением решается последней. После каждого решения ключевые расчетные значения (активный уровень, ассоциируемый с **z**, состояние уровней в соответствии с процессом **p** и типом неочищенной нефти **cr**) сохраняются для последующих отчетов. Это необходимо, потому что следующее решение восстанавливает любое существующее значение. Полная последовательность показана ниже

```
parameter report (*, *, *)  "prosecc level report" ;
qs ('upper', 'fuel-oil', 'sulfur') = 3.5 ;
solve oil using lp maximizimg phi ;
report (cr, p, 'base') = z.l(cr,p) ;

report ('sulfur', 'limit', 'base') = qs('upper','fuel-oil', 'sulful');
qs('upper','fuel-oil','sulfur') = 3.4 ;
solve oil using lp maximizimg phi ;
report (cr, p, 'one') = z.l(cr,p) ;
```

```
report ('sulfur', 'limit', 'one') = qs('upper','fuel-oil', 'sulful');
qs('upper','fuel-oil','sulfur') = 5  ;
solve oil using lp maximizimg phi ;
report (cr, p, 'two') = z.l(cr,p) ;
report ('sulfur', 'limit', 'two') = qs('upper','fuel-oil', 'sulful');

    display report;
```

Этот пример показывает не только, каким образом может быть выполнен анализ на чувствительность, но и каким образом можно просмотреть результаты всех расчетов. Параметр **qs** используется, чтобы установить верхнюю границу на содержание **surful** в топливном масле и значение восстанавливается для отчета.

Вывод, который создается в результате выполнения оператора **display**, показан ниже. Требуется учесть, что в модели не рассматриваются все аспекты производства для случая, когда уменьшено допустимое содержание **surful**. "**Case attributes**" записаны в строке **SULFUR.LIMIT**. Домен "**Wild card**" очень полезен при составлении отчета. Любая ошибка, сделанная в написании меток, используемых только в отчете, должна быть немедленно выявлена, и их действие должно ограничиваться отчетом. Раздел 13 содержит более подробную информацию о способах оформления отчетов.

```
 -----  205   PARAMETER REPORT    PROCESS LEVEL REPORT

                          BASE        ONE        TWO

MID-C  .A-DIST          89.718                 35.139
MID-C  .N-REFORM        20.000                  6.772
MID-C  . CC-DIST         7.805                  3.057
w-TEX  .CC-GAS-OIL                              5.902
w-TEX  .A-DIST                                 64.861
w-TEX  .N-REFORM                               12.713
w-TEX  .CC-DICT                                 4.735
w-TEX  .HYDRO                                  28.733
SULFUR.LIMIT              3.5         3.4        5.0
```

9.4.3. ПОВТОРЯЮЩЕЕСЯ ИСПОЛЬЗОВАНИЕ НЕСТАНДАРТНЫХ АЛГОРИТМОВ

Другое использование модели с рядом операторов **solve** заключается в том, что разрешены повторяющиеся решения различных блоков уравнений, расчетные значения первого решения используются как данные в последующих. Эти методы разложения позволяют успешно решать определенный класс задач, потому что подзадачи всегда менее сложные и поэтому легче решаются. Одним из наиболее распространенных примеров такого метода является метод Generalized Bender's Decomposition.

Пример задачи, которая решается данным способом является система ввода-вывода (input-output) с эндогенными ценами, описанными в Htnaff (1980). Модель состоит из двух групп уравнений. Первая группа уравнений использует заданный конечный вектор требований, чтобы установить уровень вывода (output) в каждом секторе. Вторая группа уравнений использует экзогенные состояния и данные ввода-вывода (input-output), чтобы рассчитать секторные уровни цен. Затем цены, полученные в результате предыдущего расчета, используются, чтобы установить новый вектор конечных требований и два блока уравнений решаются снова. Эту процедуру повторяют до тех пор, пока не будет достигнута удовлетворительная сходимость. Чтобы выполнить такого

рода расчеты Henaff использовал GAMS. Операторы, которые решали систему для первого случая и последующие повторения, приведены ниже

```
model   usaio          /mb, output/;
model   dualmodel      /dual, totp/;

solve usaio       using  lp  maximizimg  total ;
solve dualmodel   using  lp  maximizing  totprice;

pbar(ta)  = (sum(ipd.l(i,ta))/4.);
d(i,t)    = (db(i)*q(t)/(pd.l(i,t)/pbar(t));

solve usaio       using  lp  maximizimg  total ;
solve dualmodel   using  lp  maximizing  totprice;
```

mb - это набор вещественных балансовых **input-output** (*ввод-вывод*) уравнений, а **output** (*вывод*) - суммарный вывод уравнений. **Dual** - это группа ценовых уравнений, которая суммирует все секторные цены. Внутренняя цена **pd** использовала в расчетах среднюю цену **pbar** деленную на четыре, потому что в этом примере имеется четыре сектора. Суффикс **.l** прибавляется к **pd**, чтобы указать, что **pd** - это уровень переменных в решении модели, которой присвоено имя **dualmodel**. Эти повторяющиеся процедуры используют расчетные значения одного решения, чтобы получить значения параметра для следующего расчета. В частности, и **pbar**, и **pd** используются для того, чтобы рассчитать требование **d** для i-того результата в период времени **t, d(i,t).** В этом расчете также используются базовое годовое требование **db** и увеличивающийся фактор **g.** Затем, когда рассчитано новое значение для конечного вектора требования **d**, два блока уравнений решаются снова.

9.5. СОСТАВЛЕНИЕ НОВЫХ SOLVRES, ДОСТУПНЫЕ С GAMS

Этот короткий раздел написан для тех, кто имеет любимый **solver** не доступный GAMS. Соединение **solver** программ с GAMS - это открытая задача, и мы (GAMS DEVELOPMENT CORPORATION)можем предоставить документы с необходимой информацией и предоставить исходную программу, которая использована для существующих соединений. Разработчики **solver** получают следующие преимущества от соединения с GAMS

- Немедленный доступ к широкому кругу тестовых задач.
- Легкий способ выполнения сопоставления решений.
- Гарантию того, что пользователь не выполнит недопустимый ввод спецификации.
- Не требуется тщательно сделанной документации, особенно в формате input;
- Доступ к существующему обществу пользователей GAMS с целью маркетинга или тестирования.

Этим заканчивается обсуждение операторов model и solve. В следующем разделе будут детально описаны компоненты вывода GAMS.

ВЫВОД (OUTPUT) GAMS 10

10.1. ВВЕДЕНИЕ

Вывод, получаемый в результате завершения работы GAMS, содержит много информации, которая позволяет выявлять ошибки на стадии отладки программы и контролировать правильность составления модели. В этом разделе приводится полное содержание выводного файла (файла output) и показано, каким образом использовать эту информацию. На примере небольшой нелинейной задачи Алана Манне [PORTFOL] поэтапно будут рассмотрены различные компоненты файла output. Способы перехода от небольшой модели к большим моделям с многотомным output (для больших моделей диагностика действительно необходима) будут очевидны после ознакомления с данным описанием.

Вывод (output) одного расчета GAMS выполняется в один выводной файл, который может быть прочитан любым текстовым редактором. Имя файла output по умолчанию зависит от операционной системы (в приложении А описано, каким образом можно изменить имя файла output по умолчанию). Информация из программы GAMS в распечатываемый выводной файл переносится при помощи оператора **display** (оператор **display** детально описан в разделе 13.)

10.2. МОДЕЛЬ - ИЛЛЮСТРАЦИЯ

[PORTFOL] - это модель "выбора папок". Задача модели – выбор "портфеля" капиталовкладчика, который удовлетворяет запланированный доход при условии минимизации дисперсии.

Полный текст файла input (вводного файла) модели [PORTFOL] приведен ниже.

```
$title      A quadratic proggramming model for portfolio analysis
название    программная модель второго порядка для анализа портфеля

$offupper
$ontext
     This formulation is described in 'GAMS/MINOS: Three exampels'
by Alan S. Manne, Department of Operations Research, Standford
University, May 1986.
Эта редакция описана в "GAMS/MINOS: Три примера" Аланом Мане, Министерство прикладных
исследований, Стендфордский университет, Май 1986.

$offtext
* Alterations by A. Brooke to include a display statement and
* some comments
*  изменения, выполненные А.Бруком, включают оператор display и некоторые комментарии

sets i securities          /hardware   оборудование
       обеспечение          software   программное обеспечение
                            show-biz   рекламные образцы
                            t-bills          /;
alias (i,j);

scalar target          target mean annual return on portfolio %  /10/
       план             плановая среднегодовая прибыль портфеля в %
       lowyield         yield of lowest yielding security
       наинизший        доход, получаемый при наинизшем доходе от
```

```
        доход              обеспечения

       highrisk        variance of highest security risk;
       наивысший       дисперсия наивысшего риска  обеспечения
       риск
parameter mean(i)   mean annual returns on individual securities (%)
          среднее   среднегодовая прибыль от отдельных видов  обеспечения


                     /hardware  оборудование              8
                      software  программное обеспечение   9
                      show-biz  рекламные образцы         12
                      t-bills                            7   /;

Table v(i,j)  variance-covariance array (%-squared annual return)
              дисперсионно-ковариационный массив (%-квадрата  годового  дохода)

              hardware       software     show-biz

hardware         4               3            -1
software         3               6             1
show-biz        -1               1            10

lowyield   =  smin(i, mean(i));
highrisk   =  smax(i, v(i,i));
display  lowyield, highrisk ;

* in GAMS the default variable type is free
  в GAMS переменные по умолчанию имеют тип free

variables  x(i)          fraction of portfolio invested in asset i
                         доля портфеля, инвестируемого в актив i

           variance      variance of portfolio ;
           дисперсия      дисперсия портфеля

positive variable  x;

equations  fsum    fractions must add up to 1.0
                   определение доли, которые долны добавляться до 1.0

           dmean   definition of mean return on portfolio
                   определение среднего дохода портфеля

           dvar    definition of variance  ;
                   определение дисперсии

fsum..     sum(i, x(i))  =e= 1;
dmean..    sum(i, mean(i)*x(i))  =g= target ;
dvar..     variance  =e= sum(i, x(i)*sum(j, v(i,j)*x(j))) ;

model portfolio   /all/;
solve portfolio  using nlp minimizing variance;
```

10.3. КОМПИЛЯЦИОННЫЙ ВЫВОД (OUTPUT)

Output (вывод), получаемый во время первоначального контроля программы, часто называют компиляцией. Он состоит из двух или трех частей:

- эхо - принт программы;
- сообщения о выявленных ошибках;
- справочные карты.

Следующие подразделы рассказывают об этом подробно.

10.3.1. ЭХО-ПРИНТ ВВОДНОГО ФАЙЛА

Первой частью выводного файла (output) всегда является эхо - принт. Он представляет собой распечатку вводного файла (input) с пронумерованными строками. Директива **$offlisting** отключает распечатку вводного файла.

```
43       dvar    definition of variance  ;
44
45  sum..     sum(i, x(i))  =e= 1;
46  mean..    sum(i, mean(i)*x(i))  =g= target ;
47  var..     variance  =e= sum(i, x(i)*sum(j, v(i,j)*x(j))) ;
48
49  model portfolio   /all/;
50  solve portfolio  using nlp minimizing variance;
```

Так как наш вводный файл не содержал ошибок, то заключительная часть эхо-принт вводного файла (input) выглядит, как показано выше. Если при компиляции выявляются ошибки, то ниже распечатки вводного файла приводятся сообщения, поясняющие причины этих ошибок.

10.3.2. СИМВОЛЬНАЯ СПРАВОЧНАЯ КАРТА

Для тех, кто изучает модель, составленную другим, или пытается выполнить изменения в модели по прошествию некоторого времени, совершенно необходимо иметь справочную карту модели. Она состоит из двух частей. Первая часть карты представляет собой символьные перекрестные ссылки. В ней приведен перечень идентификаторов (символов) модели в алфавитном порядке, затем описан тип идентификатора, затем указан номер строки, где данный идентификатор появляется, и в конце дана классификация каждой ссылки на идентификатор. Карта символьных ссылок закрывается путем ввода в начало программы строки - **$offsymxref.**

Карта, полученная при выполнении модели [PORTFOL] показана ниже.

```
A quadratic programming model for portfolio analysis
Symbol listing

SYMBOL      TYPE    REFERENCES

DMEAN       EQU     DECLARED     42   DEFINED     46  IMPL-ASN   50
                       REF      49
DRAV        EQU     DECLARED     43   DEFINED     47  IMPL-ASN   50
                       REF      49
FSUM        EQU     DECLARED     41   DEFINED     45  IMPL-ASN   50
                       REF      49
HIGHRISK    PARAM   DECLARED     16   ASSIGNED    31        REF  32
```

```
I            SET     DECLARED     11   DEFINED     11        REF   12
                          18     24         3    2*31         36   45
                       2*46   2*47   CONTROL     30         31   45
                         46     47
J            SET     DECLARED     12       REF     24       2*47
                      CONTROL     47
LOWYIELD     PARAM   DECLARED     15   ASSIGNED    30        REF   32
MEAN         PARAM   DECLARED     18   DEFINED     19        REF   30
                         46
PORTFOLIO    MODEL   DECLARED     49   DEFINED     49   IMPL-ASN   50
                         REF     50
TARGET       PARAM   DECLARED     14   DEFINED     14        REF   46
V            PARAM   DECLARED     24   DEFINED     24        REF   31
                         47
VARIANCE     VAR     DECLARED     37   IMPL-ASN    50        REF   47
                         50
X            VAR     DECLARED     36   IMPL-ASN    50        REF   39
                          45     46              2*47
```

Для каждого символа (идентификатора) сначала указывается имя и тип символа (идентификатора). Например, последним записан символ (идентификатор) **X**, который определяется как тип **VAR**. Ниже приведен полный перечень типов идентификаторов GAMS и соответствующие им записи в карте символьных ссылок

Запись в карте символьных ссылок	**Тип идентификаторов GAMS**
SET	set
PARAM	parameter
VAR	variable
EQU	equation
MODEL	model

Затем следует перечень ссылок к символам, сгруппированных по типу ссылок, и указывается номер линии в файле output, где данный символ встречается. Фактически ссылка на символ обосновывается на базе эхо-принт программы. В вышеприведенном примере для символа **X** перечень ссылок в карте символьных ссылок выглядит следующим образом

```
DECLARED      36
IMPL-ASN      50
REF           39     45     46     2*47
```

Это означает, что **X** декларируется на линии номер **36**, неявно выполнено присвоение через оператор **solve** на линии **50**, и на него ссылаются на линии **39, 45** и **46**. Запись **2*47** означает, что на линии **47** файла input (вводного файла) на **X** имеется две ссылки.

Полный список типов ссылок приведен ниже.

Ссылка	**Описание**
DECLARED	Идентификатор декларируется как тип. Это должно быть первое появление идентификатора.
DEFINED	Служебное слово, за которым последует номер линии, на которой начинается инициализация: символа (идентификатора) (таблица или список данных между двумя наклонными чертами) или символьное определение (equation (уравнения).
ASSIGNED	Служебное слово, за которым указывается номер линии, на которой изменятся численное значение идентификатора

	(идентификатор появился слева от оператора присвоения).
IMPL-ASN	Служебное слово, за которым указывается номер линии, где происходит "неявное присвоение": **equation** (уравнение) или **variable** (переменная) изменятся, если на них имеется неявная ссылка в операторе **solve**.
CONTROL	Служебное слово, за которым указывается номер линии, где происходит использование **set** (множества) в качестве управляющего индекса в присвоении, **equation** (уравнении), цикле и других индексных операциях (**sum, prod, smin** или **smax**).
REF	Служебное слово, за которым указывается номер линии, где происходит ссылка: символ указывается с права присвоения, в **display**, в **equation** (уравнении), в операторе **model** или **solve.**

10.3.3. КАРТА ПЕРЕЧНЯ СИМВОЛОВ

Следующая карта называется "перечень символов" ("**symbol listing**"). Все идентификаторы группируются в алфавитном порядке по типам и записываются с их поясняющим текстом. Это другая очень удобная возможность GANS для тех, кто впервые изучает большую модель, составленную другим. Карта перечня символов закрывается путем ввода в начале программы строки - директиву **$offsymlist.**

Ниже приведена карта "перечня символов" для модели [PORTFOL].

```
SET

I             securities
J             Aliased with I

PARAMETERS

HAGHRISK      variance of highest security risk
LOWYIELD      yield of lowest yielding security
MEAN          mean annual returns on individual securities (%)
TARGET        tagert mean annual return on portfolio %
V             variance-covariance array (%-square annual return)

VERIABLES

VARIANCE      veriance of portfolio
X             fraction of portfolio invested in asset i

EQUATION

DMEAN         defifnition of mean return on portfolio
DVAR          definition of veriance
FSUM          fractions must add up to 1.0

MODELS

PORTFOLIO
```

10.3.4. ДИРЕКТИВА ДОЛЛАРОВОГО КОНТРОЛЯ

В этом подразделе рассматриваются наиболее полезные аспекты директивы[1] долларового контроля директив (**Dollar Control Directives**). Не нужно путать директиву долларового контроля с долларовыми операторами управления исключениями, с которыми мы имели дело в разделах 6 и 8. **Dollar Control Directives** является указанием транслятору[2], которое может быть помещено в файле ввода (input файл), чтобы контролировать внешний вид и количество деталей в выводе (output), выполняемом компилятором GAMS. Полный перечень представлен в последующих параграфах.

10.4. ВЫПОЛНЕНИЕ ВЫВОДА (OUTPUT)

Пока GAMS выполняет программу (осуществляет манипуляции с данными), вывод в файл распечатки осуществляется только через оператор display. Форма этого вывода кратко была описана в разделе 6. Все имеющиеся в GAMS способы изменения формата вывода сведений, получаемых при решении модели, подробно описаны в разделе 13. В данном разделе приводится вывод (output) оператора display, расположенный на линии 41 модели [PORTFOL]. Обратите внимание на перенос[3] поясняющего текста.

```
A quadratic programming model for portfolio analysis
E x e c u t i o n

----   32 PARAMETER  LOWYIELD          =  7.000 yield of lowest
                                                yieldding security

           PARAMETER HIGHRISK          = 10.000 variance of highest
                                                security risk
```

Если на данном этапе решения модели обнаруживаются ошибки использования недопустимых операций с данными, появляется краткое сообщение о причине ошибки и указывается номер линии, где использован недопустимый оператор.

10.5. ВЫВОД, ПОЛУЧАЕМЫЙ В РЕЗУЛЬТАТЕ ОПЕРАТОРА SOLVE

В этом разделе будет приведено содержание вывода (output), формируемого в момент выполнения оператора **solve**. Все действия, которые запускаются оператором **solve**, перечислены в разделе 9. Все выводы, представляющие собой результат выполнения оператора **solve**, помечаются подзаглавием, идентифицирующим модель, тип модели и номер линии, где расположен оператора **solve**.

---

[1]Директива - компонент программы на языке ассемблера, управляющий последующей компоновкой программы, но не вызывающий появление машинной команды. Примеры директив: выбор режима адресации генерируемой программы; начало нового сегмента; выделение в памяти места для констант и переменных.

[2]Указание транслятору – конструкция входного языка, не имеющая смысл программы, но управляющая работой транслятора или задающая ему какие-либо параметры, например, вид оптимизации или формат распечатки.

[3] Перенос - функция устройства визуального отображения, позволяющая выводить строки текста, которые при другом способе вывода слишком длины для представления их полностью; в данном случае длинная строка выводится на экран в виде нескольких последовательных строк. Такое деление строки на более короткие, соответствующие

размеру экрана, выполняется электронной частью устройства визуального отображения и не требует включения в поток данных форматирующих символов.

10.5.1. СПИСОК УРАВНЕНИЙ (THE EQUATION LISTING)

Первый вывод (output), получаемый в результате выполнения оператора **solve** - это список уравнений. Для выделения начала списка уравнений в выводном файле (файле output) записывается подназвание **"Equation Listing"**. В общем случае по умолчанию в список уравнений заносится не более трех частных уравнений, которые можно составить на базе общего уравнения (т.е. если общее уравнение включает в себя три и менее частных уравнений, то по умолчанию в список уравнений включаются все уравнения, если общее уравнение включает в себя более трех частных уравнений, то в список заносятся только первые три уравнения).

Раздел **Equation Listing** из модели [PORTFOL] приведен ниже. [PORTFOL] содержит три блока уравнений и в каждом блоке по одному частному уравнению.

```
A quadratic programming model for portfolio analysis

Egietion listing   SOLVE PORTFOLIO USING NLP FROM LINE 50

---- FSUM     =E=  fraction must add up to 1.0

FSUM..   X(HARDWARE) + X(SHOW-BIZ) + X(T-BILLS) =E=  1;
         (LNS = 0 ***)

---- DMEAN      =G= definition of mean return on portfolio

DMEAN..  8*X(HARWARE) + 9*X(SOFTWARE) + 12*X(SOW-BIZ) +7*X(T-BILLS)
          =G= 10; (LHS =0***)

---- DVAR      =E= definition of veriance

DVAR..   (0)*X(HARDWARE) + (0)*X(SOFTWARE) + (0)*X(SHOW-BIZ) +
         VERIANCE   =E=  0;  (LHS =0)
```

Список уравнений очень полезен при отладки модели. В каждом блоке он показывает переменные, которые имеют место, все частные коэффициенты с левой стороны и численное значение с правой стороны, которое вычисляется после того, как выполнена манипуляция с данными.

Большая часть информации, представляемая в списке уравнений самодостаточна. В каждом блоке приводится информация об именах, типах и поясняющие тексты. Для механического поиска предусмотрен знак четыре тире "----".

Все термы , которые включают в себя переменные, переносятся на лево от знаков **=e=, =l=, =g=,** а все постоянные термы (константы) комбинируются в одно число и переносятся на право, при этом выполняются все необходимые изменения знаков.

Если необходимо, в числе показывают четыре знака после запятой, и показывать больше знаков после запятой запрещено. Чтобы обеспечить вывод маленьких числовых значений, которые при данных ограничениях на вывод будут отображаться как ноль, используют E-формат.

Нелинейные уравнения в линейных аппроксимациях вывода трактуются различно. Но если коэффициент уравнения переменной в списке уравнений не заключен в скобках, тогда соответствующий вывод отображает нелинейные

уравнения, и значение коэффициента зависит от текущего расчетного уровня одной или более переменных. Занесение в список не является алгебраическим, но показывает частные значения каждой переменной, рассчитанной по текущему расчетному значению переменных.

Обратите внимание, что в списке уравнений из [PORTFOL] уравнение **dvar** нелинейное. Простой пример поможет нам пояснить этот момент. Рассмотрим следующее уравнение и ассоциированные значения расчетных переменных

```
eq1..  2*sqr(x) * power(y,3) + 5*x -1.5/y =e= 2;     y.l=3;
```

тогда список уравнений будет выглядеть так

```
EQ1..  (221)*X + (216.1667)*Y  =2= ;  (LHS = 225.5 ***)
```

Коэффициент **x** определяется первой линеализацией по **x** вышеприведенного уравнения. В результате получаем выражение **2*(2*x.l)*power(y.l,3) +5**, которое после подстановки всех значений дает число 221. Такой же коэффициент для **y** определяется путем линеализации по **y** вышеприведенного уравнения, в результате получим выражение **2*(sqr(x.l)*3* sqr(y.l) +1.5/sqr(y.l),** которое
дает значение **216.1667**. Напомним, что коэффициент **y** не может быть определен, если его расчетное значение оказалось равным нулю. Попытка деления на ноль привела бы к ошибке и преждевременному прерыванию решения модели.

Численный результат расчета левой стороны уравнения в инициаторной точке показан в конце каждого частного представления уравнения, занесенного в список уравнений. В вышеприведенном примере это число **225.5**; знак три звездочки **"***"** является предупреждением, что ограничение невыполнимо в инициаторной точке (точке взятой за основу данной частной линеаризации).

Последовательность, в которой записаны уравнения в списке уравнений, зависит от того, каким образом была определена модель. Если модель была определена со списком имен уравнений, тогда составление списка уравнений будет выполнено в соответствии с этим списком. Если модель была определена как /**all**/, тогда список уравнений будет составлен в соответствии с последовательностью декларации уравнений. Последовательность частных уравнений в блоке списка уравнений, формируемом на базе общего уравнения путем комбинирования индексов термах уравнений, определяются последовательность индексов в определении **set**.

10.5.2. СПИСОК КОЛОНОК (THE COLUMN LISTING)

Следующим разделом распечаткии файла output является **Colum Listing** (Список колонок). Это список частных отсортированных коэффициентов. По умолчанию сортировка производиться по колонкам, а не по строкам. Как и ранее, по умолчанию показываются первые три конкретных значений из каждой группы переменных вместе со значениями границ и расчетного значения. Формат для них такой же, как и для коэффициентов и списков уравнений, с нелинейными коэффициентами незаключенными в круглые скобки, конечные нули опускаются.

Раздел "Список колонок" для модели [PORTFOL] следующий

```
GAMS 5.2.087  386/486   DOS        07/14/96 16:26:49  PAGE 6
A quadratic programming model for portfolio analysis
Column listing  SOLVE PORTFOLIO USING NLP FROM LINE 50

---- X          fraction of portfolio insteted in asset i
```

```
X(HARDWARE)
                (.LO, .L, .UP  = 0, 0, +INF)
        1       FSUM
        8       DMEAN
       (0)      DVAR

X(SOFTWARE)

                (.LO, .L, .UP  = 0, 0, +INF)
        1       FSUM
        9       DMEAN
       (0)      DVAR

X(SHOW-BIZ)

                (.LO, .L, .UP  = 0, 0, +INF)
        1       FSUM
       12       DMEAN
       (0)      DVAR

REMAINING ENTRY SKIPPED

----  VARIANCE

VARIANCE
                (.LO, .L, .UP  = 0, 0, +INF)
        1       DVAR
```

Последовательность, в которой появляются переменные, соответствуют последовательности, в которой они были продекларированы.

10.5.3. СТАТИСТИКА МОДЕЛИ (THE MODEL STATISTICS)

Последним во время выполнения подготовки модели к решению воспроизводится статистический блок (**Model Statistics**). Наиболее очевидное использование этого блока заключается в том, чтобы в деталях показать размеры модели и ее нелинейность.

Ниже приведена **Model Statistics** для модели [Portfol]

```
A quadratic programming model for portfolio analysis
Model statistics      SOLVE PORTFOLIO USING NLP FROM LINE 50

MODEL STATISTICS

BLOKS OF EQUATION      3    SINGLE EQUATIONS      3
BLOKS OF VARIABLES     2    SINGLE VERIABLES      5
NON ZERO ELEMENTS     12    NON LINEAR N-Z        3
DERIVATIVE POOL       10    COCNSTANT POOL       10
COD LENGHT            94

GENERATION TIME       =     0.110 SECONDS

EXECUTION TIME        =     0.280 SECONDS    VERID MW2-25-087
```

В блоке (**BLOK**) подсчитывается число ссылок к уравнениям и переменным GAMS. В блоке (**SINGLE**) подсчитывается число частных строк и колонок в

воспроизводимой задаче. Блок (**NON ZERO ELEMETS**) информирует о числе ненулевых коэффициентов в матрице.

Имеется еще 4 элемента, которые предоставляют дополнительную информацию о нелинейных моделях. **NON LINEAR N-Z** отсылает к числу нелинейных матричных элементов.

Нелинейность в зависимости от вида уравнения имеет разную степень сложности. Например, **x*y** - это более простая форма нелинейности, чем **exp(x*y**). Любой из этих термом будет рассматриваться как 1 нелинейный объект в матрице, поэтому для пользователя требуется дополнительная информация о сложности нелинейности. Для этой цели GAMS предусматривает в качестве критерия **COD LENGHT**. Два других элемента - **DERIVATIVE POOL** и **CONSTANT POOL** предусматривают несколько больше информации о нелинейности. В общем случае, чем более не линейна задача, тем более трудна она в решении.

Время выполнения также указывается в **Model Statistics. GENERATION TIME** - время, используемое на контроль синтаксиса. В это время включено время, затраченное на воспроизводство модели. Единицы измерения времени даются в единицах, установленных в часах на персональном компьютере или на центральном процессоре (CPU) другой машины.

Здесь предусмотрена информация о состоянии **solver** и состоянии **model**, которые будут детально описаны в конце этого раздела.

****** ЦЕЛЕВОЕ ЗНАЧЕНИЕ (OBJECTIVE VALUE) 2.8990**

Она обеспечивает значение целевой функции при завершении **solve**. Если **Solver** и **Model** имеют взаимоувязанное и безконфликтное состояние, это значение является оптимальным значением для задачи.

РЕСУРС[1] ОБРАЩЕНИЯ, ЛИМИТ (RESOURSE USAGE, LIMIT) 0.109 1000.000

Эти два элемента предусматривают время CPU (в секундах), взятое **solver**, также как верхний предел, предоставленный для **solver. Solver** остановится, как только закончится лимит на время обращения. Лимит на время обращения по умолчанию составляет **1000** секунд. Этот лимит может быть изменен вводом в программу перед оператором **solve** строки, содержащей оператор **option reslim = xx** ;, где **xx** - требуемый лимит на время CPU в секундах.

ЧИСЛО ИТЕРАЦИЙ[2], ЛИМИТ (ITERATION COUNT, LIMIT) 5 1000

Эти два элемента предусматривают число итераций, используемых **solver**, также как верхний лимит, предоставленный для **solver. Solver** остановится, как только будет достигнут этот лимит. Лимит по умолчанию на используемые итерации равен **1000**. Этот лимит может быть изменен путем ввода в программу перед оператором **solve** строки, содержащей оператор **option itrlim = nn;**, где **nn** - требуемый лимит на используемые итерации.

ПОДСЧЕТ ОШИБОК (EVALUATION ERRORS) 0 0

Эти два блока вывода предусматривают число встреченных **solver** численных ошибок, также как верхний лимит, предоставленный для **solver**. Эти ошибки

---

[1]Ресурс - любой из компонентов вычислительной системы и предоставляемые ею возможности. Все вычислительные системы должны включать в себя один или несколько процессоров, одновременно выполняющих действия с хранимой информацией, некоторый тип ЗУ, в котором хранятся как команды для процессора, так и ожидающие обработки данные, и устройства ввода/вывода, которые могут считывать информацию из "внешнего мира" или выдавать результаты во "внешний мир".

[2]итерация - повторение преобразования, приближающего к решению. Или шаг цикла.

обусловлены численными проблемами, такими как деление на **0**. Эти ошибки пресекаются для моделей типа **LP, RMIP, MIP** до тех пор, пока модель не

пройдет проверку на **EVALUATION ERRORS** для этих типов моделей. Лимит по умолчанию на **EVALUATION ERRORS - 0.** Этот лимит может быть изменен вводом в программу перед оператором **solve** строки, содержащей оператор **option domlim = nn;**, где **nn** - требуемый лимит на **EVALUATION ERRORS**.

НАЗНАЧЕННАЯ РАБОЧАЯ ОБЛАСТЬ[1] (WORK SPACE ALLOCATED) -- 0.4 Mb

[1]Рабочая область - блок ячеек оперативной памяти, который используется для временного хранения данных а течение обработки.

Этим предусматривается количество используемой **solver** памяти. Если памяти для **solver** недостаточно, то GAMS выдаст сообщение, что память не была назначена (**no memory was allocated**). GAMS также выдаст сообщение о количестве имеющейся в наличии памяти. Пользователь может указать **solver** использовать меньше памяти, вставив в программу оператор **mymodel.workspace = xx;**, где **mymodel** - имя модели для SOLVERa, и оно соответствует тому имени которое было назначено в операторе **model; xx** -количество памяти в мегабайтах. Обратите внимание, что **solver** попытается решить задачу с памятью в **xx** мегабайт, однако это не гарантирует успех решения, т.к. задача может требовать и больше памяти.

SOLVER STATUS (состояние **solver**) и **MODEL STATUS** (состояние модели) требуют специальных комментариев. Состояние для **solver** (состояние программы) и **model** (информация что решение выполняется нормально) снабжается специальными служебными сообщениями, и полный список таких возможных сообщений для **SOLVER STATUS** (состояние **solver**) и **MODEL STATUS** (состояние модели) приведен ниже.

Список возможных сообщений MODEL STATUS (состояние модели)

Model Status (состояние модели)	**Означает**
1 OPTIMAL	Это сообщение означает, что решение оптимально. Это применимо только к линейным задачам и к релаксированным смешанным целочисленным задачам **RMIP**.
2 LOCALITY OPTIMAL	Это сообщение означает, что оптимум локален. Сообщение показывает, что задача нелинейна, и все, что можно гарантировать при решении нелинейной данной задачи - это локальный оптимум.
3 UNBOUNDED	Это сообщение означает, что решение неограниченно границами возможного изменения переменных. Если задача линейная, это сообщение достоверное. Но иногда оно появляется для сложной нелинейной задачи, тогда сообщение означает не неограниченное решение, а что отсутствуют некоторые стратегические границы, чтобы ограничить переменные до разумных значений.
4 INFEASIBLE	Это означает, что линейная задача невыполнима и уравнения содержат непреодолимые противоречия. Возможно, что-то неправильно определено в логике или в данных.
5 LOCALITY INFEASIBLE	Это сообщение означает, что невыполнимый пункт мог возникнуть для нелинейной задачи из-за заданной начальной точки. Это не обязательно означает, что не существует решения вообще. Проверьте все ссылки на нелинейность задачи (раздел 15) того, чтобы выбрать способ продолжить решение.
6	Это означает, что текущее решение не выполнимо, и solver

INTERMEDIATE INFEASIBLE	программа остановилась, или из-за лимита (итерации или ресурса) или по другой причине. Проверь SOLVER STATUS (состояние solver) для получения более полной информации.
7 INTERMEDIATE NONOPTIMAL	Это сообщение означает, что решение не достигнуто, но оно есть и может быть найдено.
8 INTEGER SOLUTION	Целочисленное решение найдено для MIP (смешанной целочисленной задачи). Более детально это означает следующее: удовлетворяет ли это решение критерию завершения (множеству опций optcr или optca, рассмотренных в разделах 6 и 9).
9 INTERMEDIATE NONINTEGER	Это сообщение означает, найдено не целочисленное решение для MIP (смешанной целочисленной задачи). Целочисленное решение еще не найдено.
10 INTEGER INFEASIBLE	Это сообщение означает, что не имеется целочисленного решения для MIP смешанной целочисленной задачи). Это сообщение является достоверным.
ERROR UNCNOWN ERROR NO SOLUTION	Нет решения для любого из этих случаев. Посмотрите более внимательно, что может быть причиной этого.

СПИСОК ВОЗМОЖНЫХ СООБЩЕНИЙ SOLVER STATUS

Solver Status (состояние модели)	**Означает**
1 NORMAL COMPLETION	Это означает, что solver завершил работу в нормальном порядке: т.е. прерывание было не из-за лимита итерации или ресурса, или каких либо внутренних проблем. MODEL STATUS (состояние модели) описывает характеристики сопровождающие решение.
2 ITERATION INTERRUPT	Это означает, что solver был прерван, потому что было использовано слишком много итераций. Используйте опции (iterlim), чтобы увеличить лимит итераций, если все остальное нормально.
3 RESOURCE INTERRUPT	Это означает, что solver был прерван, потому что использует слишком много времени. Используйте опции (reslim), чтобы увеличить лимит времени, если все остальное нормально.
4 TERMINATED BY SOLVER	Это означает, что solver натолкнулся на трудности и не может продолжать выполнение программы. Больше деталей выявит последующее сообщение ниже - в листинге вывода.
5 EVALUATION ERROR LIMIT	Слишком много расчетов нелинейных термов неопределенных величин. Необходимо использовать границы, чтобы предотвратить запрещенные операции, такие как деление на ноль. Строки, в которых встречаются ошибки, записываются перед решением.
6 UNKNOWEN ERROR PREPROCESSOR (S) ERROR SETUP FAILURE ERROR SOLVER FAILURE	Все эти сообщения появляются, когда возникают непредвиденные отказы GAMS, solver или между ними. Просмотрите тщательно всю программу и попытайтесь найти ошибку

ERROR INTERNAL SOLVER ERRROR ERROR POST-PROCESSOR ERROR (S) ERROR SYSTEM FAILURE	

10.5.5. ОТЧЕТ SOLVER (SOLVER REPORT)

Следующий раздел в распечатке файла - это краткое описание решения, которое имеет особое значение в используемой **solver** программе. Этот раздел обычно начинается с сообщения, идентифицирующего **solver** и их авторов: в данном примере был использован **MINOS**. В этом разделе будут также диагностические сообщения об особенностях языка, если будет выявлено что-то необычное, и особенное в выполнении деталей, некоторые из них, возможно, будут технические. Здесь будет помогать справочник **solver** для пользователя. В случае серьезной неисправности в распечатке файла будут включены дополнительные сообщения, выводимые на печать **solver**. Это может помочь идентифицировать причины проблемы. Если сообщение **solver** не помогает, внимательно просмотри **solver** документацию или попроси помощи у более опытного пользователя. Вы можете получить больше информации для успешной работы программы, если введете в программу перед оператором **solve** строку, содержащую оператор **option sysout = on ;** .

Отчет **solver** из модели [PORTFOL] следующий

```
M I N O S    ---   VERSOIN        5.3   JUN    1991
= = = =

    B.A. Murtagh, University of New South Waies
         and
    P.E. Gill, W.Murray,   M.A. Saunders  and M.H. Wright
    System Optimization Laboratory,  Standford university

    EXIT – OPTIMAL SOLUTION FOUND
    MAJOR ITNS, LIMIT               1   200
    FUNOBJ, FUNCON CALLS           10     0
    SUPERBASICS                     2
    INTERPRETER USAGE            0.00
    NORM RG / NORM PI       3.738E-16
```

10.5.6. РАСПЕЧАТКА РЕШЕНИЯ

Следующий раздел распечатки файла - это записанное строка за строкой, колонка за колонкой решение, возвращенное GAMSy программой **solver**. Каждое частное уравнение и переменная записываются с четырьмя пунктами информации.

Этот раздел может быть исключен, если в программу выше оператора **solve** вставить строку с оператором **option solprint = off;.**

Расчетная часть распечатки файла им модели [PORTFOL] приведена ниже.

```
LOVER   LEVEL    UPPER   MARGINAL

---- EQU  FSUM     1.000    1.000    1.000   -13.529
---- EQU  DMEAN   10.000   10.000    +INF      1.933
---- EQU  DVAR       .        .        .       1.000
```

```
     FSUM        fractions must add up  to 1.0
     DMEAN       defifnition of mean return on portfolio
     DVAR        definition of veriance

----  VAR X      fraction of portfolio invested in asset i


                   LOVER   LEVEL    UPPER   MARGINAL

HARDWARE             .     0.303    +INF       .
SOFTWARE             .     0.087    +INF      EPS
SHOW-BIZ             .     0.505    +INF       .
T-BILLS              .     0.106    +INF      EPS


                           LOVER   LEVEL    UPPER   MARGINAL

---- VAR VARIANCE           -INF   2.899    +INF        .

VARIANCE       variance of portfolio
```

Порядок уравнений и переменных такой же, как в перечне символов, описанных в разделе 10.3.3 и который еще будет разделе 13. Четыре колонки, относящиеся к каждому элементу, имеют следующие описан в значения:

ЗАГОЛОВОК В listing file	ОПИСАНИЕ
LOWER	Lower Bound (.lo)
LEVEL	Level Value (.l)
UPPER	Upper Bound (.up)
MARGINAL	Marginal (.m)

Для переменных значения в колонках **LOWER** и **UPPER** указывают верхнюю и нижнюю границы. Для уравнений они получаются из значений с правой стороны уравнения (констант) и из относительного типа уравнения. Эти связи были описаны в разделе 8.

Значения **LEVEL** и **MARGINAL** определяются **solver** . В распечатке они показаны с заданной точностью, но в GAMS значения возвращаются с полной точностью машины. Знак точка "." в распечатке означает ноль.

EPS - это расширенное понятие GAMS, которое означает величину очень близкую к нулю, но отличающуюся от нуля. Обычно можно видеть, что **marginal** дают значение **EPS**, т.к. в GAMS используется соглашение, что **marginal** есть ноль для базисных переменных, но не равен нулю для других.

EPS используется с небазисными переменными, чьи **marginal** значения очень близи нулю, или действительно равны нулю, или в нелинейных задачах с супер-базисными переменными, чьи значения равны или близки нулю. Супербазисная переменная является переменной между ее границами в конечной точке, но не в базисе.

Словарь содержит краткое описание технических терминов, используемых в этом разделе.

В модели, которая не решается оптимально, могут быть дополнительно помечены определенными опознавателями.

Опознователь (flag)	Описание
INFES	Строка или колонка невыполнима. Этот знак устанавливается для любого элемента, чей LEVEL (уровень) значения не

NORT	находится в пределах верхней (UPPER) и нижней (LOWER) границ. Строка или колонка не оптимальна. Этот знак устанавливается для любого небазисного элемента, для которых знак маргинала неверен, или для супербазисного элемента, для которого маргинальное значение слишком большое.
UNBND	Знак, который появляется в стоке или колонке и указывает, что причина проблемы заключается в том, что не установлены границы.

10.5.7. КРАТКИЙ ОТЧЕТ (REPORT SUMMARY)

Заключительный раздел распечатки решения - это **Report Summury** (краткий отчет), помеченный четырьмя звездочками (как все важные компоненты вывода (output). Он показывает количество срок и колонок, которые помечены знаками **INFES, NORT** или **UNBND** в разделе распечатки решения. Общее число невыполнений будет указано, если будет сообщение, решение является невыполнимым. Ошибка в количестве конфликтов будет только в том случае, если задача
нелинейна.

```
**** REPORT SUMMARY:              0      NONOPT
                                  0   INFEASIABLE
                                  0     UNBOUNDED
                                  0      ERRORS
```

Если модель [PORTFOL] имела оператор **display**, это могло бы быть представлено здесь.

10.5.8. КРАТКИЙ ФАЙЛ (FILE SUMMARY)

Последняя часть файла вывода (файла output) очень важная. Здесь приводится имена дисковых файлов ввода (input) и вывода (output). Если используются рабочие файлы (save или resart), то они будут названы также и будут представлены в нижеследующей форме.

```
**** FILE SUMMARY

INPUT    C:\PORTFOL.GMS
OUTPUT   C:\PORTFOL.LST
```

10.6 СООБЩЕНИЕ ОБ ОШИБКАХ (ERROR REPORTING)

Все комментарии и указания об ошибках собираются в этот раздел для легкой оценки, когда решение закончилось неудачей.

Эффективное обнаружение ошибок и исправление - это важная часть в любой моделирующей системе. При разработке GAMS учитывалось, что "состояние ошибки" (**error state**) – это нормальное состояние моделирования. Опыт показывает, что большинство компиляций на ранней стадии разработки любой

модели будут содержать ошибки. Давно установлено, что компьютер намного лучше может проверить детали модели, чем это может сделать человек, и должна быть позитивная обратная связь и сообщения о том, каким образом исправить ошибки и избежать двухсмысленности. Разрабатываемую модель лучше сначала записать на бумаге; потребуется выполнение огромного количества черновой работы прежде, чем аргументы будут расставлены в самой эффективной форме для восприятия и выполнены все требования правильного английского языка (или любого другого прим. авторов перевода). GAMS требует, чтобы вы **в совершенстве владели математическим моделированием, синтаксисом и семантикой языка.**

Ошибка обнаруживается на различных стадиях моделирования. Большинство из них улавливается на стадии компилирования, которое хорошо работает на стадии чтения гранок моделируемого процесса. Если ошибка не обнаруживается на этой стадии, то вероятность ошибки снижается почти до нуля. Большая часть запусков выполнения, которые намного более долгие, чем компиляция, выполняется без проблем, потому что GAMS "знает" о сути моделирования и имеет упреждающие задачи. Многие типичные ошибки обусловлены языком программирования и ассоциируются с общими представлениями модельера. Это такие ошибки как - адресный расчет, сохранение присваивания, связывание подпрограмм, ввод/вывод и текущий контроль, которые создают проблемы времени выполнения, их трудно обнаружить, и они часто приводят к продолжительным поискам. GAMS взял радикально другой подход. Ошибки опознаются настолько рано, насколько это возможно, и о них сообщается в форме, понятной пользователю, включая рекомендации, каким образом откорректировать задачу, и представляют сведения об источнике ошибки в терминах составляемой задачи.

Все ошибки маркируются четырьмя звездочками **'****'**, расположенными в начале строки в распечатке вывода (output).

Как только ошибка выявлена, процесс будет остановлен до следующего случая обнаружения ошибки. Модель не будет решена до тех пор, пока ошибки выявляются. Существует только один способ решения модели - исправить ошибку и повторить расчет.

Ошибки сгруппированы в три стадии моделирования GAMS: компиляция, выполнение и воспроизводство модели (которое включает последующее ее решение). Следующие три подраздела рассказывают об этих трех типах ошибок.

10.6.1. ОШИБКИ КОМПИЛЯЦИИ (COMPILATION ERRORS)

Ошибки компиляции частично уже были обсуждены в разделах 2 и 5.6 , поэтому некоторые положения этого раздела совпадают с предыдущими.

Во время компиляции может быть выявлено до нескольких сотен различных ошибок, но часто может быть выявлен только один особый символ в вводном файле (input) GAMS. Причинами наибольшего числа ошибок являются простые ошибки: не продекларирован идентификатор, неправильно расставлены индексы, отсутствуют необходимые точки с запятой, орфографические ошибки. В диагнозе и исправлении более серьезных проблем помогают расширенные сообщения о причинах ошибок. Исправляйте ошибки последовательно по ходу текста программы и лучше по одной. Остальные возможно исчезнут сами.

Когда выявлена ошибка компиляции, то появляется знак **$** и номер ошибки. Эти данные располагаются ниже (и обычно справа) символа, который вызвал конфликт, на отдельной строке, начинающейся четырьмя звездочками (********).

Если на одной линии имеется более, чем одна ошибка (из-за того, что первая ошибка повлекла за собой серию других взаимосвязанных ошибок), то знак **$** может отсутствовать, а номера ошибок будут располагаться вплотную.

GAMS не может указать на одной строке более 10 ошибок.

К конце эхо-принт программы, приводятся номера всех встреченных ошибок вместе с описанием возможной причины каждой ошибки. Сообщения об ошибке являются самодастоточными , и здесь приводиться не будет.

Существует ценное замечание, что легко создать модель, которая не выполняет то, чтобы вы хотели от нее, но которая не содержит ошибок в известном смысле. Лучшая предосторожность - проверить работу внимательно и устанавливать как можно больше последовательных автоматических проверок.

Одна ошибка может быть причиной большой путаницы в модели - если зарезервированное GAMS слово используется в метке или в идентификаторе. В этом случае по техническим причинам невозможно выявить ошибку вспомогательными сообщениями.

В некоторых случаях ошибка не может быть выявлена до тех пор, пока не появится следующий за ней оператор. Тогда здесь появиться номер ошибки и комментарии к ней, однако, комментарии могут быть совершенно непонятными и глупыми. Необходимо внимательно проверять, что было причиной первой ошибки, из какой она группы, смотреть предыдущие операторы (особое внимание необходимо обратить на отсутствующие запятые и точку с запятой).

Следующий пример иллюстрирует общий формат сообщений об ошибках компилятора.

```
1   set c crops / wheat, corn, wheat, toollonganame/
    ****                            $172          $190
2
3   parameter price(c) /wheat 200, cotton 700/
    ****                                  $170


    ERROR MESSAGES

    109 ELEMENT TOO LONG - ONLY 10 CHARACTERS ARE ALLOWED
    170 DOMAIN VOILATION FOR ELEMENT - THIS LABLE DOES NOT BELONG TO
        THE SET USED AS DOMAIN IN THIS INDEX POSITION
    172 ELEMENT IS REDEFINED - THIS LABLE OR N-TUPLE HAS BEEN REPEATED

    ****   USER ERROR (S) ECOUNTERED
```

10.6.2. ОШИБКА ВРЕМЕНИ КОМПИЛЯЦИИ (COMPILATION TIME ERROR)

Отчетный формат для ошибок, найденных во время выполнения анализа оператора **solve** более сложный, чем для обычных ошибок компиляции, в основном, потому что должны быть проверены многие вещи. Все идентификаторы, на которые имеются ссылки, должны быть определены или присвоены, математические выражения в уравнениях должны соответствовать классу модели и так далее. Требуется более сложный отчет, чтобы точно описать любую разрабатываемую задачу. Когда установлено, что модель свободна от ошибок к моменту начала проверки оператора **solve**, тогда начинается проверяться только оператор **solve**. Такая последовательность выполнения проверок установлена не потому, что проверка сравнительно долговременна, а потому, что если проверяется оператор **solve** в программе, содержащей и другие ошибки, то может быть предъявлено много ошибочных и запутывающих сообщений.

Сообщения об ошибках выполнения **solve** появляется в двух местах и в двух форматах. Первое сообщение приводится немедленно после оператора **solve** с

кратким сообщением, включающим имя неправильного идентификатора и тип модели, где он используется. Это будет встречаться в большинстве случаев. Второе, более длинное сообщение с некоторыми предположениями о причине ошибок появляется вместе с остальными сообщениями об ошибках в конце компиляции.

Следующий пример иллюстрирует, основной отчетный формат для ошибок компилятора, связанных с оператором **solve**.

```
1     variables  x, y, z;  equations  eq1, eq2
2
3     eq1..  x**2 - y =e= z;
4
5     eq2.. min(x,y)  =l= 20;
6
7     model silly  /all/;   solves silly  using  lp maximizing z;
****                                        $54,51,256
****  THE FOLLOWING LP ERRORS WERE DETECTED IN, MODEL  SILLY
****  54 IN EQUATION EQ1         .. ENDOG OPERANDS FOR **
****  51 IN EQUATION EQ2         .. ENDOG ARGUMENTS (S) IN FUNCTION
8               SOLVE SILLY USING NLP MAXIMIZING Z
****                                         $257
ERROR MESSAGES
51    ENDOGONEOUS FUNCTIN ARGUMENT (S) NOT ALLOWED IN LINEAR MODEL
54    ENDOGENOUS OPERANDS FOR ** NOT ALLOWED IN LINEAR MODELS
256   ERROR(S) IN ANALIZING SOLVE STATAMENT .MORE DETAIL APPEARS
      BELOW THE SOLVE STATAMENT ABOVE
257   SOLVE STATAMENT NOT CHECKED BECAUSE OF PREVOUS ERRORS

****  USER ERRORS (S) ENCOUNTERED
```

10.6.3 ОШИБКИ ВЫПОЛНЕНИЯ (EXECUTION ERRORS)

Ошибки времени выполнения обычно вызваны недопустимыми арифметическими операциями, такими как деление на ноль или логарифмирование отрицательного числа и т.д. GAMS в выводном файле (файле output) создает сообщение об ошибке, в котором указан номер линии, где расположен ошибочный оператор, и продолжает выполнение. Программа GAMS не должна прерываться из-за неразборчивого сообщения от оперативной системы компьютера, еслли делается попытка выполнения недопустимой операции. GAMS имеет жестко определенную расширенную алгебру, которая содержит все операции, включая недопустимые операции. Модель [CRAZY] включает все нестандартные операции и может служить учебным пособием для их изучения. Помните, что арифметические дейсвия GAMS определяются для замкнутого интервала **[-inf, +inf]** и включают в себя значения **EPS** (близкие к нулю, но не ноль), **NA** (недействительные) и **UNDF** (результат недопустимой операции). Результаты недопустимых операций распространяются через всю систему и выводятся на печать с помощью стандартного оператора **display**. Однако необходимо помнить, что GAMS не может решать модель или сохранить рабочий файл, если предварительно была выявлена ошибка.

10.6.4. ОШИБКИ РЕШЕНИЯ (SOLVE ERRORS)

Выполнение оператора **solve** может выявить дополнительные ошибки, называемые **matrix** ошибки (ошибки матрицы), которые сообщают о проблемах, возникающих при трансформации модели в формат, требуемый для **solver**.

Наиболее частыми причинами проблем являются недопустимые или непоследовательные границы, или в случае если значение расширенного диапазона начинает использоваться как матричный коэффициент.

Ниже дан пример общего формата ошибок решения.

```
1    variable x;   equation eq1;
     2
     3    eq1..  x  =l= 10;
     4    x.lo  = 10;    x.up  =5;
     5
     6   model wrong  /eq1/ ;  solve wrong using lp maximizimg x;

----    X
X
            (.LO,  .L,  .UP  = 10,  10,  5)
     1       eq1
     0       (OBJECTIVE)

     ****  MATRIX ERROE - BOUNDS OR LEVEL IN VERIABLE BELOW ILLEGAL

     X       (.LO,  .L,  .UP  = 10,  10,  5)
```

Некоторые операторы **solve** требуют оценки нелинейной функции и расчета переменных. Т.к. эти расчеты не обеспечиваются GAMS, но они направляются на контроль другими подсистемами, и ошибки, ассоциируемые с этими расчетами, направляются в отчет расчета. Если не выполнено приведение в исходное состояние при помощи опции **domlit**, то при наличии в модели арифметичских противоречий, подсистемы будут прерывать процесс решения. Затем найденные противоречия будут отражены в распечатке, как показано ниже.

```
1    variable  x, y;  equation one;
2
3    one..  y =e= sqrt(10/x);  x.l = 10;  x.l0 = 0;
4
5    model divide   /all/ ;  solve divide maximizing y using nlp;

                    SOLVE          SUMMARY
              MODEL DIVIDE         OBJECTIVE Y
              TYPE NLP             DIRECTION MAXIMIZE
              SOLVER MINOSS      FROM LINE 5

     ****   SOLVER STATUS 5 EVALUATION ERROR LIMIT
     ****   MODEL  STATUS       7 INTERMEDIATE NONOPTIMAL
     ****   OBJECTIVE VALUE         1.7025

     RESOURSE USEGE, LIMIT         0.067          1000.000
     ITTERATION COUNT, LIMIT       1              1000
     EVALUATION ERRORS             1                 0

                  <other output>
     ****  ERROR(S)   IN EQUATION  ONE
           1 INSTANSE OF - UNDEFINED SQUARE ROOT (RETURNED .0)

                 <other output>

     **** REPORT SUMMARY:          1   NONOPT  ****
                                   0   INFEASIBLE ****
                                   0   UNBOUNDED  ****
                                   1   ERRORS  ****
```

Обратите внимание, что для состояния расчета (**solver status**) указано значение **5**, что означает, что **solver** был прерван потому, что оценочных ошибок было встречено больше, чем разрешено опцией **domlit**. Также приводится тип оценочных ошибок и уравнений, ставших причиной ошибок.

Если из-за оценочных ошибок **solver** возвращает промежуточное решение переменных, то попытка следующего **solve** все же будет выполнена. Только фатальная ошибка GAMS может быть причиной того, что программой **solver** возвращаются все решения (в том числе и плохие) данного шага расчета. Если это все таки случилось, вся возможная информация будет записана в файле output GAMS, но попытки выполнить следующий **solve** не будет.

10.7. ЗАКЛЮЧЕНИЕ

На этом заканчивается описание базисных возможностей GAMS. Следующие разделы предназначены для более углубленного изучения GAMS.

ДИНАМИЧЕСКИЕ МНОЖЕСТВА (DYNAMIC SETS) 11

1.1. ВВЕДЕНИЕ

Все **sets** (*множества*), которые обсуждались до сих пор, имели членство, декларированное через оператор **set**, и членство никогда не изменялось. В этом разделе мы будем обсуждать способы изменения членства в **sets** (*во множествах*). **Set** (*множество*), чье членство может быть изменено, называется **dynamic set** (динамическим множеством) в отличии от **static set** (*статического множества*), в котором членство постоянно. Отличия между динамическим и постоянным множествами очень важные. Эта тема не рассматривалась до настоящего момента, потому, что она очень сложная для пользователя, только начинающего программировать на языке GAMS. Для подготовленного пользователя умение использовать в моделях **dynamic sets** (*динамические множества*) будет очень полезно.

11.2. ПРИСВОЕНИЕ ЧЛЕНСТВА DYNAMIC SETS (ДИНАМИЧЕСКИМ МНОЖЕСТВАМ)

Sets (*множества*) могут присваиваться таким же способом, как это выполняется для других типов данных. Отличие заключается в том, что арифметические операции не могут быть выполнены на множествах таким же образом, как они могут быть выполнены на идентификаторах "типизированных значений " (**parameters** (*параметрах*) или подтипах **variables** (*переменных*) и **equations** (*уравнений*)). **Dynamic sets** (*динамические множества*) наиболее часто используются в качестве "контролирующих индексов" в присвоениях или в определениях уравнений, или в качестве контролирующего объекта в индексной операции доллрарового контроля.

11.2.1. СИНТАКСИС

В основном в GAMS существует следующий синтаксис присвоения членства динамическим множествам

set_name(domain_name) | domain_label) = yes | no ;

set_name - внутренне имя множества (также называемое идентификатором) в GAMS. **YES** и **NO** - ключевые слова, используемые в GAMS, чтобы соответственно обозначить членство или его отсутствие в присваеваемом множестве.

Самый важный принцип следующий : во время декларации **dynamic set** (*динамическое множество*) всегда должно контролироваться доменом, чтобы быть подмножеством **static set(s)** (*статического множества или множеств*).

Конечно, можно использовать динамические множества, которые не контролируются доменом, это обеспечивает дополнительную мощность и гибкость, в тоже время это делает модель непонятной, а вероятность ошибок более высокой. Любая однажды установленная метка является допустимой и

сохраняется также долго, как используемый и установленный **dynamic sets** (*динамическое множество*).

11.2.2. ИЛЛЮСТРИРУЮЩИЙ ПРИМЕР

Следующий пример, взятый из [ZLOOF], используется для иллюстрации присвоения членства **dynamic set** (*динамическому множеству*).

```
set  item     all items             /dish,ink,lipstick,pen,pencil,perfum /
subitm1(item) first  subset of item              /pen, pencil/
subitm2(item) second subset of item;

subitem1('ink') = yes;     subitem1('lipstick') = yes;
subitem2(item)  = yes;     subitem2('perfume' ) = no ;

display subitem1, subitem2;
```

Обратите внимание, что **sets** (*множества*) **subitem1** и **subitem2** декларируются подобно любому другому **set** (*множеству*). Два множества стали динамическими из-за присвоения. Они также контролируются доменом: только, члены, которые они когда-либо смогут иметь, должны быть также членами **item**. Но **item** является статическим множеством и с этого времени их членство заморожено. Каждое из двух первых присвоений прибавляет один новый элемент к подмножеству subitem1 (сначала элемент **ink**, затем элемент **lipstick**). Третье присвоение - это хорошо известное индексное присвоение: подмножеству **subitem2** присваиваются все члены **item**. Четвертое присвоение удаляет один элемент (**perfume**) из подмножества **subitem2**. Вывод, выполненный посредством оператора **display**, покажет все элементы подмножеств. Вывод демонстрируется ниже для проверки.

```
----  7 SET      SUBITEM1   first subset of item
 INK,     LIPSTICK,  PEN,   PENCIL

----  7 SET      SUBITEM    second subset of item
 DISH ,   INK,     LIPSTICK,  PEN,   PENCIL
```

Элементы отображаются в порядке, установленном в декларации **item**.

11.2.3. DYNAMIC SETS (ДИНАМИЧЕСКИЕ МНОЖЕСТВА) С МНОЖЕСТВЕННЫМИ ИНДЕКСАМИ

Dynamic set (*динамические множества*), подобно **static set** (*статическим множествам*), не могут быть более, чем 10 мерные (т.е. не могут содержать более 10 элементов). Следующий пример иллюстрирует присвоение многомерным **sets** (*множествам*).

```
sets    item             items         /pencil, pen/
        sup              suppliers     /bic, parker, waterman /
        supply(item,sup);

supply('pencil','bic')  = yes;
supply('pen'   ,'sup')  = yes;
```

Все правила использования кавычек и круглых скобок, которые мы дали для статических множеств (раздел 4) применимы для динамических множеств.

11.2.4. ПРИСВОЕНИЕ DYNAMIC SETS (ДИНАМИЧЕСКИХ МНОЖЕСТВ) ЧЕРЕЗ ДОМЕН

Так как **dynamic sets** (*динамические множества*) являются подмножествами **static sets** (*статических подмножеств*), то можно выполнять присвоение через домен **dynamic sets** (*динамических множеств*). Это не является тем же самым как выполнение проверки домена, используя **dynamic set** (*динамическое множество*).

Следующий пример, взятый из раздела 11.2.2 иллюстрирует использование динамических множеств в качестве доменов

```
subitem1(item)     =  no
subitem1(subutem2) = yes;
```

Первое присвоение указывает, что **subitem1** пустой. Обратите внимание, что это также может быть сделано с **parameters** (*параметрами*). Например,

```
parameter   inventory(item);
inventory(subitem1) = 25;
```

11.2.5. УРАВНЕНИЯ, ОПРЕДЕЛЕННЫЕ ЧЕРЕЗ ДОМЕН DYNAMIC SETS (ДИНАМИЧЕСКИХ МНОЖЕСТВ)

Иногда необходимо определить **equation** (*уравнение*) через **dynamic set**(*динамическое множество*).

Хитрость заключается в том, чтобы декларировать уравнения через полный домен, а определить его через **dynamic set** (*динамическое множество*).

Следующий пример иллюстрирует, как это достигается.

```
set    allr    all  regions       /n, s, w, e, n-e, s-w /
     r(allr)   region  subset  from  particular solution;

scalar  price  /10/;

equations   prodbal(allr)  production balance ;

variable    activity(allr)  first activity
            revenue(allr)   revenue

prodbal(r).. activity(r) *price =e= revenue(r);
```

Повторяем важное положение: уравнение декларируется через **allr**, но указывается через **r**. Затем в пределах членства **allr** может быть выполнено условная декларация для **r**.

11.3. ИСПОЛЬЗОВАНИЕ ДОЛЛАРОВОГО КОНТРОЛЯ С DYNAMIC SETS (ДИНАМИЧЕСКИМИ МНОЖЕСТВАМИ)

Этот раздел требует понимания "долларового условия". Все механизмы долларового контроля, приведенные в разделе 15, действительны для

использования с **dynamic sets** (*динамическими множествами*). Действительно, полная мощь **dynamic set** (*динамического множества*) может проявиться при использовании здесь долларовых контролей.

Обратите внимание, что **dynamic set** (*динамическое множество*) имеет только значения **yes** и **no** , и поэтому может быть использовано в качестве основы для логического оператора. Поэтому в динамических множествах внутри долларового оператора могут быть выполнены только следующие операции: **not, and, or** или **xor**, и также ряд операций, описанных в разделе 11.4.

В основном **dynamic sets** (*динамические множества*) внутри долларовых условий используются в присвоениях, индексных операциях и в уравнениях. Каждое из них будет детально описано в следующих подразделах.

11.3.1. ПРИСВОЕНИЯ

Dynamic sets (*динамические множества*) могут быть использованы внутри долларового условия в пределах присвоений, определяющих другие **dynamic sets** (*динамические множества*) или **parameters** (*параметры*).

Как иллюстрация использования **dynamic set** (*динамического множества*) внутри долларового контроля для определения другого динамического множества, два оператора из примера, приведенного в разделе 11.2.4, могут быть записаны с эквивалентным результатом в виде

```
subitem1(item)  = yes$subitem2(item);
```

который является выразительной формой следующего оператора

```
subitem1(item) = yes$subitem2(item) + no$(not subitem2(item));
```

В присвоении **set** (*множества*), которое получается, в логическом варианте ответа - "**else**" при использовании оператора "**доллар справа**", является **NO**. И это удобнее, чем ноль, который в аналогичной ситуации возникает при работе с обычными данными.

Второй пример из раздела 11.2.4 может быть переписан, чтобы проиллюстрировать использование динамического множества в определении **parameters** (*параметров*)

```
inventory(item)$subitem1(item) = 25;
```

11.3.2. ИНДЕКСНЫЕ ОПЕРАЦИИ

Другое важное использование долларового контроля с **dynamic sets** (*с динамическими множествами*) заключается в контроле домена во время выполнения индексных операций, подобных **sum** и **prod**.

Рассмотрим трансформированный пример из раздела 11.3.1

```
parameter totinv  total inventory;
totinv = sum(item$subitem1(item), inventory(item));
```

Этот пример представлен только для иллюстрации. Обратите внимание, что вышеприведенный второй оператор может быть переписан более выразительно как

```
totinv = sum(subitem1, inventory(subitem1(item);
```

Это не всегда возможно. Рассмотрим следующий абстрактный пример

```
Sets  item  items  sold    /pencil, pen/
      sup   suppliers      /bic, prker, waterman/
      dep   department     /stationery, household/
      supply(item, sup);

supply('pencil', 'bic') = yes; supply('pen', 'sup') = no;

parameter totsales(dep);
totsales(dep) = sum(item$supply(item,'bic'), sales(dep,item));
```

Вышеприведенное присвоение используется, чтобы найти полную продажу (**totalsales**) всех департаментов, которые выполняют продажу отдельных предметов (**items**), поставляемых (**supplied**) **bic**.

Обратите внимание, что динамическое множество используется, чтобы ограничить домен суммирования, к домену, для которого **supply(item,'bic')** является истина.

11.3.3. УРАВНЕНИЯ

Dynamic sets (*динамические множества*) могут использоваться внутри долларовых условий как часть алгебраического уравнения или во время определения домена уравнения. В первом случае это похоже на случай присвоения, описанный в разделе 11.3.1. Следующий случай используется, чтобы ограничить уравнение через домен **dynamic set** (*динамического множества*). Уравнение, определенное в примере из раздела 11.2.5, может быть переписано с эквивалентным результатом следующим образом

```
prdbal(allr)$r)allr)..  activity(allr)*price  =e= revenue(allr);
```

Домен определения уравнения **prodbal** ограничивается к элементами, которые принадлежат только **dynamic set** (*динамическому множеству*) **r**.

11.4. ОПЕРАЦИИ НАД МНОЖЕСТВАМИ

Этот раздел описывает, каким образом используя **dynamic sets** (*динамические множества*) могут быть выполнены различные символьные операции над множествами. В последующих подразделах последовательно описываются операции над множествами: объединение, пересечение, дополнение, разность.

11.4.1. ОБЪЕДИНЕНИЕ МНОЖЕСТВА

Символ **+** выполняет операцию объединения множества. Рассмотрим следующий пример

```
subitem3(item) = subitem1(item) + subitem2(item);
```

Членство подмножества **subitem3(item)** - это множество всех элементов подмножества **subitem1(item)** и всех элементов подмножества **subitem2(item).** Вышеприведенная операция эквивалентна следующему более длинному способу

```
subitem3(item)=no; subitem3(subitem2)=yes;  subitem3(subitem1)=yes;
```

11.4.2 ПЕРЕСЕЧЕНИЕ МНОЖЕСТВА

Символ * выполняет операцию пересечения[1] множества. Рассмотрим следующий пример

```
Subitem3(item)  = subitem1(item) * subitem2(item);
```

Членство подмножества **subitem3(item)** является множество, состоящее из элементов, входящих как в подмножество **subitem1(item)** так и в подмножество **subitem2(item).**

Вышеприведенная операция эквивалентна следующему более длинному способу

```
subitem3(item) = yes$(subitem1(item) and subitem2(item));
```

11.4.3 ДОПОЛНИТЕЛЬНОЕ МНОЖЕСТВО

Оператор **not** выполняет операцию дополнения множества. Рассмотрим следующий пример

```
subitem3(item) = not subitem1(item);
```

Членство **subitem3(item)** является множество, состоящее из элементов, входящих в множество **item**, но не содержащихся в подмножестве **subitem1(item).**

Вышеприведенная операция эквивалентна следующему более длинному способу

```
subitem3(item) = yes; subitem3(subitem1) = no;
```

11.4.4. РАЗНОСТЬ МНОЖЕСТВ

Оператор «-« выполняет операцию разность[2] множества. Рассмотрим следующий пример

```
Subitem3(item) = subitem1(item) - subitem2(item);
```

Членство в **subitem3(item)** является множество, состоящее из элементов, которые являются членами подмножества **subitem1(item),** но не членами подмножества **subitem2(item).**

Вышеприведенная операция эквивалентна следующему более длинному способу

```
subitem3(item) = yes$(subitem1(item); Subitem3(subitem2) = no;
```

[1]Пересечение множества - операция над множествами: пресечению множества A и B принадлежат те и только те элементы, которые входят и в A, и в B. входящих и в A, и в B.

[2]Разность множеств, дополнение - множество, являющееся разность множеств A и B, состоит из элементов, принадлежащих A и не принадлежащих B)

11.5. ЗАКЛЮЧЕНИЕ

Задача операций над множествами заключается в том, чтобы выполнить расчеты, базирующиеся на заданных данных **static sets** (*статических множествах*) для использования в обработки особых ситуаций. Это дополнительный пример принципа ввода небольшого количества данных и составления модели из самой изначальной информации.

МНОЖЕСТВА КАК ПОСЛЕДОВАТЕЛЬНОСТЬ: УПОРЯДОЧЕННЫЕ МНОЖЕСТВА 12

12.1. ВВЕДЕНИЕ

При обсуждении **sets** (*множеств*) разделе 4 говорилось, что если нет особой необходимости выделять элементы множества, тогда одномерные множества можно рассматривать как неупорядоченный набор меток. В этом разделе будут рассмотрены особые случаи, когда необходимо иметь дело с упорядоченными множествами.

Упорядоченные множества используются в экономических моделях, в которых в каждый период времени имеются различные условия, и необходимо выполнять ссылки на "следующий" или "предыдущий" период времени, чтобы обеспечить соответствие между периодами. Другой пример использования упорядоченных множеств - модели, описывающие запас капитала, которые включают в себя уравнения, когда запас на конец периода **n** равен запасу на конец периода **n-1** плюс нетто прибыли в течении периода **n**. Локальные задачи типа "Сеть низкого напряжения в городе" или "Задачи планирования", где требуется непрерывная область и в которых множества должны иметь свойства последовательности - это задачи другого класса.

Модели, включающие в себя последовательности периода времени, часто называются динамическими моделями, потому что они описывают изменение условий во времени. Здесь неудачно использовано слово "динамическое". Слово "динамическое" в данном контексте имеет отличное значение от того, как оно было использовано в отношении множеств в разделе 11. Такие неувязки неизбежны.

12.2. УПОРЯДОЧЕННЫЕ И НЕУПОРЯДОЧЕННЫЕ МНОЖЕСТВА

Когда множество должно быть представлено как последовательность, накладываются ограничения, как это было использовано с множествами в контроле домена. Понятие **static set** (*статическое множество*) было введено в разделе 11: **static set** (*статическое множество*) должно быть инициализировано вместе со списком меток, заключенных между наклонными чертами **"/"** во время, когда множество декларируется и никогда не изменяется в последствии.

Упорядоченные множества должны быть **static sets** (*статическими множествами*). Иными словами неупорядоченность возможна в **dynamic sets** (*динамических множеств*).

GAMS поддерживает только один список "исходных" элементов - меток, которые используются как элементы в одном или более **sets** (*множествах*). Упорядоченность элементов в каком либо одном последующем **set** (*множестве*) является такой же, как упорядоченность этих элементов в этом "исходном" списке элементов. Это означает, что упорядоченности множества может не быть в последующих декларациях **set** (*множеств*), и это всегда так, если какие-то метки были использованы в ранних определениях **set** (*множеств*).

В GAMS карту меток можно увидеть, если где-то перед первой декларацией множества установить указание транслятору **$onuellist.**

Есть хорошее правило "большого пальца", говорящее, что если метки од-

ного **set** (*множества*) хотят быть упорядочены, но еще не были использованы, то они будут упорядочены.

Карта меток демонстрируется вместе с другими картами компилятора после распечатки программы. В нижеприведенном примере мы видим упорядоченные и неупорядоченные множества и карту, демонстрирующую последовательность. Ввод (input) следующий

```
$onuellist

    set  t1  /1987, 1988, 1989, 1990, 1991/
         t2  /1983, 1984, 1985, 1986, 1987/
         t3  /1987, 1989, 1991, 1983, 1985/ ;
```

Нижеприведенная карта вывода (см.ниже) показывает введенную упорядоченность (это важно) и отсортированную упорядоченность, получаемую путем сортировки меток в словарную последовательность. Имеется ввиду что сортировке в общем случае подвергаются не только цифры но и буквы в порядке их появления в английском алфавите.

Одиночные цифры слева карты вывода - это порядковый номер первой метки на линии, где записана метка.

```
General Algebraic Modeling System
Unique Element Listing     список "исходных" элементов

Unique Elements in Entry Order  "исходных" элементы в введенной упорядоченности

1  1987   1988   1989   1990   1991   1983
7  1984   1985   1986

Unique Elements in Sorted Order  "исходных" элементы в отсортированной упорядоченности

1   1983   1984   1985   1986   1987   1988
7   1989   1990   1991
```

Всегда можно сделать множество упорядоченным, для чего нужно декларацию множества ввести в начале программы. При таких действиях и в середине программы мы будем иметь операции, которые имеют дело с множествами как упорядоченными последовательностями.

12.3. ОПЕРАТОРЫ ORD И CARD

В разделе 4 было объяснено, что метки не имеют численного значения. В используемых примерах говорилось, что метка **'1986'** не имеет численного значения 1986, а метка **'01'** отличается от метки **'1'**. Этот раздел представляет два оператора **ord** и **card**, которые возвращают целочисленные значения, когда обращаются к множествам. Несмотря на то, что возвращенные целочисленные значения не представляют собой численные значения меток, они необходимы для описания определенных ситуаций в моделировании.

Следующие два подраздела описывают каждую из этих двух операторов.

12.3.1. ОПЕРАТОР ORD

ORD возвращает соответствующую позицию члена во множестве.

ORD может быть использован только с одномерными, статическими, упорядо-

ченными множествами.

Некоторые примеры показывают, как используется оператор **ORD**.

```
set  t  time  period  /1985*1995/
parameter  val(t);
val(t) = ord(t);
```

Результатом вышеприведенного оператора будет:
- **val('1985')** будет иметь значение **1**;
- **val('1986')** будет иметь значение **2** и так далее.

Обычное использование оператора **ORD** заключается в установлении векторов, которые представляют аналитически заданные возрастающие количества. Например, предположим, что государство имело 56 миллионов населения в базовый период, и население возрастает с темпом 1.5 % каждый год. Тогда население в последующие годы может быть рассчитано следующим образом

```
population(t) = 56 * (1.015**(ord(t) - 1));
```

В GAMS часто успешно моделируются общие матричные операции. Первый индекс в двухмерных **parameters** (*параметрах*) может условно быть представлен в строке, а второй - в колонках, при этом необходимо соблюдать упорядоченность элементов. Нижеприведенный пример показывает, каким образом оформить запись, чтобы верхний треугольник матрицы был равен индексу строки плюс индекс колонки, а диагональ и нижний треугольник были равны нулю.

```
set  i  row  and  column  labels  /x1*x10/; alias(i,j);
parameter  a(i,j)  a general square matrix;
a(i,j)))$(ord(i) lt ord(j))  =  ord(i) + ord(j);
```

12.3.2. ОПЕРАТОР CARD

CARD возвращает число элементов во множестве. **CARD** может быть использован с любым **set** (*множеством*), даже с динамическим и неупорядоченным.

Следующий пример иллюстрирует это.

```
set  t  time  period  /1985*1995/
parameter  s ;  s = card(t);
```

В результате вышеприведенного оператора **s** будет присвоено значение **11.** Обычно оператор **card** используется только в заключительный период для некоторых особых условий, например, чтобы зафиксировать переменную. Рассмотрим абстрактный пример:

```
c.fx(t)$(ord(t) = card(t)) = demand(t);
```

который фиксирует переменную только для последнего члена, но не выполняет присвоения для других членов **t**. Преимущество этого способа фиксирования **c** заключается в том, что членство **t** может быть безопасно изменено, а оператор **card** всегда будет фиксировать **c** для последнего члена.

12.4. ОПЕРАТОРЫ LAG И LEAD

Операторы **LAG** и **LEAD** используются, чтобы отнести "текущий" член множества к "следующему" или "предыдущему" члену множества. Эти операторы применимы только к упорядоченным множествам. GAMS предусматривает две формы

операторов **LAG** и **LEAD**:

- линейные операторы **LAG** и **LEAD (+, -)**
- круговые операторы **LAG** и **LEAD (++, --)**.

Различия между этими двумя типами операторов включают в себя управление конечными точками в последовательностях. GAMS содержит некоторые встроенные средства, чтобы оперировать с такими начальными и конечными элементами **set** (*множеств*), но в любой работе, включающей последовательности, пользователь должен внимательно думать об обработке конечных точек, и все модели будут нуждаться в особой управляющей логике исключения, чтобы оперировать с ними.

В линейном случае члены множества, которые являются конечными точками, - представляют собой левое "подвешивание". Другими словами, это не члены, предшествующие первому члену или следующие за последним. Это может быть причиной использования не существующих элементов. В следующем разделе будет показано, как это управляется в GAMS. Операторы **LAG** и **LEAD** успешно используются при моделировании периода времени, который не повторяется. Примером является множество лет (с 1990 по 1997). Операторами являются **+** и **-**.

GAMS в состоянии различить линейные операторы (+, -) от арифметических операторов по контексту.

В круговом случае принимается, что первые и последние члены множества смежные. Существует понятие, что "**first-1**" является указателем к "**last**", а "**last+2**" - указатель к "**first+1**" и так далее. Все указатели и присвоения определяются. Это успешно используется при моделирования периода времени, которое повторяется (например, месяцы в году или часы в сутках). Естественно думать, что январь - это месяц, следующий за декабрем. Модели бюджета сельскохозяйственных ферм являются примером применения кругового оператора **LEAD.** Операторами являются ++ и --.

В следующих двух разделах будет рассмотрено использование операторов **LAG** и **LEAD** в операторах присвоения и в уравнениях.

12.5 ОПЕРАТОРЫ LAG и LEAD В ПРИСВОЕНИЯХ

Операторы **LAG** и **LEAD** используются в операторах присвоения один раз. Использование операторов **LAG** и **LEAD** с правой стороны присвоения называется **указателем**, а использование этих операторов с левой стороны называется **присвоением** и включает в себя определение домена присвоения. Понятия после **указателя** и **присвоения** являются в раной мере действительны для линейной и циркульной форм операторов **LAG** и **LEAD**. Однако для циркульных операторов **LAG** и **LEAD** важного различия между **указателем** и **присвоением** не существует, потому что недействительные элементы в этом случае не используются.

Указатель к не существующим **parameters** (*параметрам*) приводит к использованию значения по умолчанию (в данном случае к нулю), тогда как попытка выполнить **присвоение** к не существующему элементу приводит к невыполнимым **присвоениям**.

12.5.1. ЛИНЕЙНЫЕ ОПЕРАТОРЫ LAG и LEAD - УКАЗАТЕЛИ.

Рассмотрим следующий пример, где два параметра иллюстрируют применение линейных операторов **LAG** и **LEAD** в качестве **указателя**

```
set  t  time  sequence   /y-1987*y-1991/;
parameter a(t), b(t);
a(t)  =  1986 + ord(t);
b(t)  =  -1; b(t) = a(t-1);
option decimals = 0; display a, b;
```

Оператор **optoin** пресекает десятичные знаки в значениях при выводе на печать, когда используется оператор **display** (см. раздел 13).

Результаты работы оператора показаны ниже

```
    ----   6 PARAMETERS A
Y-19U87 1987, Y-1988 1988, Y-1989 1989, Y-1990 1990, Y-1991 1991

    ----   6 PARAMETERS B
Y-1988 1987, Y-1989 1988, Y-1990 1989, Y-1991 1990
```

Для **a**, как и ожидалось, значения **1987, 1988** и до **1991** соответствуют меткам **Y-1987**, **Y-1988** и так далее. **b** инициализируется к **-1**.

Для **b** выполняется присвоение через все элементы множества **t** следующим образом: значение **a** предыдущего периода присваивается к текущему элементу **b**. Если значения предыдущего периода не существует (как, например, для первого элемента **a**, равного **Y-1987**), то используется ноль, и **b('Y-1987')** становится нулем, заменяя предыдущее значение **-1**.

12.5.2. ЛИНЕЙНЫЕ ОПЕРАТОРЫ LAG и LEAD - ПРИСВОЕНИЕ

Рассмотрим следующий пример, где два параметра **a** и **c** используются для иллюстрации присвоения линейных операторов **LAG** и **LEAD**.

```
set  t  time  sequence  /Y-1987*Y-1991/;
parameter a(t), c(t) ;
a(t)  = 1986 + ord(t);
c(t)  = -1; c(t+2) = a(t);
option decimals=0;  display a,c;
```

Результат показан ниже

```
    ----  6  PARAMETER  A
Y-1987 1987, Y-1988 1988, Y-1989 1989, Y-1990 1990, Y-1991 1991

    ----  6 PARAMETER C
Y-1987  -1, Y-1988  -1, Y-1989  1987, Y-1990  1988, Y-1991  1989
```

Присвоение к **a** объяснено в разделе 12.5.1. **Присвоение** к **c** отличается от **присвоения** к **a**. Это лучше выразить словами "В последовательности для каждого члена **T** найди член **C**, соответствующий **T+2**. Если такой член существует, замени его значение на значение **a(t**). Если такого члена не существует (как с величиной как для **Y-1990** и **Y-1991**), не выполняй **присвоения**". Первое число T есть Y+1987, поэтому первое **присвоение** сделано к **c('Y-1989')**, которое берет значение **a('Y-1987') - 1987**. Для **c('Y-1987')** и **c('Y-1988')** никакого **присвоения** не выполнялось: эти два сохраняют их предыдущие значения **-1**.

Значение **LAG** (или **LEAD**) не должно представлять собой определенную константу, оно может представлять собой произвольное выражение, при условии, что оно приводится к целочисленному. Если это условие не выполняется, то будет дано сообщение об ошибке. Отрицательный результат приведет к отключению считывания (восприятия) (например из **LEG** и **LEAD**).

В следующем примере все **set** (*множество*) **d(t)** будет нулевым.

```
d(t) = d(t - ord(t));
```

12.5.3. КРУГОВЫЕ ОПЕРАТОРЫ LAG и LEAD

Следующий пример иллюстрирует использование круговых операторов **LAG** и **LEAD.**

```
   set          s  seasons   /spring, summer, autumn, wnter/;
parameters  val(s)   /spring  10, summer  15, autumn  12, wnter  8/

      lagval2(s)
      leadval(s) ;
Lagval2(s) = -1; lagval2(s)  = val(s--2);
leadval(s) = -1; leadval(s++1)  = val(s);
option decimals = 0;  display val, lagval2, leadval;
```

В результате получаем

```
----    7 PARAMETER  VAL
SPRING  10, SUMMER  15, AUTUMN  12,  WINTER  8

----    7 PARAMETER  LAGVAL2
SPRING  12, SUMMER  8,  AUTUMN  15,  WINTER  12

----    7 PARAMETER  LEADAVL
SPRING  8,  SUMMER  10,  AUTUMN  15,  WINTER  12
```

Parameter (*параметр*) **lafval2** используется в качестве **указателя**, в то время как **lagval1** используется в качестве **присвоения**. Обратите внимание, что круговые операторы **LAG** и **LEAD** неприменимы к не существующим элементам. Поэтому различия между **указателем** и **присвоением** не важны.

Учтите, что следующие два оператора из вышеприведенного примера

```
lagval2(s)  =  val(s--2);
leadval(s)  =  val(s);
```

является эквивалентом

```
lagval2(s++2)  = val(s);
leadval(s) = val(s--1);
```

Различий в использовании **указателя** и **присвоения** для круговых операторов **LAG** и **LEAD** не существует, т.к. не существует различий в эффекте использования этих операторов.

12.6. LAGS и LEADS В УРАВНЕНИЯХ

Принципы, установленные в предыдущем разделе, распространяются на определения **equations** (*уравнений*). Операторы **LAG** и **LEAD** в теле уравнений (справа от символа "..") являются **указателем**, и, если ассоциированная индекс не определен, терм стремится к нулю (исчезает). Операторы **LAG** или **LEAD** слева от символа ".." является **модификацией к домену определения equation** (*уравнения*). Линейная форма может быть причиной, что одно или более

частных уравнений будут игнорированы в действиях SOLVERa.

Все операнды **LAG** и **LEAD** должны быть экзогенными.

В следующих двух разделах приведены примеры, иллюстрирующие использование линейных форм операторов **LAG** и **LEAD** в уравнениях в качестве **указателя** и для того, чтобы **модифицировать домен определения уравнений**. В разделе 12.6.3 будет проиллюстрировано использование круговых форм операторов **LAG** и **LEAD** в **equetions** (*уравнениях*).

12.6.1. ЛИНЕЙНЫЕ ОПЕРАТОРЫ LAG и LEAD - КОНТРОЛЬ ДОМЕНА

Рассмотрим следующий пример, взятый из [RAMSEY]

```
set  t  time  periods  /1990*2000/
tfirst(t)  first period
tlast(t)   last period;

tfirst(t)  = yes$(ord(t)  eq1);
tlast(t)   = yes$(ord(t)  eq (card(t));
display  tfirst,  tlast;

variables  k(t)  capital stock (trillion rupess)
           i(t)  investment (trillion rupes per year);

equations  kk(t)  capital balans (trillion rupes)
           tc(t)  terminal condition (for post-term growth);

kk(t+1)..      k(t+1) =e= k(t) +i(t);
tc(tlast)..    g*k(tlast)  =l= i(tlast);
```

Декларация **t** включается как пара **dynamic sets** (*динамических множеств*), которые используются, чтобы управлять первым и последним периодами (конечными условиями) в чистом виде.

Интересующее нас уравнение - **kk** (баланс капитала). **Set** (*множество*) **t** содержит в себе члены от 1990 до 2000, и таким образом они будут ограничением запаса капитала с 1990 по 2000. Запись ограничения для 1991 следующая

```
k('1991')  =e=  k('1991') + i('1990');
```

Оператор **LEAD** в домене определения ограничил число воспроизводимых ограничений таким образом, что они не указываются к не существующим переменным: генерируемая задача будет иметь 10 ограничений **kk**, определяющих связи между 11 значениями капитала **k**.

Обратите внимание, что ограничение **tc** полностью определяется только для конечного периода из-за присвоения к **set** (*множеству*) **tlast**. Обратите внимание, что использование **dynamic sets** (*динамических множеств*) приводит к контролю домена двух уравнений. В другой части модели **set** (*множество*) **tfirst** также используется, чтобы была возможность впоследствии установить начальные условия, особенно запас капитала в первый период **k('1990').**

12.6.2. ЛИНЕЙНЫЕ ОПЕРАТОРЫ LAG И LEAD - УКАЗАТЕЛИ

В примере из раздела 12.6.1 уравнение **kk** может быть переписано с

эквивалентным эффектом

```
kk(t)$(not  tfirst(t)).. k(t)  +  i(t);
```

Долларовое условие будет причиной того, что одно частное уравнение будет игнорироваться в расчетах.

Однако обратите внимание, что использование операторов **LAG** и **LEAD** в домене уравнения всегда будет приводить к тому, что одно или более частных уравнений будут пресечены, что не всегда желательно. Для одного из предыдущих примеров рассмотрим следующий модифированный ряд ограничений. В данном примере использовались операторы **LAG** и **LEAD** для **контроля домена** определения уравнения

```
kk(t+1).. k(t+1) =e= k(t) + i(t);
kfirst(tfirst).. k(first) =e= k0;
```

Здесь, установление границы в начале множества предпочтительнее, чем в конце. Это может быть выражено более компактно в следующем виде

```
kk(t).. k(t-1)  +  k$tfirst(t)  +  i(t+1);
```

В общем, выбор между использованием операторов **LAG** и **LEAD** в качестве **указателя** или **в контроле домена** - дело вкуса.

12.6.3. КРУГОВЫЕ ОПЕРАТОРЫ LAG И LEAD

В случае круговых операторов **LAG** и **LEAD**, различия между их использованием в области контроля или в качестве указателя не важны, потому что они не работает в пресеченных уравнениях или термах.

Рассмотрим следующий пример.

```
set   s  season  /spring, summer, autumn, winter/;

variable  prod(s)    amount of goods produsedin each season
          avail(s)   amount of goods available in each season
          sold(s)    amount of goods sold in each season  ;

equation  matbal(s);

matbal(s)..  avail(s++1) =e=  prod(s)  +  sold(s);
```

В этом примере возникают четыре отдельных примера. Они приведены ниже.

```
avail(summer)  =e=  prodn(spring) + sold(spring);
avail(autumn)  =e=  prodn(summer) + sold(summer);
avail(winter)  =e=  prodn(autumn) + sold(autumn);
avail(spring)  =e=  prodn(winter) + sold(winter);
```

Обратите внимание, что ни одно из уравнений не пресекаются.

12.7. ЗАКЛЮЧЕНИЕ

В этом разделе было дано понятие об упорядоченных множествах. В разделе были также детально описаны все возможности GAMS, когда он имеет дело с функциями **ord** и **card**, а также с линейными и круговыми формами операторов **lag** и **lead**.

ОПЕРАТОР DISPLAY 13

13.1. ВВЕДЕНИЕ

В этом разделе будет детально рассмотрен оператор **display**, включая различные способы контроля, доступные пользователю путем компоновки и изменения внешнего вида вывода (output). Контроль с помощью оператора **display** - это в некотором роде компромисс, чтобы обеспечить гибкость модели. Выполнение оператора **display** позволяет данным быть записанными в распечатку файла вывода.

13.2. СИНТАКСИС

В общем случае синтаксис оператора **display** следующий

display indent-ref|qouted text {, indent-ref|qouted text}

Indent-ref обозначает имя **set** (*множества*) или **parameter** (*параметра*), или **equation** (*уравнения*) или **variable** (*переменного*), назначенных к выводу без списков доменов или управляющих индексов **set** (*множества*).

Идентификатор указывает, какой объект выводится на печать. Текст заключается в кавычки, он может быть смешанным и подобранным в любом порядке. Оператор может занимать несколько строк. Вывод, получаемый при помощи оператора **display**, включает в себя метки и данных. Для **sets** (*множеств*) символ **yes** (обозначающий существование) используется вместо значения.

Для всех типов данных отображаются только значения не по умолчанию.

Значения по умолчанию, в основном, - это нулевые значения (кроме **.lo** и **.up** подтипов **variable** (*переменных*) и **equation** (*уравнений*)). Для них значения по умолчанию показаны ниже.

Переменная	**Уравнение**	**.lo**	**.up**
Positive	=g=	0	+inf
Free	=n=	-inf	+inf
Negative	=l=	-inf	0
Integer		0	100
Binary		0	1
	=e=	0	0

13.3. ПРИМЕР

Пример оператора **display** приведен ниже

```
set  s  /s1*s4/,  t /t5*t7/;
parameter  p(s)  /s1  0.33,  s3  0.67/;
parameter  q(t)  /t5  0.33,  t7  0.67/;
variable v(s,t) ;  v.l(s,t)  =  p(s)*q(t);
display 'first a set', s, 'then parameter', p,
        'then the activity level of a variable', v.l;
```

Результирующая распечатка файла будет состоять из следующих разделов, которые соответствуют оператору **display**

```
----  5 first a set
----  5 SET      S

S1,   S2,   S3,   S4

----   5 then a parameter
----   5 PARAMETER   P

S1  0.330     S3  0.670

----  then the activity level of a variable
----  5 VARIABLE  V.L

        T5      T7

S1    0.109    0.221
S3    0.221    0.449
```

Обратите внимание, что выводятся на печать только ненулевые значения. В случае многомерных идентификаторов для более легкого восприятия данные представляются в табличном виде.

13.4. ПОРЯДОК МЕТОК В DISPLAYS

Компоновка отображения по умолчанию идентификаторов различной размерности в сжатом виде показана в таблице. Цифры в таблице указывают на позицию индекса в списке домена идентификатора. Например, если мы выводим на печать **C**, где **C** декларируется как **C(i,j,k,l)**, тогда метки **i** (первый индекс) будет ассоциироваться с планшетным способом вывода или с отдельными подтаблицами, **j** и **k** с метками строк, а **l** (четвертый и последний индекс) с заглавием колонки

Номер индекса	**План**	**Положение индекса в строке**	**Колонка**
1	-	List format *списочный формат*	1
2	-	l	2
3	-	1,2	3
4	1	2,3	4
5	1,2	4,3	5
6	1,2,3	4,5	6

С 7 по 10-ый индексы следует естественная прогрессия.

Метки меняются наиболее медленно в первой позиции и наиболее быстро в высшей. Положение каждого индекса в упорядоченности меток является внутренней упорядоченностью меток, установленной GAMS (который обсуждался в разделе 12 в связи с упорядоченными множествами).

Упорядоченность индексов всегда такая же как для символов в операторе декларации. Можно декларировать их в упорядоченности, которая устанавливается при обращении, или выполнить присвоение новым идентификаторам с другой упорядоченностью. Сохраните в памяти строки (**guidelines**), при выборе и упорядочивании индексов.

Существует только один способ изменить упорядоченность, в которой метки для каждой индексной позиции появляются при выводе на печать – изменить упорядоченность появления меток в программе GAMS. Наиболее простой путь для достижения этого - продекларировать множество, которое будет существовать только для того, чтобы все метки были расположены в том порядке, в котором необходимо. В программе GAMS декларацию этого множества выполняется самой первой.

13.4.1. ПРИМЕР

Рассмотрим следующий пример. **X** является четырехмерным или имеет четыре индексных позиции. Он инициализируется, используя формат **parameter** (*параметра*) и затем выводится на печать, как показано ниже.

```
    set   i  first index        /first, second      /
          j  second index       /one, two, three    /
          k  third index        /a,     b           /
          l  fourth index       /i,     ii          /;

    parameter x(i,j,k,l)   a four dimentional structure  /

second.one.a.i         +inf,   first .three.b.i    -6.3161
first .one.b.i      5.63559,   second.two  .b.i    19.8350
second.one.b.ii   -17.29948,   first .two  .b.ii   10.3457
first .two.a.ii     0.02873,   second.one  .a.ii    1.0037
second.two.a.ii        +inf,   first .two  .a.i    -2.9393
first .one.a.ii     0.00000                         /;

    display x;
```

Результатом выполнения этих операторов будет следующий вывод (output)

```
    ----  17  PARAMETER X   A FOUR DIMENTION STRURTURE

    INDEX 1 = FIRST
                              I             II

    ONE     .B              5.636
    TWO     .A             -2.939
    TWO     .B                            0.029
    THERE   .B                           10.346

    INDEX 1 = SECOND


                              I             II

    ONE     .B              +INF          1.004
    TWO     .A                           -17.299
    TWO     .B                             +INF
    THERE   .B             19.835         10.346
```

Обратите внимание, что здесь есть две подтаблицы, по одной для каждой метки в первой индексной позиции. Также обратите внимание, что ноль в списке для **x('first', 'one', 'a', 'ii')** исчез, так как нулевая величина пресекается отдельно в каждой подтаблице. Упорядоченность меток не является таким же как в списке данных ввода.

13.5. DISPLAY КОНТРОЛИ

GAMS позволяет пользователю модифицировать количество строк и колонок меток при выводе на печать, точно также, как это делается при выводе на печать данных. Глобальный **display** контроль (**Global dispaly controls**) позволяет пользователю достигать большего эффекта, чем просто оператор **display**. Если специфика данных требует, чтобы они были записаны в особом формате, тогда вместо глобального **display** контроля (**global display control**) можно использовать локальный **dispaly** контроль (**local display control**). Следующие два подраздела пояснят, как работать с каждым **display** контролем.

13.5.1. ГЛОБАЛЬНЫЙ DISPLAY КОНТРОЛЬ

Наипростейшей из этих опций - показать после десятичной точки одну контролирующую цифру. Это влияет на число цифр, появляющихся после десятичной точки, для всего вывода на печать, следующего за оператором option.

Основной вид оператора

```
option decimal = "value";
```

где **"value"** - целочисленное значение между 0 и 8. Если используется 0, то десятичная точка пресекается. Ширина поля для вывода числа на печать не может быть изменена, может быть изменено только количество десятичных знаков, поэтому числа могут отображаться в обычном E-формате, т.е. в экспонентном представлении точности.

Рассмотрим продолжение примера из предыдущего раздела

```
option decimals  = 1;  display x;
```

Где необходимо, GAMS округляет числа или преобразует их в E-формат. Вывод выглядит следующим образом:

```
----  18  PARAMETER  X   A FOUR  DIMENTIONAL  STURTURE


INDEX 1 = FIRST
                           I            II

ONE    .B             5.6
TWO    .A            -2.9         2.8730E-2
TWO    .B                         10.3
THERE  .B            -6.3

INDEX 1 = SECOND

                           I            II

ONE    .B             +INF
TWO    .A             -2.9       -17.3
TWO    .B                          +INF
THERE  .B             19.8
```

13.5.2. ЛОКАЛЬНЫЙ DISPLAY КОНТРОЛЬ

Часто более полезным является контролировать количество десятичных знаков в числе отдельно для каждого идентификатора или для группы идентификаторов. Это выполняется с помощью оператора, основной вид которого следующий

option "ident": "d-value";

- **"ident"** представляет собой имя **parameter** (*параметра*), **variable** (*переменной*) или **equation** (*уравнения*);
- **"d-value"** - как прежде число от 0 до 8. **"d-value"** показывает количество десятичных знаков, которые будут выводиться на печать.

Именно **"d-value"** определяет точность, с которой будут распечатаны все **"ident".**

Формулировка, приведенная выше, может быть продолжена, чтобы контролировать компоновку данных, выводимых на печать. Общая форма записи следующая

option: "ident": "d-value": "r-value": "c-value" ;

Здесь **"r-value"** - количество индексных позиций, которые комбинируются в форме меток строк, **"c-value"** - количество названий колонок.

Чтобы проиллюстрировать работу локального **display** контроля, разовьем пример из предыдущего раздела дальше

option x:5:3:1; display x;

Вывод выглядит следующим образом

```
----  14 PARAMETER  X    a four dimentional  strurture

                          I           II
FIRST  .ONE  .B     5.63559
FIRST  .TWO  .A    -2.93930        0.02873
FIRST  .TWO  .B                   10.34570
FIRST  .THREE.B    -6.31610
SECOND .ONE  .A        +INF        1.00370
SECOND .ONE  .B                   -17.29948
SECOND .TWO  .A                       +INF
SECOND .TWO  .A    19.8350
```

Цифровые значения указаны с точностью пять знаков после десятичной точки, три метки используются, чтобы обозначить строку и одна - чтобы обозначить колонку. Поскольку это четырехмерная структура, здесь не осталось индексов, чтобы использовать их как подтабличные метки (в плане) и мы имеем результат, приведенный в одном куске. Оператор **option** контролируется непротиворечивостью с размерностью идентификатора, и если будет несоответствие, то будет сообщение об ошибке.

В следующем примере по два индекса в строке и колонке, и цифровые значения приведены с точностью до пяти знаков после запятой.

option x: 5:2:2; display x;

```
----  14 PARAMETER   a four dimentional structure

               A.I       A.II         B.I        B.II

FIRST .ONE                           5.63559
FIRST .TWO    -2.93930   0.02873                10.34570
FORST .THREE                        -6.31610
SECOND.ONE      +INF     1.00370               -17.29948
SECOND.TWO                 +INF     19.83500
```

13.5.3. ОПЕРАТОР DISPLAY, ЧТОБЫ ВОСПРОИЗВОДИТЬ ДАННЫЕ В СПИСОЧНОМ ФОРМАТЕ

Это особое использование локального **display** контроля - воспроизводить данные в списочном формате используя оператор **display**. Это происходит, когда метки для каждого значения считываются в таком же стиле как при инициализации данных **parameters** (*параметров*). Формат опции следующий:

option: "d-value": 0: "c-value";

и в этом случае **"c-value"** обозначает максимальное число элементов, выводимых на линии. Фактическое число будет зависеть от ширины страницы, числа и длины меток.

Используя тот же самый пример из предыдущего раздела

```
option x:5:0:1;  display x;

----  14 PARAMETER        a four dimentional srurture

FIRST .ONE  .B.I       5.63559
FIRST .TWO  .A.I      -2.93930
FIRST .TWO  .A.II      0.02873
FIRST .TWO  .B.II     10.34570
FIRST .THREE.B.I      -6.31610
SECOND.ONE  .A.I          +INF
SECOND.ONE  .A.II      1.00370
SECOND.ONE  .B.II    -17.29948
SECOND.TWO  .A.II         +INF
SECOND.TWO  .B.I      19.83500
```

Этот вывод хорошо иллюстрирует использованную упорядоченность меток. Первый индекс изменяется наиболее в последнюю очередь, последний индекс изменяется в первую очередь, каждый индексный набор управляет выводом от начала до конца вывода и до появления следующего комплекса индексов.

Эта упорядочивающая схема используется также в уравнениях, в записях колонок и отчете о решении. Все выполняется оператором **solve.**

СРЕДСТВА ДЛЯ ЗАПИСИ ВЫВОДА 14

14.1. ВВЕДЕНИЕ

В этом разделе представлены средства языка GAMS для выполнения записей вывода. Целью этих средств записи является вывод отдельных элементов данных модели в определенном формате в различные файлы. В отличии от оператора **display**, с помощью которого может быть выведено все множество значений для индексных идентификаторов (здесь идентификаторы - это имена, данные заданным элементам данных, такие как имена для **parameters** (*параметров*), **sets** (*множеств*), **variables** (*переменным*), **equations** (*уравнениям*) и т. д.), при использовании одного оператора **put** (оператор **put** - это оператор средств ввода, который будет описан в этом разделе) в файл выводится только один элемент данных. Структура средств для записи вывода является более комплексная, в ней имеется больше гибкости и возможностей контроля модели и результатов расчета через вывод отдельных элементов данных, но она требует выполнения большего объема работ при составлении программы, чем это требуется для вывода с помощью оператора **display**.

В этом разделе дано описание работы средств для выполнения записи вывода, синтаксис для доступа к средствам выполнения записи вывода, показаны возможности глобального форматирования документов с помощью файловых суффиксов, указывающих различные характеристики файла. Сами средства для выполнения записи вывода на базе информации, которая содержится в системе GAMS, не могут воспроизводить структуированные документы. Необходима дополнительная информация, которая задается пользователем при помощи численных суффиксов, присоединенных к идентификаторам, моделям или системам. Форматирование документов осуществляется с помощью файловых суффиксов и контролирующих литер.

Средства для выполнения записи вывода воспроизводятся автоматически во время выполнения GAMS. Документы записываются во внешний файл последовательно. Одновременно может выполняться заполнение только одной (текущей) страницы. Текущая страница помещается в буфер и автоматически записывается во внешний файл всякий раз, как только значение параметра, характеризующий длину страницы, становится больше заданного. Следовательно, средство для выполнения записи вывода имеет контроль только над текущей страницей, и GAMS не предусматривает возможность входа в заполненную часть файла вывода для изменения шаблона страницы документа. Однако, пока страница является текущей, информация, размещенная на ней, может быть затерта или перемещена как угодно.

14.2. СИНТАКСИС

Базисная структура средств для выполнения записи вывода в самой простой форме выглядит следующим образом

```
file fname(s) ;
put fname     ;
put item(s)   ;
```

где **fname** - представляет собой имя, используемое в модели GAMS, чтобы указать внешний файл. **Item(s)** - это любая информация, подлежащая выводу, например, поясняющий текст, метки, значения **parameters** (*параметров*), **variables** (*переменных*) или **equations** (*уравнений*). В показанной выше базисной структуре первая строка определяет одно или более файлов, в которые

предполагается выполнять запись. Вторая строка назначает один из ранее определенных файлов текущим файлом. Третья строка указывает, какая информация будет вводится в текущий файл.

14.3. ПРИМЕР

В этом разделе приведен небольшой пример, в котором продемонстрированы основные возможности средств для выполнения записи вывода. Пример базируется на модели "оптимизация транспортных перевозок" [TRANSPORT]. Для вывода необходимой пользователю информации (в данном учебном пособии это будет называться создание документа или отчета) в два внешних файла можно поместить приведенный ниже фрагмент программы в конце модели "оптимизация транспортных перевозок".

```
file factors /factors.dat/, results /results.dat/ ;
put factors ;
put 'Transportation Model Factors'///
    'Freight cost      ', f,
     @1#6, 'Plant capacity' /;
     loop(i, put @3, i.tl, @15, a(i)/);
     put /'Market demand'/;
     loop(j, put @3, j.tl, @15, b(j)/);

     put results;
put 'Transportation Model Results'// ;
     loop ((i,j), put i.tl, @12, j.tl, @24, x.l(i,j):8:4 /);
```

В первой строке определяются внутренние имена файлов **factors** и **results** и осуществляется связь с внешними именами файлов **factors.dat** и **results.dat.** Внутренние имена файлов используются внутри модели, чтобы указать имена файлов, которые являются внешними по отношению к модели. Вторая строка этого примера назначает файл **factors.dat** текущим файлом. Вывод всегда записывается в текущий файл (в данном случае в файл **factors.dat**).

Третья строка примера начинается с использования оператора **put**, далее следует текст **'Transportation Model Factors'**, который будет занесен в документ. Текст заключается в кавычки. Наклонные черточки **"/"**, следующие за текстом в кавычках, необходимы для указания операции возврат каретки.

Пример продолжается другим текстом (элементом данных) - скаляром **f.** Обратите внимание, что элементы данных разделяются между собой запятыми. Пробелы, запятые, наклонные черточки служат в качестве разделителей различных элементов данных вывода. Эти делители возвращают курсор к следующей после записанного элемента данных позиции-столбца документа. В большинстве случаев пробел и запятая могут быть взаимозаменяемыми, однако, запятая имеет преимущество на использование, так как ее использование предотвращает любую неоднозначность в записи.

В пятой строке вышеприведенного фрагмента программы курсор переводится в первую позицию шестой строки выводного файла, в который записываются другой текст. Литеры контроля курсора **#** и **@** служат для перемещения курсора к определенным строкам или позициям-колонкам в строке. Положение курсора определяется цифрами, следующими за литерами, контролирующими курсор (**#** и **@**). Запись оператора **put** заканчивается литерой точка с запятой **';'**.

Parameters (*параметры*) **a** и **b** записываются вместе с соответствующими метками **sets** (*множеств*). При использовании оператора **put** может быть записан только один элемент индексного множества. Чтобы записать все элементы **parameters** (параметров) **a** и **b**, оператор **put** должен быть вставлен внутрь оператора **loop** (*цикла*), который выполняет цикл (итерацию) через

индексное множество. В вышеприведенном примере метки элемента множества идентифицируются при помощи идентификатора множества и суффикса **.tl**. Как можно видеть, метки элементов множества располагаются, начиная с третьей позиции в строке, а значения **parameter** (*параметра*) **a** - с 15-й. Пример продолжается текстом в кавычках, за ним записывается **parameter** (*параметр*) **b**.

После завершения выполнения файл **factors.dat** будет выглядеть следующим образом:

```
Transportation Model Factors

Frieght Cost            90.00

Plant Capacity
      seattle           350.00
      san-diego         600.00

Market Demand
       new-york         325.00
       chicago          300.00
       toreka           275.00
```

При форматировании данного отчета применялось форматирование значений по умолчанию. Методы изменения вывода значений по умолчанию будут описаны в этом разделе позднее.

В последних двух строках примера текущим сделан файл **result.dat**, в который строка за строкой будут записаны значения, ассоциированные с **variable** (*переменной*) x вместе с соответствующими метками элементов множества. Результаты вывода **variable** (*переменной*) **x** форматируются с определенной шириной поля в 8 цифр, из них 4 цифры зарезервированы для цифр после десятичной запятой. Обратите внимание, что опции локального форматирования разделяются двоеточием. После выполнения операций файл **result.dat** будет выглядеть следующим образом:

```
Transportation Model Results

seattle     new-york      50.0000
seattle     chicago      300.0000
seattle     toreka         0.0000
san-diego   new-york     275.0000
san-diego   chicago        0.0000
san-diego   toreka       275.0000
```

Это краткое введение показывает возможности средств для выполнения записи вывода, такие как создание отчета (документа), который обеспечивает вывод к файлу для использования другой компиляционной программой или просто для вывода на печать промежуточных расчетов. В следующих подразделах все средства для выполнения записи вывода будут рассмотрены более детально и пояснены специальными примерами.

14.4. ФАЙЛЫ ВЫВОДА

Как было отмечено ранее, оператор **put** позволяет пользователю выполнять запись во внешние файлы.

14.4.1. ОПРЕДЕЛЕНИЕ ФАЙЛОВ

Полный синтаксис для определения файлов следующий:

file fname text /external file name/

где: **file** - ключевое слово, используемое для определения файлов;
fname - внутреннее имя файла, используется в моделе GAMS, чтобы указать на внешний файл.

Внешние файлы являются действительными файлами, в который записывается вывод. Во время декларации файла внешнее имя файла и поясняющий текст необязательны. Если внешнее имя файла опускается, GAMS обеспечит внешнее имя файла по умолчанию, (часто это **fname.put**). Используя один оператор **file** можно выполнить определение нескольких файлов.

Рассмотрим следующий пример.

```
file  class1
      class2  this defines a specific external file  /report.txt/
      con     this defines access to the screen for  dos system;
```

Первый файл вывода опознается в модели под именем **class1** и для совместимой DOS системы направляется к файлу по умолчанию **class1.put**. Второй файл вывода опознается в модели под именем **class2** и он направляет к определенному внешнему файлу report.txt. И последним определяется особое внутреннее имя **con**, которое служит для выполнения записи вывода на экран системы DOS. Вывод на экран может быть очень полезен при отладке программы, так как сообщает пользователю о различных аспектах модели во время ее выполнения.

14.4.2. ПРИСВОЕНИЕ ФАЙЛОВ

Оператор put используется для назначения текущего файла и для записи элементов данных в этот файл.

Полный синтаксис оператора put следующий:

put fname item(s) fname items ... ;

Как показано в синтаксисе, используя один оператор **put** может быть выполнено присвоение множеству файлов, записанным последовательно. Необходимо помнить, что текущим всегда является только один файл. После завершения вывода элементов данных в один текущий файл записывается следующее внутреннее имя файла. Текущий файл переприсваивается. Переприсвоение основывается на следующем в операторе **put** внутреннем имени файла. Последнее использованное внутренне имя в операторе **put** остается в качестве текущего до тех пор, пока следующий оператор **put** не использует внутреннее имя.

14.4.3. ЗАКРЫТИЕ ФАЙЛА

Во время выполнения программы GAMS для закрытия файла используется ключевое слово **putclose.**

Синтаксис следующий:

putclose myfile item(s)

где: **myfile** - внутренне имя файла, который должен быть закрыт;
item(s) - последний вводимый в файл элемент данных, прежде чем он будет закрыт.

Если внутреннее имя файла опущено из оператора **putclose**, будет закрыт текущий файл. Помните, что после использования команды **putclose** закрытый файл не может быть переопределен, чтобы использовать его еще раз. Если необходимо заново использовать закрытый файл, его нужно сделать текущим, использовав, как обычно, оператор **put**. В этом случае в зависимости от значение суффикса, который будет присоединен к имени файла, существующий файл будет или перезаписан, или присоединен к файлу (см. ниже).

Закрытие файла может быть успешно применено, если вы хотите представить опции solver внутри модели GAMS. В этом случае в представление опций выполняется с помощью оператора **put**, а закрытие файла с помощью команды **putclose** выполняется перед оператором **solve**.
Следующий пример иллюстрирует создание и закрытие файла опций для solver MINOS

```
file opt  Minos option file   /minos5.opt/;
put opt;
put  ' Iteration limit     500'/
     ' Feasibility tolerance 1.0E-7'/  ;
putclose opt;
```

Этот фрагмент программы должен бы быть помещен внутри модели GAMS перед оператором **solve.**

14.4.4. ПРИСОЕДИНЕНИЕ К ФАЙЛУ

Оператор **put** - записывающее средство, он обладает возможностью присоединить или перезаписать существующий файл. Файловый суффикс **.ap** определяет, какая операция будет иметь место. По умолчанию суффикс принимает значение **0**, и это означает перезапись существующего файла, в то время как значение **1** означает присоединение к файлу. Рассмотрим пример с существующим файлом **report.txt**. Использовав следующую запись, мы присоединяем к нему выводные элементы данных:

```
class2.ap = 1;
```

Начиная с этого оператора любые выводимые элементы данных будут добавляться в конец существующего файла. Если файл не существовал, он должен быть создан.

14.5. ФОРМАТ СТРАНИЦЫ

Страницы внутри файла можно форматировать. С помощью файловых Суффиксов можно задавать параметры страницы, такие как формат распечатки, размер страницы, ширина страницы и полей в случае, если выполняется вывод на печать текстовых фрагментов. Для форматирования используются следующие файловые суффиксы.

Суффиксы		Описание
.pc	Print control	Используется для назначения формата внешнего файла. Последние три опции создают разделенные

	(*контроль распечатки*)	файлы, которые являются особенно полезны при подготовке вывода для использования в других компьютерных программах, таких как электронная таблица.
	0	Стандартное размещение страницы, основанное на текущем размере страницы. Частично страницы бывают выравнены с помощью пустых строк. Файловый суффикс **.bm**, используемый с этой опцией контроля распечатки, является только функциональным.
	1	Формат страницы "Фортран". Эта опция ставит цифру один в первой позиции-колонке первой строки каждой страницы в стандартной конвенции "Фортран".
	2	Продолжение страницы (по умолчанию). Эта опция подобна нулевой опции **.pc**, только не выполняется выравнивание страниц пустыми строками для заполнения страницы.
	3	Вставлены литеры контроля страницы ASII.
	4	Форматированный вывод. Не цифровой вывод в кавычках. Каждый элемент данных разделен пробелом.
	5	Форматированный вывод. Не цифровой вывод в кавычках. Каждый элемент данных разделен запятыми.
	6	Форматированный вывод. Не цифровой вывод в кавычках. Каждый элемент данных разделен табуляциями.
.ps	page size *размер страницы*	Используется, чтобы определить число строк, которые могут быть помещены на одной странице документа. Может быть приведено в исходное положение пользователем в любом месте программы. Будет сообщение об ошибки, если будет указано меньшее значение, чем число строк, которое уже было записано для текущей страницы. Максимальное число строк **130**. Значение строк по умолчанию - **60**.
.pw	page width *ширина страницы*	Используется, чтобы определить число позиций-колонок, которое может содержаться в одной строке страницы. Может быть приведено в исходное положение пользователем в любом месте программы. Будет сообщение об ошибке, если будет указано меньшее значение, чем число позиций-колонок, которое уже было записано для текущей страницы. Максимальное значение **255**. Значение по умолчанию - **60**.
.tm	Top margin *Верхнее поле*	Число пустых строк, которые формируют верхнее поле страницы. Эти строки являются дополнительными к числу строк, определенных файловым суффиксом **.ps**. Значение по умолчанию равно **0**.
.bm	botton margin *нижнее поле*	Число пустых строк, которые формируют нижнее поле страницы. Эти строки являются дополнительными к числу строк, определенных файловым суффиксом **.ps.** Эта опция является функциональной только для **.pc** с опцией **0**. Значение по умолчанию равно **0**.
.case	*используется в*	Используется, чтобы определить случай, когда в выводной файл выводятся на печать алфавитные

	случае по алфавиту 0 1	литеры. Для распечатки вызывает смешанный регистр клавиатуры. Для распечатки вызывается регистр заглавных букв (верхний регистр), вне зависимости от того какие буквы (заглавные или прописные) были использованы для ввода.

Чтобы проиллюстрировать использование этих файловых суффиксов, отформатируем **report.txt** следующим образом: страница вывода имеет 72 знака в ширину, 58 строк, дополнительное верхнее поле включает 6 строк, используются литеры контроля страницы ASCII (включенные в каждую 64-ю строку), вывод должен быть распечатан заглавными буквами.

```
file class2  /report.txt/;
class2.pw  =  72;  class2.ps  = 58;   class2.tm = 6;
class2.pc  =  3;   class2.case  = 1;
```

Если для суффикса **print control** (*контроля распечатки*) (**.pc**) принять значения 4, 5, 6, то данные будут сжаты, а информация об оставленных промежутках, выполненная пользователем через литер **@**, будет игнорироваться. Однако эти значения могут быть использованы, чтобы направить данные для прочтения электронной таблицей.

14.6. ЗОНЫ СТРАНИЦЫ

На каждой странице документа имеется три независимые записывающие зоны: **title block** (*блок названия*), **header block** (*головной блок*) и **window** (*окно*). Это очень удобно, когда они являются зонами страницы, которая остается относительно постоянной для всего документа. Для **title blok** (*блока названия*) и **header block** (*головного блока*) предусмотрено использование для создания организационной информации документа, **window** (*окно*) используется для специальных отчетов.

Записывающие зоны всегда расположены последовательно, как показано ниже в диаграмме. Важно помнить, что по существу **title block** (*блок названия*) и **header block** (*головной блок*) являются такими же, как **window** (*окно*), и для них применимы точно такие же правила синтаксиса, как для **window** (*окна*). Однако, **window** (*окно*) должно быть на каждой странице, в то время как **title block** (*блок название*) и **header block** (*головной блок*) необязательны. Также необходимо помнить, что если уже один раз в **window** (*окно*) была выполнена запись, то следующая модификация **title** (*заглавия*) и **header block** (*головного блока*) будут приведены не на текущей странице, а на следующей. Запись в **window** (*окно*) является тем, что в конечном счете заставляет страницу быть записанной.

Title block *Блок названия*
Header block *Головной блок*
Window *окно*

В примере из раздела 2 все данные были записаны в **window** (*окно*). Если необходимо уточнение, которое обеспечит появление имени модели на каждой странице, то включают **title block** (*блок названия*). Если необходимо вывести на печать отказ или инструкцию, которая должна появляться на каждой странице, дополнительно используется **header block** (*головной блок*). Если однажды информация помещена в **title** (*название*) или **header block** (*головной блок*), она будет выводится на печать на каждой странице до тех пор, пока не будет выполнена модификация. Использование **title block** (*блока названия*) и **header block** (*головного блока*) особенно полезно при составлении длинных документов, занимающих много страниц.

14.6.1 ДОСТУП К РАЗЛИЧНЫМ ЗОНАМ СТРАНИЦЫ

Каждый из этих трех зон страницы доступен при помощи различных ключевых слов. Эти слова:

puttl писать в **title block** (*название*)
puthd писать в **heared block** (*головной блок*)
put писать в **window** (*окно*)

Размер любой зоны внутри данной страницы базируется всецело на числе определенных для нее строк. Необходимо помнить, что общее число строк для всех зон должно соответствовать определенным размерам страницы. Если общее число строк, записанное для **title** (*названия*) и **header block** (*головного блока*) равно или превышает размер страницы, то в программой распечатке будет дано сообщение об ошибке. Когда встречается такое сообщение, то оно означает, что на странице не осталось места для записи **window** (*окна*).

Как было упомянуто выше, синтаксис для записи вывода элементов данных в любую из этих трех зон страницы одинаков, отличаются только начальные ключевые слова. Ниже иллюстрируется выполнение записи **'GAMS Put Example'** в **title block** (*в блок названия*) файла **report.dat**

```
puttl class2 'GAMS Put Example' ;
```

В этом случае текст **"GAMS Put Example"** будет помещен в первую колонку первой строки **title block** (*блока названия*). Любая следующая страница файла **report.dat** будет начинаться с этой информации.

Если название было изменено или **header block** (*головной блок*) был запущен после того, как была выполнена запись в **window** (*окно*) текущей страницы, изменение появится на следующей странице, а не на текущей.

14.6.2 ЗАМЕЩЕНИЕ СТРАНИЦ

Замещение страниц имеет место всякий раз, когда случается, что страница полная. Однако необходимо помнить, что для того, чтобы страница была записана в файл вывода **window** (*окно*) должно быть использовано. Когда страница не имеет вывода в **window** (*окне*), страница в файл не записывается вне зависимости от того, имеются ли элементы данных в **title block** (*блоке названия*) или в **header block**(*головном блоке*). Чтобы заставить страницу с пустым **window** *(окном*) быть распечатанной, нужно записать в него что-то безвредное, например

```
put ' ';
```

Теперь **window** (*окно*) страницы будет инициализировано и страница будет записана в файл.

14.7. ПОЛОЖЕНИЕ КУРСОРА В СТРАНИЦЕ

Курсор располагается на следующей позиции от последней записанной литеры до тех пор, пока курсор не будет перемещен с помощью специальных контролирующих символов:

Контроль курсора	Описание
#n	перемещает курсор на n-ую строку текущей страницы
@n	перемещает курсор на n-ую позицию-колонку текущей строки
/	перемещает курсор к первой позиции-колонке следующей строки. Действует как разграничитель между элементами данных вывода.

В дополнение к числам, за символами **#** и **@** может быть записано любое выражение или символ с численным значением. Следующий пример иллюстрирует использование вышесказанного. Рассмотрим, как выполнить запись значение **parameter** (*параметра*) **a(i,j)** в табличной форме.

```
file  out;  put  out;
scalar   col  column   number  /1/;
loop(i,
     loop (j, put @col a(i,j);  col=col + 10;)  put/;
     );
```

14.8. СИСТЕМА СУФФИКСОВ

Для получения информации об управлении GAMS имеется система суффиксов. Полный список системы суффиксов приведен ниже.

Система суффиксов	Описание
.date	дата выполнения программы
.ifile	имя файла ввода
.ofile	имя файла вывода
.page	текущая страница файла
.rdate	перезапуск файла данных
.rfile	перезапуск имени файла
.rtime	перезапуск времени файла
.sfile	сохранить имя файла
.time	время выполнения программы
.title	название модели как было определено **$title**

В качестве иллюстрации рассмотрим пример из предыдущего раздела. Модифицировав оператор **puttl** можно присоединять номер страницы к названию отчетного файла. Запись модифицированного оператора выглядит следующим образом

```
puttl  class2  ' GAMS Put Example', @65, 'page', system.page ///;
```

В этом случае за словом **'page'** следует номер страницы, запись появляется в **title** (*названии*) каждой страницы, начиная с позиции-колонки 65.

14.9. ВЫВОДНЫЕ ЭЛЕМЕНТЫ ДАННЫХ

Выводными элементами для оператора **put** являются следующие формы:

- **text** Любой текст в кавычках, метка элемента **set** (*множества*) или текст, любой идентификаторный символьный текст или содержание системы суффиксов (см. предыдущий подраздел)
- **numeric** Значения, ассоциируемые с **parameters** (*параметрами*), **variables** (*переменными*), **equations** (*уравнениями*) или какими либо суффиксами модели
  - **set value** Представляет наличие элементов **set** (*множеств*) и принимает только значение **yes** или **no.**

14.9.1. ТЕКСТОВЫЕ ЭЛЕМЕНТЫ ДАННЫХ

Выводными текстовыми элементами данными является комбинация литер вне зависимости от того, какие кавычки использованы (одинарные или двойные). Однако длина текста в кавычках, как и для любых выводных элементов данных, имеет предел. Выводные элементы данных не могут быть продолжены за границы страницы. Когда запись выводных элементов данных превышает определенную ширину страницы, в конце строки появляется несколько звездочек и в распечатку программы записывается сообщение об ошибке.

В дополнение к тексту в кавычках возможен вывод других текстовых элементов данных через использование системных или идентификаторных суффиксов.

Суффиксы		Описание
.ts	идентифицирующий символ текста	Выводит на печать текст, ассоциированный с каким либо идентификатором.
.tl	множество меток элементов	Выводит на печать отдельные метки элементов **set** (*множества*)
.te(index)	множество элементов текста	Выводит на печать текст, ассоциированный с элементом множества. Необходимо помнить, что суффикс **.te** требует управляющий индекс. Этот управляющий индекс контролирует **set** (*множество*), которое будет выводиться на печать и не обязательно должно быть такое же, как контролируемое множество. Часто используется подмножество индексов контролируемого **set** (*множества*).
.tf	Полнота текста	Используется для контроля вывода на печать недостающего текста для элементов **set** (*множества*) (по умолчанию = **2**)
	0 1	Не полный

	2	Полный только существующий Полный всегда

Следующий пример иллюстрирует применение вышесказанного:

```
file  out;  put  out;
set  i  master set of sites /i1  Seattle, i2  Portland
                            i3  San fracisco, i4 Los Angeles
                            i5 /
     j  subset of sites     /i3*i5/ ;

    put j.ts /;
    loop(j, put j.tl, i.te(j) /);
```

В результате файл **out.put** будет выглядеть так

```
subset of sites
i3   San Francisco
i4   Los Angeles
i5   I5
```

В этом тексте символьный текст для идентификатора подмножества **j** записан первым. Затем следует текст с метками подмножества **j,** и он относится к элементами текста, который находится в области множества **i**. Обратите внимание, что управляющее множество **j** используется для элементов текста множества **i**. Так как не было элементов текста, относящегося к элементу **i5** множества **i**, метка элемента множества была выведена на печать снова. Если перед последней строкой поместить следующую запись

```
out.tf = 0;
```

тогда отсутствующие элементы текста не помещаются с метками текста.

В результате файл **out.put** будет выглядеть так

```
subset of sites
i3   San Francisco
i4   Los Angeles
i5
```

14.9.2. ЧИСЛЕННЫЕ ЭЛЕМЕНТЫ ДАННЫХ

Синтаксис, используемый для вывода на печать численных элементов данных, является, в основном, удобным для работы. Для вывода **parameter** (*параметра*) используется только идентификатор вместе с индексами. Для вывода значений **variable** (*переменных*) и **equation** (*уравнений*) идентификатор комбинируется с одним из суффиксов **variable** (*переменной*) или **equation** (*уравнения*).

Суффикс	**Описание**
.l	Уровень или маргинальное значение
.lo	Нижняя граница
.m	Маргинальное или двойственное значение
.prior	Приоритет
.scale	Масштабирование
.up	Верхняя граница

14.9.3. ЗНАЧЕНИЯ ЭЛЕМЕНТОВ ДАННЫХ МНОЖЕСТВА

Значения элементов данных множества является удобными для работы с ними. Для вывода значения **set** (*множества*) должен быть использован только идентификатор вместе с индексом множества. В примере из раздела 8.1 рассмотрим измененный оператор **loop**, чтобы прочитать

```
loop(i,  put i.tl, j(i),  '  ',  i.te(j) /);
```

Результирующий файл вывода будет выглядеть так

```
subset   of sites
i1     NO  Seattle
i2     NO  Portland
i3    YES  San Francisco
i4    YES  Los Angeles
i5    YES
```

Вторая колонка указывает, какие элементы относятся к множеству **j**, а какие нет.

14.10. ГЛОБАЛЬНОЕ ФОРМАТИРОВАНИЕ ЭЛЕМЕНТОВ ДАННЫХ

Очень часто необходимо контролировать формат элементов данных, выводимых на печать. В этом разделе мы рассмотрим, как это делается. В зависимости от цели форматирования выводимые на печать элементы данных разделены на четыре категории: метки, численные значения, значения **set** (*множества*) и текст. Для каждой категории возможно глобальное форматирование ширины поля и выравнивание поля.

14.10.1. ВЫРАВНИВАНИЕ ПОЛЯ

Для выравнивания поля используются следующие суффиксы.

Суффиксы	Описание	По умолчанию
.lj	Выравнивание меток	2
.nj	Выравнивание цифр	1
.sj	Выравнивание значений множеств	1
.tj	Выравнивание текста	2

Возможно глобальное выравнивание для любой из категорий

Выравнивание	опции
Правое	1
Левое	2
По центру	3

14.10.2. ШИРИНА ПОЛЯ

Для форматирования ширины поля используются следующие суффиксы.

Суффиксы	Описание	По умолчанию	Максимум
.lw	Ширина поля меток	12	
.nw	Ширина поля цифр	12	
.sw	Ширина поля значения множества	12	20
.tw	Ширина поля текста	0	

Ширина поля - это количество позиций-колонок в строке, предназначенных для поля. Для суффиксного значения **0** ширина поля может принимать различные значения. Это означает, что определенная ширина поля является ограничением для ширины, которую занимают выводимые на печать элементы данных. Если текстовый выводной элемент данных не соответствует определенной ширине поля, то будет выполняться усечение текста. Усечение текста выполняется справа. Для численных выводных элементов данных чтобы разместить число в заданном поле десятичная часть числа округляется или используется экспоненциальный формат. Если число все таки слишком большое, в выводном файле появятся звездочки.

Рассмотрим пример, где необходимо установить глобальную ширину поля для цифр равную 4, в отличие от ширины поля для файла **out.put** по умолчанию, равной 12. Для этого необходимо использовать следующий оператор

```
out.nw  = 4;
```

14.11. ЛОКАЛЬНОЕ ФОРМАТИРОВАНИЕ ЭЛЕМЕНТОВ ДАННЫХ

Очень часто необходимо форматировать только отдельные элементы данных **put**. Для этого мы используем локальное форматирование признака, который изменяет (замещает) установки глобального форматирования.

Синтаксис выполнения локального форматирования следующий

```
item: {<>} widht:decimals;
```

Элемент данных будет выведен на печать с выравненными символами, шириной поля и определенным количеством десятичных знаков в числе. Определение количества десятичных знаков в числе является действительным только для вывода.

Применяются следующие локальные выравнивания символов

> правое выравнивание
< левое выравнивание
<> выравнивание по центру

Пропуск каких либо компонентов вызывает установку соответствующего глобального формата, который будет использоваться. Как и для глобального форматирования, когда ширине поля задается значение **0**, ширина поля является переменной в размерах. Элементы данных, ширина поля и количество десятичных знаков в числе разделяются двоеточиями, как показано выше. Использование возможностей локального форматирования, таких как включение каких либо компонентов для выравнивания, ширины поля или количества десятичных в числе, является совершенно необязательным.

Следующий пример иллюстрирует некоторые возможности локального форматирования:

```
* default justification and a field width of variable size
* with no decimals
loop (i, put dist(i):0:0 /);

put 'Right justified comment' :<>50;
    'Center justified truncated  comment' :<>20;

* left justified scalar with a zix space field width and
* two decimalls

    put f:<6:2;
```

14.12 ДОПОЛНИТЕЛЬНЫЙ ЧИСЛЕННЫЙ КОНТРОЛЬ ВЫВОДА НА ПЕЧАТЬ

В дополнение к цифровой ширине поля и цифровому выравниванию, которые были описаны в предыдущем подразделе, для цифрового вывода на печать могут быть также глобально определены следующие файловые суффиксы

Суффиксы		Описание
.nd	количество десятичных знаков в числе	Устанавливает количество десятичных знаков для цифровых элементов данных. Значение 0 означает, что будет выведена на печать только целочисленная часть числа. Максимальное значение - 10. Значение по умолчанию - 2.
.nr	формат численного округления	Позволяют выводить на печать числовые значения в экспоненциальном формате. Иначе эти значения были бы выведены на печать как ноль, потому что они меньше, чем можно показать при помощи суффикса .nd. Такая же ситуация возможна, когда число меньше, чем определено при помощи суффикса .nd, но больше чем уровень допустимых от нуля отклонений, устанавливаемый при помощи суффикса .nz. Очень часто важно знать, что эти маленькие значения существуют.
	0	вывод на печать в F или E формате
	1	округленные до размеров установленного поля
	2	вывод на печать в экспоненциальном формате
.nz	число допустимого от нуля отклонения	устанавливает уровень допустимого отклонения, при которых число существует

??? нет страницы 152.
Стр. 153.

```
comb2    123.457       0.123          1.2345679e-4
comb3    123.457       0.123          0.000
comb4    1.23457e+2    0.12345679     0.00012346
comb5    1.234567e+2   1.234567e-1    1.234567e-4
```

В comb1 вывод на печать значения переключается к экспоненциальному формату, когда значение становиться меньше, чем число позволенных десятичных. Это запускается при помощи суффикса .nr , который устанавливает нулевое значение. Особенно интересными являются value3 для comb2 и comb3. Value3 больше, чем уровень допустимого от нуля отклонения в.nz, но меньше, чем десятичные, позволенные .nd. Так как .nr утанавливается 0, в comb2 значение отображается в экспоненциальном формате. Для comb3, .nr устанавливается 1, а так как это маленькое значение, то оно округляется до нуля. Конечно, value4 меньше, чем .nz, поэтому устанавливается к нулю. Учти, что в comb4 значения являются

отображенными или в експоненциальном формате, или округленными до ширины поля, размер ширины поля и количесво десятичных знакв позволили выполнить это. В comb5 value1 округляется к целочисленному, потому что .nd установлено 0. Другие велличины меньше, но так как .nr установлен 0, то они отображаются в экспоненциальном формате.

14.13. КОНТРОЛЬ КУРСОРА

Ранее описаны различные способы вывода элементов данных с помощью оператора **put**. В этом разделе описываются средства, относящиеся к определению положения этих элементов данных в файле вывода. GAMS имеет несколько файловых суффиксов, которые определяют положение курсора и последней строки в файле. Они служат средством для форматирования выводных элементов данных в документах. Эти суффиксы группируются по записывающим зонам: **title block** (*блок названия*), **header block** (*головной блок*) и **window** (*окно*), для которых они имеют силу.

14.13.1. КУРСОР В ТЕКУЩЕЙ КОЛОНКЕ

Эти суффиксы имеют численные значения, соответствующие координатам **window** (*окна*) страницы. Поэтому они вместе с буквами контроля курсора применяются для управления положением курсора в файле вывода.

.cc	курсор в текущей колонке в **window** (*окне*)
.hdcc	курсор в текущей колонке в **header block** (*головном блоке*)
.tlcc	курсор в текущей колонки в **title block** (*блоке названия*)

Изменение значений с помощью суффикса **.cc** означает изменение окончания оператора **put**. Поэтому значение **.cc** остается постоянным для следующего оператора **put** на протяжении всей записи элементов данных, даже если на печать выводятся сложные элементы данных.

Следующий пример иллюстрирует изменение суффиксов, контролирующих курсор и использование литер, контролирующих курсор. Пример не сложный, но поучительный

```
scalar   lmarg  left  margin  /6/;
file  out;  put  out;
put @(lmarg+2) 'out.cc = ',  out.cc:0:0  '  ' ;
put @out.cc  'x'/@out.cc 'y'/@out.cc  'z  ';
put 'out.cc = '  out.cc:0:0;
```

В файле **out.put** будет следующий вывод

```
out.cc  = 1  x
             y
             z  out.cc  = 23
```

Первоначально скаляру **lmarg** назначается специальное значение, чтобы использовать его в качестве управления табуляцией. Символы, которые обычно имеют силу управляющих значений, такие как **margins** или **tabs**, являются часто полезными для больших структурных документов. Первый оператор **put** использует букву контроля курсора текущей колонки, чтобы передвинуть

курсор. В этом примере курсор передвинут на позицию **8**, где выводятся на печать **out.cc** и его значение.

Второй оператор **put** иллюстрирует изменение суффиксов, контролирующих курсор, чтобы записать буквы **x, y** и **z** на трех различных строках. Каждая является предшествующей курсору, передвинутому к значению **out.cc**. Первоначально значение для суффикса, контролирующего курсор, было равно **20**. Так как одиночный оператор **put** использовался для этих трех элементов данных, значение **out.cc** остается постоянным и ,следовательно, буквы оканчиваются в тех же самых позициях-колонках. Следующий оператор put изменяет значение **out.cc** на **23**, которое является положением курсора в конце второго оператора **put** (обратите внимание , что дополнительные пустые места выводятся на печать с буквой **z**).

14.13.2. КУРСОР В ТЕКУЩЕЙ СТРОКЕ

Эти суффиксы имеют численные значения, соответствующие координатам в окне страницы. В связи с этим они могут быть использованы вместе с буквами, контролирующими курсор, чтобы управлять положением курсора в файле вывода.

.cr курсор в текущей строке **window** (*окна*)
.hdcr курсор в текущей строке **header block** (*головного блока*)
.tlcr курсор в текущей строке **title block** (*названия*)

Изменение значений с помощью суффикса **.cr** означает изменение окончания оператора **put**. Поэтому значение **.cr** остается постоянной во всех записях элементов данных для следующего оператора **put**, даже если на печать выводятся много элементов. Это подобно поведению суффикса **.cl**.

14.13.3. КОНТРОЛЬ ПОСЛЕДНЕЙ ЛИНИИ

Эти суффиксы контроля последней линии используются в записывающей зоне.

.ll последняя линия, используемая в **window** (*окне*)
.hdll последняя линия в **header block** (*головном блоке*)
.tlll последняя линия в **title block** (*блоке названия*)

Это непохоже на контроль строки или позиции-колонки в строке, суффикс последней линии изменяется постоянно. Суффикс последней линии особенно полезен для видоизменения записывающей зоны страницы.

Суффиксы **.tlll** и **.ndll** могут не содержать значения, применимые к текущей страницы, потому что когда **title** (*название*) или **header block** (*головной блок*) видоизменяются, они направляют к **title** (*названию*) или к **header block** (*головному блоку*) следующей страницы всякий раз, как выполнена запись в **window** (*окно*) текущей странице.

Этот суффикс может быть использован не только для того, чтобы определить последнюю использованную строку в записывающей зоне, он может быть также использован для удаления линий внутри этой зоны.

В следующем примере **header block** (*головной блок*) будет полностью удален при помощи присвоения суффиксу **.hdll** значение равное **0**.

```
file out;
puthd  out  'This header statement will be eliminated';
out.dhll = 0;
```

В этом примере **header block** (*головной блок*) сначала был записан. При изменении значения суффикса **.hdll** на **0**, курсор был перемещен на верх header block (головного блока). Поэтому **header block** (*головной блок*) не будет записан до тех пор, пока что-то новое не будет присоединено к **header block** (*головному блоку*).

14.14. КОНТРОЛЬ РАЗМЕЩЕНИЯ СТРАНИЦ

В дополнение к автоматическому перемещению страниц в файл (переносу страницы во внешний файл), которое имеет место, когда достигается конец страницы, страница может быть записанной во внешний файл раньше. Для выполнения этой операции используется ключевое слово **putpage. Putpage** заставляет текущую страницу быть немедленно записанной в файл, делая существующей для оператора **put** новую страницу. В этой простейшей форме ключевое слово **putpage** используется для того, чтобы пропустить текущую страницу. Такая же операция может быть выполнена с элементами выводных данных. Когда необходимо выполнить какую то запись элементов выводных данных во внешний файл раньше, страница записывается в файл, включая элементы выводных данных, которые относятся и к оператору **putpage. Putpage** является в сущности другим вариантом оператора **put**. В следующем операторе текст в кавычках помещен в текущую страницу, которая затем записывается в файл **out.put.**

```
putpage  out 'This text is placed in window and the page ands';
```

В GAMS имеется два дополнительных файловых суффикса, которые могут помочь пользователю в определении того, какая страница файла в данный момент

Суффикс		Описание
.lp	последняя страница	указывает количество страниц, которое уже имеется в документе. Необходимо помнить, что если для этого суффикса установить значение 0, это не удалит страницы, которые ранее уже были записаны в файл.
.ws	размер окна	Показывает количество строк, которые могут быть помещены в **window** (*окно*), учитывая количество линий, которое требуется для **title block** (*блока названия*) и **header block** (*головного блока*) текущей страницы, и существующие размеры страницы. Значение файлового суффикса **.ws** рассчитывается GAMS и не может изменяться пользователем. Этот суффикс полезен для ручной нумерации страниц, когда он используется вместе с файловым суффиксом **.ll**.

14.15. ИСКЛЮЧЕНИЕ РЕГУЛИРОВАНИЯ

В этом разделе рассматривается исключение регулирования. Исключение регулирования долларовым контролем для контроля вывода на печать отдельных выводных элементов данных с операторами **put** используется также как с другими операторами GAMS. В следующем примере оператор **put** выполняет вывод на печать, если долларовые условия **true** (*истина*). Если нет, то оператор **put** игнорируется.

```
put$(flag gt 10)  'some  output  items';
```

14.16. ИСТОЧНИКИ ОШИБОК АССОЦИИРУЕМЫЕ С ОПЕРАТОРОМ PUT

Существуют два типа ошибок, которые могут иметь место при использовании средств для записи вывода:

- ошибки синтаксиса;
- ошибки **put**.

Следующие подразделы рассматривают каждый из этих типов ошибок.

14.16.1. ОШИБКИ СИНТАКСИСА

Ошибки синтаксиса появляются из-за неправильного использования языка GAMS. Эти ошибки такие же или похожие на те, которые могут иметь место в любом в другом месте модели GAMS (такие как отсутствие кавычек, неопределенные идентификаторы, неконтролируемые множества или неправильное использование ключевых слов или суффиксов). Эти ошибки всегда выявляются в время компилирования программы и они всегда фатальны для выполнения программы. Ошибки этого типа распознаются в распечатке программы по положению ошибки с символом **$** и сопровождающим номером ошибки. Распечатка программы включает краткое описание возможной причины ошибки.

14.16.2. ОШИБКИ PUT (ВЫВОДА)

Ошибки **put** (*вывода*) являются уникальными для средств написания **put.** Этот тип ошибок имеет место во время выполнения программы и причиной таких ошибок является нарушение одного или более файловых или страничных признаков. Эти ошибки не являются фатальными и записываются в конце распечатки программы. Обычно они имеют место, когда оператор **put** пробует записать за границами страницы, например, передвинув курсор с буквой **@** в зону за границу ширины страницы. Другая типичная ошибка – невозможность открыть определенный файл, выход за границу страницы или неподходящее значение, присваиваемое суффиксу. Для многих таких ошибок в выводном файле в месте нахождения ошибки будет добавлен ряд звездочек. Так как ошибки **put** (*вывода*) не фатальны и не имеют особого значения в выводном файле, их представление иногда является незамеченным. Без оценки распечатки программы эти ошибки **put** (*вывода*) могут быть не выявленными, особенно в больших выводных файлах. Поэтому GAMS имеет следующие файловые суффиксы, которые помогают выявить эти ошибки:

.errors позволяет вывести на печать количество ошибок **put** (*вывода*) в файле

Чтобы проиллюстрировать использование этого суффикса, оператор, приведенный ниже, должен быть помещен в любую точку программы. Он позволяет выявить количество ошибок, которые имеются в программе, начиная с места расположение этого оператора. Результаты об ошибках могут быть выведены в тот же выводной файл, в другой файл или на пульт оператора (операторский терминал).

```
putpage error /// '*** put errors: ', out.errors:0:0,' ***'/;
```

В этом примере принимается, что файлы **out.put** и **error.put** были предварительно определены при помощи оператора **file**. При помощи этого оператора количество ошибок **put** (*вывода*), которые имеют место в файле **out.put**, будут выведены на печать в файле **error.put**. Если операторский терминал имеет выводное устройство, то использование **putpag** позволяет непосредственно вывести на экран системы DOS результаты действия этого оператора.

14.17. ПРИМЕНЕНИЕ ЭЛЕКТРОННЫХ ТАБЛИЦ/БАЗ ДАННЫХ

Этот последний раздел представляет простой пример подготовки вывода для электронных таблиц, баз данных или других пакетов программных средств, которые можно перемещать в отдельные файлы. Как упомянуто в разделе 14.2, элементы данных вывода могут быть подготовлены при помощи разделителя-запятой, а текстовые элементы данных берутся в кавычки. Это выполняется при помощи суффикса **.pc** опция **5**. Разделенные файлы отличаются от обычных файлов **put** (*вывода*). Все выводные элементы данных записываются с различной шириной полей и разделяются при помощи делителя.

Поэтому, GAMS игнорирует все глобальные и локальные спецификации (описания) форматирования для ширины поля и выравнивание. Необходимо помнить, что количество десятичных знаков в числе для численных элементов данных могут быть все равно обозначены с помощью файлового суффикса **.nd**. Каждый элемент данных записывается сразу после предшествующего делителя на той же самой линии, до тех пор, пока курсор не будет сброшен. Избегайте в программе, которая создает раздельный файл, горизонтальное перемещение курсора. Горизонтальное перемещение курсора в разделенном файле является потенциальным источником будущих проблем, так как разделение может быть стертым.

Несмотря на то, что запятая является в большинстве случаев разделяющим знаком для электронных таблиц, другие разделяющие знаки, такие как пробел и символ табуляции, могут быть также использованы.

14.17.1 ПРИМЕР

В следующем примере подтаблица **capacity** (*объем*) программы [MEXSS] подготовлена как разделенный файл. Следующий фрагмент программы демонстрирует использование суффикса **.pc** опция **5**. Фрагмент программы мог быть помещен в конце модели [MEXSS].

```
file out; put out; out.pc =5;
put 'capacity (metric tons)';
loop(i, put i.tl);
loop(m,
```

```
        put / m.te(m);
        loop(i,  put k(m,i));
    );
```

Первая строка фрагмента программы превращает файл **out.put** в разделенный файл. Обратите внимание, что в оставшейся части этой программы ширина файла, выравнивание и горизонтальное перемещение курсора полностью удаляются. Все текстовые элементы данных берутся в кавычки. Следующее является результирующим выводным файлом.

```
"CAPACITY (tons)", "AHMSA", "FUNDIDORA", SICARTSA","HYLSA","HYLSAP"
"BLAST FURNACES", 3.25,1.40,1.10,0.00,0.00
"OPEN HEARTH FURNACES",1.50,0.85,0.00,0.00,0.00
"BASIC OXYGEN CONVERTERS",2.07,1.50,1.30 0.00,0.00
"DIRECT REDUCTION UNITS",0.00,0.00,0.00,0.98,1.00
"ELECTRIC ARC FURNACES",0.00,0.00,0.00,1.13,0.56
```

Помните, что все элементы данных разделяются запятой, а все текстовые выводные данные берутся в двойные кавычки.

УСЛОВНЫЕ ВЫРАЖЕНИЯ,

ПРИСВОЕНИЯ И УРАВНЕНИЯ 15

15.1. ВВЕДЕНИЕ

Этот раздел посвящен вопросам работы в GAMS с условными выражениями, присвоениями и уравнениями. Для полного освоения данных вопросов необходимо владеть информацией об индексных операциях, описанных ранее. Примером условной задачи может быть тяжеловесный грузовик, который не может быть использован на отдельных маршрутах, потому что не допускается передвижение по
легким мостам или какие-то другие ограничения не позволяют осуществлять передвижение продукции. Ранее уже было показано, как, используя подмножества в индексных выражениях, выполнять некоторые расчеты, когда имеются исключения такого рода.

15.2. ЛОГИЧЕСКИЕ УСЛОВИЯ

Логические условия - особые выражения, которые оценивают, является ли значение **True** (*истинным*) или **False** (*ложным*). Численные выражения также могут служить в качестве логических условий. В GAMS предусмотрены численные связи и логические операторы, которые могут быть использованы, чтобы воспроизводить логические условия. В следующих четырех подразделах описаны различные способы построения блоков, которые используются для создания комплексных логических условий.

Логические условия, выраженные в этом разделе действительны только для **sets** (*множеств*) и **parameters** (*параметров*). Трактовка логических условий, включающих переменные, будет рассмотрена отдельно в следующем разделе.

15.2.1. ЧИСЛЕННЫЕ ВЫРАЖЕНИЯ КАК ЛОГИЧЕСКИЕ УСЛОВИЯ.

Численные выражения могут также служить в качестве логических условий - результат ноль трактуется как логическое значение **False** (*ложь*), ненулевой результат трактуется как логическое значение **True** (*истина*).

Следующее численное выражение может быть использовано для иллюстрации вышесказанного

```
2*a - 4
```

Когда **a** равно **2** , то результат этого выражения в логическом значении является **False** (*ложь*), потому что выражение принимает нулевое значение. Во всех остальных случаях выражение является **True** (*истина*), потому что результат не нулевой.

15.2.2. ОПЕРАТОРЫ ЧИСЛЕННЫХ СВЯЗЕЙ

Операторы численных связей сравнивают два численных выражения. Все операторы численных связей представлены ниже:

Операторы	Описание
lt, <	меньше чем
le, <=	меньше или равно
eq, =	равно
ne, <>	не равно
ge, >=	больше или равно
gt, >	больше чем

Следующий пример иллюстрирует, каким образом численные связи используются в качестве логических условий

(sqr(a) > a)

Это условие превращается в **False** (*ложь*), если **-1 <= a <=1**. Для всех других значений **a** это условие превращается в **True** (*истину*). Обратите внимание, что это же выражение может быть записано в виде

(sqr(a) gt a)

15.2.3. ЛОГИЧЕСКИЕ ОПЕРАТОРЫ

Существующие в GAMS логические операторы представлены ниже

Операторы	Описание
Not	нет
And	и
Or	или включающий в себя
xor	или исключающий

Таблица, предложенная ниже, отражает результаты использования логических операторов.

Операнды		Результаты			
a	b	a and b	a or b	a xor b	not a
0	0	0	0	0	1
0	не ноль	0	1	1	1
не ноль	0	0	1	1	0
не ноль	не ноль	1	1	0	0

15.2.4. ЧЛЕНСТВО ВО МНОЖЕСТВЕ

Членство во множестве может быть также использовано в качестве логического условия. Метка дает в результате логическое значение **True** (*истина*), если она имеется во множестве, и **False** (*ложь*) - если ее нет во множестве. Это используется при работе с **subsets** (*подмножествами*) и **dynamic sets** (*динамическими множествами*) (которые были рассмотрены в разделе 11).

Для иллюстрации рассмотрим следующий пример

```
set  i  /1*10/
     subi(i) /1*3/;
```

В логическом значении множество **subi(i)** приводит к результату **True** *(истина)* для всех элементов множества **i**, которые принадлежат к множеству **subi**, и к результату **False** (*ложь*) для всех элементов **i**, которые не принадлежат к множеству **subi**.

Использование членства во множестве в качестве логических условий является крайне эффективным свойством GAMS. Иллюстрация того, как используется это свойство, будет приведена в этом разделе в дальнейшем. Полностью понять всю мощь этого свойства можно будет только поняв особенности использования в качестве логического условия членства в **dynamic sets** (*динамических множествах*).

15.2.5. ЧИСЛЕННЫЕ ЗНАЧЕНИЯ ЛОГИЧЕСКИХ УСЛОВИЙ

Предыдущие четыре подраздела рассматривали различные свойства GAMS, которые могут использоваться в логических условиях. Однако GAMS не имеет **Boolean** (*логических*) типов данных.

GAMS придерживается соглашения, что результат операций сравнения (реляционных операций) есть ноль, если утверждение есть **False** (*ложь*), и результат сравнения (реляционных операций) есть единица, если утверждение - **True** (*истина*).

Рассмотрим следующий пример:

```
x  = (1 < 2) + (2 < 3)
```

Выражение по отношению к правому присвоению имеет численное выражение 2, так как оба логических условия в скобках имеют значение **True** (*истина*), однако **x** принимает значение **1**. Обратите внимание, что это отличается от присвоения, показанного ниже

```
x = (1 < 2)  or  (2 < 3)
```

который принимает численное значение **1**, обусловленный оператором **or**, который ведет себя в соответствии с таблицей в разделе 15.2.3.

15.2.6. СМЕШАННЫЕ ЛОГИЧЕСКИЕ УСЛОВИЯ - ОПЕРАТОР ПРИОРИТЕТА ОПЕРАЦИЙ

Составление блоков, рассмотренное в первых четырех подразделах, может быть использовано для создания более сложных логических условий.

GAMS в логических условиях по умолчанию при отсутствии скобок использует следующую последовательность старшинства, как показано ниже (в уменьшающемся порядке).

Операции	**Операторы**
Экспоненцирование	**
Численные операторы	
умножение, деление	, /
знак одноместной операции - плюс, минус	+, -
знак бинарной операции - сложение, вычитание	+, -
Численные операторы связи	<, <=, =, <>, >=, >
Логические операторы	
Not	not
And	or

Or, xor	xor

В случае, когда имеются операторы с одинаковым старшинством, последовательность, в качестве критерия старшинства используется последовательность, в которой операторы появляются в выражении, старшинство уменьшается слева на право.

Для указания старшинства в последовательности операций желательно всегда использовать скобки. Это позволит исключить ошибки и сделает замысел выражения понятным.

Рассмотрим следующий пример:

x - 5*y and z - 5

Эквивалентом вышеприведенной записи будет выражение

(x - (5*y)) and (z-5)

Однако использование скобок сделало выражение более прозрачным для понимания.

15.2.7. СМЕШАННЫЕ ЛОГИЧЕСКИЕ УСЛОВИЯ - ПРИМЕРЫ

Некоторые простые примеры логических условий, содержащие в себе составные блоки и рассмотренные в предыдущих подразделах, показаны ниже в качестве примеров составления и использования более комплексных логических условий

Логическое условие	**Численное значение**	**Логическое значение**
(1<2) + (3<4)	2	True (*Истина*)
2<1) and (3<4)	0	False (*Ложь*)
(4*5 -3) + (10/8)	17,125	True (*Истина*)
(4*5 - 3) or (10-8)	1	True (*Истина*)
(4 and 5) + (2*3<=6)	2	True (*Истина*)
4 and 0) + (2*3<6)	0	False (*Ложь*)

15.3. ДОЛЛАРОВОЕ УСЛОВИЕ

Этот раздел знакомит с долларовым оператором, который является одним из самых мощных свойств GAMS. Долларовый оператор выполняет операции с логическими условиями. Терм **$ (condition** *условие*) может быть прочитан как "такое, что условие является действительным", где условие является логическим условием, как это было определено в разделе 15.1.

Долларовые логические условия не могут содержать в себе переменные. Однако атрибуты переменных (такие как **.l** и **.m**) допускаются.

Долларовый оператор используется в модели для условных присвоений, выражений и **equations** (*уравнений*). В следующих разделах будут отдельно рассмотрены правила использования долларовых условий в условных присвоениях, выражениях и уравнениях.

15.3.1. ПРИМЕР

Рассмотрим простое условие:

if(b>1.5), then a = 2

В GAMS это условие может быть смоделировано с помощью долларового условия

a$(b > 1.5) = 2;

Если условие не удовлетворяется, присвоение не выполняется. GAMS "читает" символ **$** как "тогда, когда", чтобы пояснить значение: "**a** равно **2** тогда, когда **b** является больше, чем **1.5** ".

15.3.2. ВЛОЖЕННЫЕ ДОЛЛАРОВЫЕ УСЛОВИЯ

Долларовые условия могут быть вложенными. Терм

$(condition1$(condition2))

может читаться как

$(condition1 and condition2)

Для вложенных долларовых условий все последующие выражения после знака доллар "**$**" должны быть заключены в скобки.

Рассмотрим следующий пример

u(k)$(s(k)$t(k)) = a(i);

k, s(k), t(k) и **i** являются **sets** (*множествами*), в то время как **u(k)** и **a(i)** являются **parameters** (*параметрами*). Присвоения будут выполнены только для тех членов **k**, которые также являются членами и **s**, и **t**. Необходимо учитывать положение скобок в долларовых условиях. Вышеприведенный оператор может быть переписан как

u(k)$(s(k) and t(k)) = a(k);

Для большей удобочитаемости операторов, настоятельно рекомендуется вместо вложенных долларовых операторов использовать логический оператор **and**.

15.4. УСЛОВНЫЕ ПРИСВОЕНИЯ

Оператор, взятый из примера в разделе 15.2.1, представляет собой условное присвоение. В этом примере долларовое условие расположено на левой стороне присвоения. Эффект от долларового условия в значительной степени зависит от того, на какой стороне присвоения он расположен.

В любом случае, чтобы описать присвоение, можно использовать каждое из двух форм долларового условия. И в каждом случае критерием выбора должна быть ясность логики.

В следующих подразделах описывается использование долларового условия с каждой стороны присвоения.

15.4.1. ДОЛЛАР СЛЕВА

В примере из раздела 15.2.1. долларовое условие помещено с левой стороны оператора присвоения.

Для оператора присвоения с долларовым условием с левой стороны если не удовлетворены логические условия, то присвоение не выполняется. Это означает, что предыдущее содержание **parameter** (*параметра*) слева останутся неизмененными для меток, которые не удовлетворяют условие.

Если **parameter** (*параметр*) с левой стороны присвоения раньше не инициализировался или ему не присваивалось какое либо значение, то любой метке, для которой было пресечено присвоение из-за долларового условия с лева, будут присвоены нулевые значения.

Рассмотрим следующий пример из [CHEMISTRY]

```
rho(i)$(sig(i) ne 0) = (1./sig(i)) -1.;
```

Условимся, что **parameter** (*параметр*) **sig(i)** ранее был определен. Используем вышеприведенный оператор, чтобы рассчитать **rho(i)**. Долларовое условие в операторе защищает выражение от недопустимой операции деления на ноль. Если какое либо значение, ассоциируемое с **sig(i)**, окажется равным нулю, присвоение не выполнится, и для **rho(i)** останется действительным предыдущее значение. Как это происходит: **rho(i)** раньше не было инициализировано, и поэтому все метки, для которых **sig(i)** является **0**, дадут в результате значение **0**.

Теперь отменим условие, описанное в разделе 15.2.1, которое говорит, что не ноль предполагает **True** (*истина*), а ноль предполагает **False** (*ложь*).

Тогда вышеприведенное присвоение должно быть записано следующим образом

```
rho(i)$sig(i)  =  (1./sig(i)) -1.;
```

15.4.2. ДОЛЛАР СПРАВА

Для оператора присвоения с долларовым условием с правой стороны присвоение выполняется всегда. Если логическое условие не выполняется, соответствующий терм, над которым логическое долларовое условие выполняет действие, превращается в 0.

Рассмотрим следующий пример, который является незначительной модификацией, примера, описанного в разделе 15.2.1.

```
x =  2$(y > 1.5);
```

Выраженное в словах, это условие является эквивалентом нижеприведенной записи

```
if(y > 1.5)  then (x=2),  else (x=0)
```

Поэтому подразумевается тип конструкции **if-then-else** (*если-тогда-иначе*), но операция **else** (*иначе*) является встроенной и никогда не делается явной. Обратите внимание, что оператор в вышеприведенном примере может быть переписан с определенным явным **if-then-else** (*если-тогда-иначе*) и эквивалентная

запись выгладит следующим образом

```
x = 2$(y  gt  1.5) + 0$(y  le  1.5);
```

Использование этого способа более очевидно в случае, когда условие **else** (*иначе*) необходимо сделать явным. Рассмотрим пример из [FERTD]. **Set** (*множество)* **i** является множеством растений и используется в качестве контролирующего индекса при расчете **mur(i)** - стоимости транспорта импортируемого сырья. В задаче рассматриваются два способа перевозки сырья. В некоторых случаях баржевая перевозка должна проходить по дороге, потому что местоположение растений находится от реки, поэтому необходимо комбинировать отдельные стоимости перевозки. Присвоение выполнено следующим образом

```
mur(i) =   (1.0 + . 0030*ied(i, 'barge'))$ied(i, 'barge') +
           (0.5 + .0144*ied(i, 'road'))$ied(i, 'road');
```

Эта запись означает, что если элемент в таблице расстояний имеет не нулевое значение, тогда стоимость перевозки грузов, терм которой включает в себя постоянные и переменные компоненты, прибавляется к общей стоимости. Если в таблице расстояний нет элемента расстояния, тогда нет вклада транспортных расходов по автодороге в общую стоимость, и этот режим не используется.

15.5. УСЛОВНЫЕ ИНДЕКСНЫЕ ОПЕРАЦИИ

Другим важным использованием долларового условия является контроль домена операции индексных операций. Схематически это похоже на "доллар слева", описанный в разделе 15.4.1.

Рассмотрим следующий пример из [GTM]

```
tsubc  =  sum(i$(supc(i)  ne  inf),  supc(i));
```

Этот оператор вычисляет сумму значений, имеющих предел, в **supc**.

Обычно использование долларовых контролируемых индексных операций имеет место тогда, когда контролем является само множество. Важность этого понятия станет ясной при обсуждении динамического множества в разделе 11.

В разделе 4 использовалось множество, чтобы определить связь между источниками продукции и портами, куда продукция должна быть доставлена.

Другим типичным примером является соотношение **set-to-set** (*множество-ко-множесту*), определяющее связь между штатами и областями, используемое для агрегирующих данных, полученных штатом в соответствии с требованиями модели (через область).

```
sets  r  / west, east /
      s  / florida, texas, vermont, maine/
      corr(r,s)  /north.(vermont, maine)
                  south.(florida, texas)/;

parameter  y(r)
           income (s)  "income of each state"
                  / florida  4.5,  vermont  4.2
                    texas    6.4,  maine    4.1/;
```

Set (*множество*) **corr** обеспечивает соответствие, чтобы показать какие штаты к каким областям относятся. **Parameter** (*параметр*) **income** является

доходом каждого штата. **y(r)** может быть рассчитан с помощью оператора присвоения

```
y(r) = sum(s$corr(r,s), income(s));
```

Для области **r**, суммирование через **s** возможно только через их пары **(r,s)**, для которых существует **corr(r,s)**. Схематично существование множества является аналогией логического значения **true** (*истина*) или арифметического значения "**не ноль**". Результат этого присвоения следующий: сумма для "**north**" включает в себя только доходы "**vermont**" и "**maine**", а сумма для "**south**" включает в себя только доходы "**texas**" и "**florida**".

Вышеприведенное суммирование может быть также записано в виде

```
sum(s, income(s)$corr(r,s))
```

но эта форма записи трудно для прочтения, каким образом выполняется суммирование в соответствии с контролирующими индексами.

15.6. УСЛОВНЫЕ УРАВНЕНИЯ

Долларовый оператор также используется для исключения управления в уравнениях. В следующих двух подразделах обсуждаются два основных вида использования долларовых операторов в уравнениях - в теле уравнения и через область определения.

15.6.1. ДОЛЛАРОВЫЕ ОПЕРАТОРЫ В ТЕЛЕ УРАВНЕНИЯ

Долларовые операторы в уравнении аналогичны долларовому контролю справа присвоения, как это было показано в разделе 15.3.2; аналогия будет даже ближе, если представить "справа" как значение справа '..'. Операция **if-else** (*если-иначе*) предполагается такой же, как это было описано для присвоения. Она используется, чтобы исключить части определений из некоторых воспроизводимых ограничений.

Рассмотрим следующий пример из [CHENERY]

```
mb(i).. x(i) =g= y(i) + (e(i) - m(i))$t(i);
```

Терм присоединяется к правой стороне уравнения только для тех элементов **i**, которые относятся к **t(i).**

Используя долларовое условие, контролирующие индексирующие операции могут выполняться также, как с любым присвоением.

Рассмотрим следующее уравнение баланса запасов **(sb)** из [GTM]

```
sb(i).. sum(j$ij(i,j), x(i,j)) =l= s(i);
```

15.6.2. ДОЛЛАРОВЫЙ КОНТРОЛЬ ЧЕРЕЗ ОБЛАСТЬ ОПРЕДЕЛЕНИЯ

Долларовый контроль через область определения аналогичен долларовому контролю слева присвоения, как описывалось в разделе 15.3.3, аналогия будет даже ближе, если условиться, что "слева" означает слева от '..'.

Целью долларового контроля через область определения уравнения является ограничить количество воспроизводимых ограничений таким образом, чтобы оно было меньше, чем число ограничений, которые воспроизводятся через домен, определяющего **set** (*множества*).

Рассмотрим следующий пример из [FERTS]

```
cc(m,i)$mpos(m,i)..
sum(p$ppos(p,i),  b(m,p) * z(p,i)) =l=  util*k(m,i);
```

cc - вся совокупность ограничения, определенного для всех элементов (**m)** и положений **(i).** Однако не все типы элементов существуют во всех положениях, и чтобы ограничить число действительно воспроизводимых ограничений, используется преобразование **set** (*множества*) **mpos(m,i).** Контроль суммирования
через **p** с **ppos(p,i)** является дополнительным и он необходим, потому что не все процессы **(p)** возможны во всех положениях **(i).**

СРЕДСТВА

УПРАВЛЕНИЯ ПОТОКОМ ДАННЫХ 16

16.1. ВВЕДЕНИЕ

Предыдущие разделы фокусировались на возможностях GAMS описать модель. В этом разделе будут описаны имеющиеся в распоряжении GAMS средства управления потоком данных. Ниже приведены операторы, с помощью который это можно выполнить

Loop - оператор
If-else - оператор
For - оператор
While - оператор

Каждый из этих операторов будет детально обсужден в следующих подразделах.

16.2. ОПЕРАТОР LOOP

Оператор **loop** предусмотрен для случаев, когда параллельного присвоения недостаточно. Наиболее часто оператор **loop** используется при отсутствии аналитических связей, например, между значениями для выполнения присвоения **parameter** (*параметру*). Оператор **loop** успешно применяется при работе с циклическими множествами обычного программирования, например, при создании документов с использованием оператора **put**.

16.2.1. СИНТАКСИС

Синтаксис оператора **loop**

```
loop(controlling_domain      [$(condition)]
     контролирующий домен          условие
     statement  {; statement }
     оператор        оператор
);
```

Если **controlling_domain** содержит более чем одно **set** (*множество*), то вокруг них требуются скобки.

Оператор **loop** вызывает GAMS, чтобы выполнить операторы в пределах области действия цикла по очереди для каждого члена управляющего **set(s)** (*множества или множеств*). Последовательность вычислений - это последовательность, в которой был выполнен ввод меток. Таким образом, **loop** (*цикл*) является другим, более общим, типом индексных операций. Set (множество), которое входит в состав **controling_domaim** в операторе **loop**, не должно быть статическим или вложенным и может быть контролируемым долларом **'$'**. **Loops** (*циклы*) могут быть контролируемы более чем одним множеством.

Внутри оператора **loop** нельзя выполнять декларацию или определение уравнений.

Внутри тела **loop** (*цикла*) недопустимо видоизменять какое-либо контролирующее множество.

16.2.2. ПРИМЕР

Рассмотрим гипотетический пример, в котором рассматривается эмпирический рост цен.

```
set  t  /1985*1990/

parameter  pop(t)     /1985   3456/
           growth(t)  /1985  25.3, 1986  27.3,  1987  26.2
                       1988  27.1, 1989  26.6,  1990  26.6/;
```

Совокупные цены рассчитываются с помощью оператора **loop**

```
loop(t, pop(t+1) = pop(t) + growth(t));
```

итерационным (повторяющимся) способом, который здесь предпочтительнее, чем параллельный. В этом примере имеется один оператор в зоне влияния **loop** (*цикла*) и одно управляющее (или контролирующее) **set** (*множество*).

Loop (*цикл*) часто используют для выполнения повторяющихся расчетов.

Рассмотрим следующий пример, где для нахождения квадратного корня используется метод Ньютона. Этот пример чисто иллюстративный. В практике необходимо использовать функцию **sqrt.** Метод Ньютона - это утверждение, что если **x** - приближенное к квадратному корню от **v**, то **(x+v/x) > 1**

```
set i  "set to drive iteration" /i-1*i-100/;
parameret  value(i)  "used to hold successive approximations";

scalars
 target   "number whose square root is needed"  /23.456/
 sqrtval  "final approximation to sqrt(target)"
 curacc   "accuracy of current approximatoon"
 reltol   "required relative accuracy"   /1.0e-06/;

abort$(target <= 0)  "argument to newton must be positive", target;
value("i-1")  =  target/2 ;  curacc   =  1;
loop(i$(curacc > reltol),
    value(i+1) = 0.5*(value(i) + target/value(i));
    sqrtval = value(i+1);
    curacc  = abs (value(i+1) - value(i))/(1+abs(value(i+1)))
    );

    abort$(curacc > reltol)  "aquare root not found"
    option decimals = 8;
    display "square root found within tolerance", sqrtval, value;
```

Выводом является

```
----  21  sguare root found within tolerance
----  21  PARAMETER SQRTVAL   =  4.84313948 final approximation
                                            to sqrt(target)

----  21  PARAMETER VALUE  used to hold successive approximations
```

```
I-1  11.7280000, I-2  6.8640000, I-3  5.1406247, I-4  4.8517471
I-5   4.8431471, I-6  4.8431394, I-7  4.8431394
```

16.3. ОПЕРАТОР IF-ELSEIF-ELSE

Оператор **if-else** полезен при выполнении условного перехода от одних операторов другим. В некоторых случаях условный переход может быть записан как ряд долларовых условий, но с помощью оператора **if** это делать более удобно. Необязательная часть **else** позволяет формулировать традиционную конструкцию **if-then-else.**

16.3.1. СИНТАКСИС

Синтаксис для оператора **if-elseif-else**

```
if (condition ,
    условие
    statements ;
    операторы
{ elseif condition, statement }
          условие       оператор
[ else statements ;]
       операторы
);
```

где **condition** (*условие*) - логическое условие, описаное в разделе 15.

Нельзя выполнять декларацию или определение уравнения внутри оператора **if.**

16.3.2. ПРИМЕР

Рассмотрим следующие операторы

```
p(i)$(f <= 0) = -1;
p(i)$(f >  0)  and (f < 1)) = p(i)**2;
p(i)$(f >  1) = p(i)**3;

q(j)$(f <= 0) = -1;
q(j)$(f >  0)  and (f < 1)) = q(j)**2;
q(j)$(f >  1) = q(j)**3;
```

Эти же условия могут быть выражены с помощью операторов **if-elseif-else**

```
if (f <= 0,
   p(i) = -1;
   q(j) = -1;
elseif ((f > 0) and (f < 1)),

   p(i) = p(i)**2;
   q(j) = q(j)**2;
```

```
else

   p(i) = p(i)**3;
   q(j) = q(j)**3;
   );
```

Тело оператора **if** может содержать в себе операторы **solve.**
Например, рассмотрим следующий бит операторов GAMS:

```
if  (ml.modelstat eq 4),
*     model ml  was  infeasible
*     relax bounds on x and solve again
   x.up(j) = 2*x.up(j) ;
   solve ml  using lp  minimizing  lp;

else
     if (ml.modelstat  ne  1),
         abort "error solving model ml"  ;
      );
      );
```

Внутри оператора **if** нельзя выполнять определение уравнения, поэтому следующий блок операторов в GAMS недопустим

```
if (s  gt  0,
    eq..  sum(i, x(i))  =g= 2;
    );
```

Внутри оператора if нельзя выполнять декларацию, поэтому следующий блок операторов в GAMS недопустим

```
if (s  gt  0,
   scalar  y;  y = 5;
    );
```

16.4. ОПЕРАТОР WHILE

Оператор while служит для того, чтобы зациклить через блок операторов.

16.4.1. СИНТАКСИС

Синтаксис оператора **while** следующий

```
while  (condition,
        условие
        statements;
        операторы
        );
```

Внутри оператора **while** нельзя выполнять декларацию и определение уравнений.

16.4.2. ПРИМЕРЫ

Оператор **while** можно использовать для контроля оператора **solve**. Например, рассмотрим следующий бит операторов GAMS, который выполняет случайный поиск для глобального оптимума невыпуклой модели:

```
scalar  count  ;  count  =1;
scalar  globmin ;  globmin  = inf;
option  bratio = 1;
while ((count  le  1000),
       x.l(j) = uniform(0,1)  ;
       solve ml  using  lp  minimizing  obj;
       if  (obj.l  le  globmin,
           globmin = obj.l;
           globinit(j)  = x.l(j);
         );
       count = count +1;
         );
```

В этом примере невыпуклая модель решается из 1000 случайных начальных точек, и прослеживается глобальное решение.

Модель [PRIME] из библиотеки моделей иллюстрирует использование оператора **while** через пример, где воспроизводится множество простых чисел меньших, чем 200.

Внутри оператора while нельзя выполнять определение уравнения, поэтому следующая запись операторов в GAMS недопустима

```
while  (s  gt  0,
                  eq..  sum(i, x(i))  =g= 2;
       );
```

Внутри оператора while нельзя выполнять декларацию, поэтому следующая запись в GAMS недопустима

```
while  (s  gt  0,
                  scalar  y;  y = 5;
        );
```

16.5. ОПЕРАТОР FOR

Оператор for служит для того, чтобы зациклить блок операторов.

16.5.1. СИНТАКСИС

Синтаксис оператора **for** следующий

```
for  (i = start     to|downto      end [by  incr]
          начало       к  вплоть до    конца
             statements;
             операторы
        );
```

где **i** - не **set** (*множество*), а **parameter** (*параметер*);
incr - прирост (или шаг), на который изменяется **i** после каждого выполнения цикла.

Внутри оператора **for** нельзя выполнять декларацию или определение.

Start (*начальное*), **end** (*конечное*) и **incr** (*прирост или шаг*) значения не обязательно должны быть целочисленными. **Start** (*начальное*) и **end** (*конечное*) значения могут быть положительными или отрицательными действительными числами. **incr** (*прирост или шаг*) должен быть положительным действительным числом.

16.5.2. ПРИМЕР

Оператор **for** можно использовать для контроля оператора **solve**. Например, рассмотрим следующий бит операторов GAMS, который выполняет случайный поиск для глобального оптимума невыпуклой модели

```
scalar i;
scalar globmin ; globmin = inf;
option  bratio = 1;
for (i = 1 to 1000,
    x.l(j)  =  uniform(0,1)  ;
    solve ml  using  nlp  minimizing  obj ;
    if  (obj.l  le globmin,
         globmin = obj.l ;
         globinit(j)  =  x.l(j)  ;
    );
    );
```

В этом примере невыпуклая модель решается из 1000 случайных начальных точек, и прослеживается глобальное решение.

Использование действительных чисел в качестве **start** (*начальных*) и **end** (*конечных*) значений и для **incr** (*прироста или шага*) может быть понято из следующего примера.

```
for  (s  =  -3.4  to  0.3  by 1.4,
     display s ;
     );
```

Результирующая распечатка файла будет содержать следующие строки:

```
----   2 PARAMETER  S         =  -3.400
----   2 PARAMETER  S         =  -2.000
----   2 PARAMETER  S         =  -0.600
```

Обратите внимание, что с каждым выполнением цикла значение **S** увеличивалось на 1.4 до тех пор, пока оно не стало превышать значение 0.3.

Внутри оператора **for** нельзя выполнять определение уравнения, поэтому следующая запись в GAMS недопустима.

```
for    (s = 1 to 5 by 1,
       eq..  sum(i, x(i))  =g= 2;
       );
```

Внутри оператора for нельзя выполнять декларацию, поэтому следующая запись в GAMS недопустима.

```
for    (s = 1 to 5 by 1,
       scalar  y;  y = 5;
       );
```