В. Б. Ильин, О. П. Желенкова

СТРУКТУРЫ ДАННЫХ И ПРОГРАММИРОВАНИЕ В MIDAS

Учебное пособие
для студентов астрономических отделений
университетов

Санкт-Петербург
BBM
2006

И46

Печатается по постановлению
Редакционно-издательского совета
Санкт-Петербургского государственного университета

Р е ц е н з е н т ы :

проф. **В. П. Решетников**, доц. **В. А. Яковлева**

Ильин В. Б., Желенкова О. П.

И46 Структуры данных и программирование в MIDAS: Учебное пособие для студентов астрономических отделений университетов. — СПб.: BBM, 2006. — 000 с.

ISBN 5-9651-0241-?

Система MIDAS (Munich Image Data Analysis System), разработанная в Европейской южной обсерватории (ESO), обладает широкими возможностями и является одной из наиболее часто используемых систем для обработки и анализа наблюдательных данных в астрономии. В пособии детально рассматриваются различные аспекты программирования и работы со структурами данных в системе MIDAS. Программирование — написание процедур (программ) присутствует в том или ином виде во всех разделах пособия, начиная с первого. Структуры данных последовательно объясняются и связываются с типичными задачами, для решения которых они привлекаются. Подробно обсуждаются запуск и настройка системы MIDAS, ее язык процедур, справочная система, таблицы и их аппроксимация, форматы ввода/вывода, включение программ на Фортране/Си и визуализация данных. Излагаемый материал служит основой специального практикума, проводимого на астрономическом отделении СПбГУ для студентов-астрономов 4-го курса (один семестр, 4 часа в неделю) в течение последних нескольких лет. В приложениях даны необходимые для работы с MIDAS сведения об операционной системе Linux, систематический обзор команд MIDAS, описание весьма полезного Python-интерфейса в MIDAS и мини-глоссарий. Таким образом, пособие может также служить справочником при работе с MIDAS пользователей разного уровня.

Учебное пособие издается при финансовой поддержке Федерального агентства по науке и инновациям РФ (грант РПН 2.1.1.2852).
В.И. благодарит за частичную поддержку грант НШ-8542.2005.2.

ISBN 5-9651-0241-?

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ

Основную информацию об астрономических объектах мы получаем при их наблюдениях на разных длинах волн с использованием разнообразных телескопов и приемников. Поэтому крайне важной является задача обработки, визуализации и анализа наблюдательных данных. Существует большое количество программных средств, предназначенных для решения этой задач, однако чаще всего используют несколько универсальных систем редукции астрономических данных. Такие системы включают широкий набор команд, множество пакетов для специализированной обработки и встроенный командный язык. Последний позволяет работать с системами в интерактивном и пакетном режимах, писать программы, приспосабливающие возможности систем для обработки наблюдательных данных пользователя. Предлагаемый набор команд и функций обычно позволяет выполнять все основные этапы обработки данных: исправление дефектных пикселей и инструментальных ошибок, калибровку данных, классификацию и поиск объектов, анализ их параметров и т. п. Как правило, в таких системах имеются специальные возможности, учитывающие особенности обработки данных, получаемых в разных спектральных диапазонах на различных инструментах.

Универсальных, ставших уже стандартными, систем обработки астрономических данных немного. К ним можно отнести *MIDAS*, *IRAF*, *AIPS* и с некоторыми допущениями *STARLINK* с *IDL*.

1. Система MIDAS — Munich Image Data Analysis System

Эта система создана и поддерживается в Европейской Южной Обсерватории (European Southern Observatory, ESO). MIDAS начали разрабатывать в 1979 году как открытую систему обработки данных, в которую можно встраивать дополнительные программы, обрабатывающие информацию, получаемую на новых астрономических инструментах. С 1982 года MIDAS является стандартной системой в ESO. На ее создание и отладку ушло примерно 300 человеко-лет. Имеется обширная документация, описывающая возможности MIDAS. Система содержит около 400 базовых команд и большое количество пакетов для обработки и анализа данных (еще дополнительно порядка 300 команд).

Система MIDAS позволяет:

- работать в интерактивном режиме;
- объединять команды в исполняемые командные файлы (пакетный режим);
- работать с разными по структуре данными: изображениями, таблицами, дескрипторами, ключевыми словами, каталогами, fit-файлами;
- осуществлять ввод/вывод данных в разных форматах;
- выводить графические файлы и изображения на различные внешние устройства;
- пользоваться развитой системой подсказок с детальным описанием всех команд и их параметров.

Базовый набор функций MIDAS для работы с данными включает:

- просмотр изображений в разных цветовых режимах;
- обширный набор команд по визуализации изображений (увеличение/уменьшение, прокрутка и т. п.);
- разнообразные преобразования изображений (вращение, масштабирование, фильтрация, Фурье-преобразование и т. д.);
- выполнение арифметических операций и статистики над данными разных типов;
- работу с таблицами (сортировка, поиск, объединение, операции со столбцами);
- извлечение участков изображений, генерацию тестовых фреймов;
- построение спектров, контуров, гистограмм, перспектив и т. п.

В системе можно выделить следующие основные части (см. также *Приложение Б*):

- базовые команды: ввод/вывод данных, работа с внешними устройствами, с изображениями, таблицами, спектрами, масками, MIDAS-каталогами, фильтрация и различные преобразования, статистический анализ, визуализация изображений и т. д.;
- набор дополнительных команд;
- пакеты (контексты) для редукции и анализа обработанных данных.

Командный язык MIDAS достаточно развит. Он включает операторы цикла и условных переходов, определения локальных/глобальных переменных, возможность вызывать подпрограммы и передавать значения параметров при их вызове и т. п. Все это позволяет легко писать собственные сценарии обработки данных. Для расширения возможностей системы разрешено создание дополнительных программных модулей на языках Фортран 77 и Си с использованием средств MIDAS. Более детальную информацию о системе можно найти на ее сайте *http://www.eso.org/projects/esomidas/*.

2. Система IRAF — Image Reduction and Analysis Facility

Система создана и поддерживается специальной группой в Национальной Оптической Астрономической Обсерватории (National Optical Astronomy Observatory, NOAO, Аризона, США). В базовый состав системы входит большой набор программ для обработки, визуализации и анализа данных в оптическом и инфракрасном диапазонах. Ядро системы включает инструментарий для чтения/записи данных в FITS (Flexible Image Transport System) формате, интерактивные программы для визуализации изображений, средства для обнаружения и исправления технических дефектов цифровых приемников, разнообразные фильтры для сглаживания данных, средства для выполнения арифметических операций над изображениями, их комбинирования, статистического анализа. Задачи или команды системы позволяют работать с ней в интерактивном и пакетном режимах. В систему могут быть включены дополнительные пакеты, расширяющие функциональные возможности ядра, например:

- FTOOLS — набор утилит для создания, проверки и модификации данных в FITS-формате;

- CRUTIL — удаление дефектов, вызванных космическими частицами;
- FINDER — определение точных координат объектов на ПЗС-изображениях с использованием в качестве опорного каталога Guide Star Catalogue;
- FOCAS — классификация и анализ слабых источников, используя каталоги объектов, созданные на основе цифровых изображений;
- DIGIPHOTX — фотометрия на основе алгоритмов DAOPHOT;
- COLOR — создание композитных RGB-изображений;
- ESOWFI — составление мозаичных изображений;
- MSCRED — редукция мозаичных ПЗС-изображений;
- IMMATCHX — совмещение изображений;
- ICE — контроль накопления данных с ПЗС-матрицы в оболочке системы IRAF;
- GMISC — пакет редукции для данных, получаемых с Gemini;
- STSDAS — калибровка и анализ данных, получаемых с Hubble Space Telescope;
- EUV — анализ данных, получаемых с Extreme Ultraviolet Explorer;
- XRAY — анализ и редукция рентгеновских данных.

Кроме этого, имеется богатый набор средств для спектральной редукции и анализа, который расширяет базовые функции для одно- и двухмерной, эшелле- и оптоволоконной спектроскопии.

Система IRAF работает с несколькими вариантами систем UNIX, Solaris. Для обучения работе с системой ее разработчики подготовили вводный курс и упражнения для разной методики обработки фотометрических и спектральных данных. Подробнее ознакомиться с системой можно по адресу *http://iraf.noao.edu* .

3. Система AIPS — Astronomical Image Processing System

Разработка этой системы была начата в 1978 году в Национальной Радиоастрономической Обсерватории (National Radio Astronomy Observatory, NRAO, США). Она предназначена для калибровки, обработки и анализа радиоинтерферометрических данных. В системе более 400 000 строк документации и справочной

(on-line) информации, более миллиона строк в 2300 программах на языках Фортран и Си, а также более 350 команд. Система имеет средства для визуализации и анализа двух- и трехмерных радиоизображений, применяется для интерактивной и пакетной обработки данных, полученных на VLA (Very Large Array) и других инструментах NRAO. Имеется простой командный язык для выполнения задач. Дальнейшую информацию о системе можно получить на ее сайте *http://www.aoc.nrao.edu/aips/* .

4. Система STARLINK

Программное обеспечение STARLINK — это набор средств для интерактивной обработки астрономических данных, получаемых на наземных и космических телескопах, и для визуализации результатов теоретических модельных расчетов. В состав системы входят пакеты и утилиты. Пакеты обеспечивают общие функции системы: обработку изображений, выполнение фотометрии, спектроскопии и поляриметрии, анализ временных рядов, управление базами данных, математические операции с данными, их графическое представление, преобразование данных из одного формата в другой и т. п. В состав STARLINK входят также библиотеки подпрограмм, упрощающие разработку астрономического программного обеспечения, и в частности необходимые для выполнения астрономических расчетов, управления данными, создания графических приложений и т. д. Сайт системы — *http://star-www.rl.ac.uk/* .

5. Система IDL — Interactive Data Language

IDL — коммерческая система визуализации и обработки цифровых данных. Она не является специализированной системой для обработки астрономических данных, но имеет огромный набор дополнительных пакетов, написанных астрономами для своих задач. Система широко используется также физиками, биологами, геофизиками и представителями других научных дисциплинах для визуализации и анализа данных. Это объясняется тем, что IDL работает с большим количеством форматов данных, включает встроенные библиотеки для выполнения математических операций с данными, обработки изображений и сигналов, позволяет встраивать программы, написанные пользователем на языках Фортран и Си. Имеется широ-

кий набор функций для визуализации данных (от построения простых диаграмм до создания трехмерной графики) и удобные утилиты для создания графических интерфейсов. Отладка пользовательских приложений происходит по обычной цепочке (создание программы — компиляция — включение библиотечных функций — тестирование) и выполняется в единой графической среде IDL. Дальнейшие сведения о системе можно найти на сайте *http://www.ittvis.com/idl/* .

Из такого, даже краткого, обзора видно, что все перечисленные системы обладают сходными возможностями: большим набором команд для визуализации и преобразования разнообразных астрономических данных, а также для анализа обработанных данных. Командный язык, обычно входящий в состав систем, позволяет работать в интерактивном и пакетном режимах, облегчает написание и выполнение собственных сценариев обработки данных. Имеется программный интерфейс к известным языкам программирования Фортран и Си, позволяющий расширять функции систем пользовательскими разработками.

Кроме пяти рассмотренных систем обработки, существует много удобных программных средств для различных операций с астрономическими данными. В частности, для предварительного ознакомления с данными приходится просматривать изображения, записанные в FITS-формате, и эту возможность предоставляют интерактивные программные пакеты ds9, skycat, Aladin. Редактирование, алгебраические операции со столбцами и их графическое представление удобно выполнять с помощью интерактивного редактора астрономических таблиц TOPCAT. Эти программные инструменты позволяют использовать астрономические Интернет-ресурсы — изображения и данные из каталогов, находящихся в центрах астрономических данных.

Достаточно сложно отдать предпочтение какой-либо из перечисленных систем. Во многих случаях они хорошо дополняют друг друга: одна система имеет преимущества в алгоритмах графического представления определенного типа данных, другая — предпочтительнее при фильтрации данных или классификации объектов. Поэтому часто астрономы используют не одно программное средство при работе со своими данными, например, MIDAS и IRAF или MIDAS и пакеты STARLINK, ds9, skycat — для быстрого просмотра

наблюдательного материала или Aladin — для поиска вспомогательного материала для изучаемого объекта и т. д. При этом, однако, одна из универсальных систем используется практически всегда, являясь базовой.

В этом пособии мы детально рассмотрим программирование и работу с разными структурами данных в стандартной европейской универсальной системе обработки астрономических данных MIDAS. В следующем разделе будет обсуждаться начало работы с этой системой, затем будут описаны командный язык MIDAS, его справочная система, работа с разными структурами и форматами данных, расширение возможностей за счет включения новых программ на языках Фортран и Си, детали визуализации спектров и изображений. В приложениях представлена информация об операционной системе Linux, с которой обычно работает MIDAS, систематизированы команды этой системы, приведены некоторые детали работы с Python-интерфейсом в MIDAS.

Авторы благодарны преподавателям СПбГУ проф. Н.В.Вощинникову, проф. В.П.Решетникову и доц. В.А. Яковлевой за полезные замечания, сделанные при чтении рукописи, проф. В.А. Гаген-Торну за идею объединения работ авторов в одно пособие и финансовую поддержку, а также многим студентам астрономического отделения СПбГУ, чьи вопросы в процессе выполнения специального практикума по тематике данного пособия способствовали более ясному изложению материала.

0. ЗАПУСК И РАБОТА С MIDAS

1. Запуск системы

После установки системы MIDAS в операционной системе появляется 4 дополнительные команды **inmidas**, **gomidas**, **helpmidas**, **drs**. Справочную информацию об этих командах в Linux (Unix) можно получить командой этой операционной системы **man** (см. *Приложение А*).

1.1.Команда inmidas

Систему MIDAS можно вызвать, набрав команду **inmidas**. Выполнение команды приводит к старту MIDAS-сессии и созданию окружения MIDAS. При этом в директории, в которой выполнена команда, создается новая поддиректория со стандартным названием **/midwork**. Это рабочая директория MIDAS, содержащая служебные файлы для текущей сессии.

Команда **inmidas** может иметь параметры. Подробную информацию о них можно получить, выполнив **man inmidas**, а кратко о параметрах команды можно узнать, запустив **inmidas –help**. Во втором случае на экран терминала выводится примерно следующее:

```
Usage: inmidas [unit] [-h midashome] [-r midvers]
               [-d display] [-p/-P/-nop]
               [-m mid_work] [-noh]
               [-j "midas-command-line"] [-help]
```

Рассмотрим параметры этой команды:

unit определяет номер сессии. Если это цифры в диапазоне от 00 до 99, то MIDAS работает в графическом режиме. Если значение

unit — буквы от xa до zz, то сессия запускается без использования графических окон, т. е. возможна работа на простейшем ASCII-терминале. Значение по умолчанию равно 00;

-p/-P/-nop указывает режим работы в MIDAS сессии. Без указания этого параметра с системой MIDAS может работать один пользователь (режим по умолчанию). Если указывается **-p** или **-P**, то работает несколько пользователей одновременно. Опция **-nop** запрещает режим параллельной работы нескольких пользователей.

Следует помнить, что при запуске команды **inmidas** или **inmidas 00** в поддиректории **/midwork**, упомянутой выше, стираются все служебные файлы, относящиеся к любым другим MIDAS-сессиям. Если указать параметры **-p** или **-P**, то эти файлы не стираются, а создается свой набор для новой MIDAS-сессии (если, конечно, имена новых файлов не совпадают с уже имеющимися). Несколько примеров:

inmidas 00
производится запуск сессии **00** (отметим, в именах служебных файлов присутствует идентификатор сессии, в данном случае **00**, например, **FORGR00.CTX**, **FORGR00.KEY**, **FORGR00.LOG**);

inmidas 10 -p
происходит запуск следующей сессии 10 с сохранением данных о предыдущих;

inmidas xx
начинается работа с MIDAS без графики.

Заметим, что при запуске двух сессий с одинаковыми номерами (например, 10) MIDAS выдает предупреждение:

```
Unit 10 is locked by another MIDAS session.
To unlock just continue.
Do you want to continue [yn]? (n):
```

-r midvers позволяет запустить нужную версию MIDAS. На компьютере могут быть установлено несколько версий, и по умолчанию **inmidas** вызывает наиболее свежую из установленных. Указав этот параметр, можно вызвать любую требующуюся версию. Например, запуск версии MIDAS **03SEP** происходит так:

```
inmidas -r 03SEP
```

-h midashome указывает директорию, в которой установлена версия MIDAS, с которой собирается работать пользователь. По умолчанию значение **midashome** берется из переменной MIDAS-окружения **MIDASHOME**. Для примера запуск версии **03SEP**, установленной в директории **/users/my/proba**, происходит по команде

```
inmidas -r 03SEP -h /users/my/proba
```

-m mid_work указывает имя рабочей директории. По умолчанию это **/midwork** в директории, из которой производится запуск MIDAS. В частности, старт версии **03SEP**, установленной в **/users/my/proba**, для обработки наблюдений в директории **/users/my/proba/my_observations** производится командой

```
inmidas -r 03SEP -h /users/my/proba -m
/users/my/proba/my_observations
```

-d display назначает X-сервер ОС для отображения графики. Напомним, что в среде X Windows можно работать на одном терминале, а графический вывод MIDAS переназначить на другой. Например, на терминал с номером **0.0** на компьютере **gong.astro.spbu.ru**

```
inmidas 00 -d gong.astro.spbu.ru:0.0
```

-j "midas_command" позволяет выполнить первой в данной MIDAS-сессии команду MIDAS, приводимую в кавычках.

-noh не очищает экран терминала и не выводит никаких дополнительных сообщений при старте MIDAS.

-help дает короткую справку о параметрах команды **inmidas**.

1.2. Команда gomidas

Команда **gomidas** возобновляет прерванную сессию. Параметры сессии сохраняются командой **inmidas** перед завершением сессии в служебных файлах, которые находятся в директории **/midwork**. При вызове команды **gomidas** можно указывать следующие параметры:

```
-d display
-m midwork
```

которые имеют тот же смысл, что и выше в команде **inmidas**.

При запуске **inmidas** в параллельной моде создается несколько наборов служебных файлов с сохраненными сессиями. И в случае старта **gomidas** будут использоваться те файлы, которые связаны терминалом, откуда запускается эта команда. Если такого набора не имеется, то будет использован самый последний.

1.3. Команда helpmidas

Команда **helpmidas** открывает графический интерфейс для получения справочной информации о командах MIDAS. Команда допускает следующие параметры:

```
-d display
-m midwork
-r midvers
-h midashome
```

которые имеют тот же смысл, что и для **inmidas**.

1.4. Команда drs

Эта команда позволяет выполнять команды системы MIDAS из *командной строки* операционной системы Linux. Формат команды

```
drs [-d(ebug)] [-u(pdate)] [-f(its output)]
    Midas_command
```

где при заданных опциях:

-d на терминал выводится сообщение о выполняемой команде **Midas_command**;
-u FITS-файлы, используемые в этой команде, перезаписываются, и изменения сохраняются;
-f результат выполнения команды, если в ней участвуют изображения, записывается в FITS-формате;
-h дает справку о команде.

Заметим, что по умолчанию после выполнения команды **Midas_command**, вызванной с помощью **drs**, FITS-файлы не обновля-

ются, но файлы, представленные во внутреннем формате MIDAS (**.bdf**, **.tbl**), обновляются всегда.

Естественно, что командой **drs** не поддерживаются графические возможности MIDAS, включая визуализацию данных.

Заметим, что из-за правил формирования имен файлов в операционной системы Linux символы * и) должны экранироваться \, то есть надо записывать их как * и \), или всю строку заключать в двойные кавычки. Например, прочитать все FITS ключевые слова, начинающиеся с **qq** из FITS-файла можно так:

```
drs read/descr ima0001.fits qq\*
```

или

```
drs read/descr ima0001.fits "qq*"
```

Удалить из этого FITS-файла все ключевые слова, начинающиеся на **ESO**, можно так:

```
drs -u delete/descr ima0001.fits ESO.\*
```

Заметим, что при выполнении последней команды FITS-файл преобразуется во внутренний формат MIDAS. Все дескрипторы, описывающие изображения во внутреннем формате, с именами, начинающиеся с **ESO**, удаляются, и результирующее изображение сохраняется в FITS-формате. В этом случае необходимо указывать ключ **-u**. Если ключ не указан, то после выполнения команды файл **ima0001.fits** не изменится.

2. Начало и окончание работы в MIDAS

Итак, начать сеанс работы с MIDAS в операционной системе Linux можно командой **inmidas** с параметрами, поясненными выше. Эта команда инициализирует параметры сессии, записывает их в служебные файлы, устанавливает рабочее окружение MIDAS, и на терминале появляется *командная строка* с приглашением

```
Midas 001>
```

Заметим, что **001** — номер команды, которая будет введена. Эти номера часто используется при интерактивной работе с MIDAS.

Когда надо завершить работу, то вводится команда MIDAS

```
BYE
```

Для продолжения прерванного сеанса работы с MIDAS следует использовать команду **gomidas**, также обсуждавшуюся выше.

3. Пакетный и интерактивный режимы работы

Как уже отмечалось, в MIDAS возможны оба режима. В *интерактивном* режиме в командной строке MIDAS дается одна команда за другой и так до выхода из системы. В *пакетном* режиме, используя командный язык MIDAS, пользователь пишет процедуру (пакет), включающую необходимые команды MIDAS и полностью определяющую разные последовательности их вызова, если в этом есть необходимость. Запуск процедуры с разными параметрами обычно происходит из командной строки.

В этом пособии мы будем иметь дело в основном с пакетным режимом, который используется в подавляющем большинстве случаев при нетривиальной обработке данных. Тем не менее, полезно представлять, как происходит работа с системой в интерактивном режиме.

4. Замечания об интерактивном режиме

При работе с MIDAS в этом режиме пользователь взаимодействует со специальной программой, которая анализирует, интерпретирует и выполняет вводимые команды. Эта программа называется *MIDAS-монитором*. Кратко об особенностях работы этой программы:

- MIDAS-монитор не различает большие и малые буквы, т. е. все равно малыми или большими буквами записана команда. (Отметим, что команда анализируется монитором, но имя изображения, заданное как параметр, он передает для анализа операционной системе. В Linux большие и малые буквы различаются, и поэтому если изображение на диске имеет, например, имя **`test.bdf`**, то именно так его и нужно указывать в MIDAS-команде. Если написать **`READ/DESCRIPTOR TEST`**, то операционная система не обнаружит файла **`TEST.bdf`** и выдаст сообщение об ошибке.)
- Максимальная длина строки при вводе равна 256 символам.
- Для продолжения команды на следующую строку используется символ — (минус).

  Запись в одну строку

```
EXTRACT/IMA pice = testima[100,120:200,220]
```

эквивалентна записи в несколько строк

```
EXTRACT/IMA pice = testima-
[100,120:200,220]
```

- Можно записывать несколько команд в одной строке, разделителем является символ **;**.
- После символа **!** записываются комментарии.
- Если текстовый параметр содержит пробелы, то его надо заключать в двойные кавычки.
- Существуют несколько способов записи параметров команд MIDAS, которые в общем случае являются позиционными. Кроме имен P1, P2, ..., P8, каждый параметр имеет собственное название, которое можно пользовать для вызова. Эти названия можно посмотреть в справочной информации о команде. Три способа записи параметров в команде (позиционный, через стандартные имена P1-P8 и собственные имена) в общем эквивалентны

```
STAT/IMA testima [<,<:>,>] ? ? ? test P
STAT/IMA testima P6=test P7=P
STAT/IMA FRAME=testima OUTTAB=test PLOT=P
```

- Прервать выполнение любой команды можно, нажав одновременно клавиши **Ctrl** и **C**. Заметим, что некоторые версии MIDAS реагируют на это иначе и, нажав эти клавищи, можно завершить всю MIDAS-сессию.
- Последние выполненные 15 команд остаются в буфере команд. Размер буфера меняется командой **SET/BUFFER**. Просмотреть содержимое буфера можно, нажав клавишу **Enter**.
- Любую команду, находящуюся в этом буфере, можно вызвать на выполнение еще раз, введя ее номер в буфере. Например, ввод в командной строке MIDAS

```
3;4
```

приведет к выполнению команд из буфера с номерами 3 и 4.

- Можно выполнить команду из буфера по заданному образцу, введя символ : и шаблон для поиска. В частности, вызвать первую команду из буфера, начинающуюся с WRITE можно так:

```
:WRITE
```

Заметим, что при поиске MIDAS-монитор начинает различать большие и малые буквы!

- Для редактирования команды из командного буфера надо ввести номер команды, добавив к нему символ точки. Вызов для редактирования 10-й строки из буфера происходит просто как

  10.

- Можно вызвать команду по шаблону для ее редактирования, введя символы :. и указав образец для поиска в буфере. Вызвать на редактирование первую команду из буфера, начинающуюся с WRITE, можно так

  :.WRITE ! (*или* .:WRITE)

- Команды в буфере можно просматривать и выполнять, пользуясь клавишами перехода вверх/вниз на клавиатуре.
- Если вы стали набирать команду и забыли ее точный синтаксис, то вы можете прервать эту команду, набрав символ **** и **Enter**. Команда запомнится в буфере командной строки, и вызвать ее на редактирование снова можно той же комбинацией клавиш **** и **Enter**.
- В MIDAS можно выполнить любую команду операционной системы, введя символ **$** и команду. В частности, список всех файлов в текущей директории выдается по команде

  $ ls

- Монитор MIDAS запоминает последнее имя каждого параметра и команды. Воспользоваться этим можно, введя символ точки, например, так:

  SHOW/TABLE mytable — показывает заголовок таблицы **mytable.tbl**
  EDIT/TABLE . — исполняется команда EDIT/TABLE mytable
  . yourtable — исполняется команда EDIT/TABLE yourtable

- В любой версии MIDAS всегда существует набор команд, дублирующих команды операционной системы. Эти команды можно использовать в интерактивном и пакетном режимах. Команды начинаются со знака — и перечислены ниже вместе с вызываемыми командами ОС

```
-DIR            $ls
-COPY           $cp
-@              $sh
```

```
-DELCNF         $rm  -i
-DELETE         $rm  -f
-RENAME         $mv
-TYPE           $cat
-MORE           $page
-PRINT          $lpr
```

Например, копирование таблицы в текущей директории может быть сделано командой

```
-COPY MID_WORK:tmp.tbl ./
```

- В MIDAS имеется возможность сделать дополнительные настройки окружения MIDAS сессии, удобные для пользователя. При старте MIDAS командами **inmidas** или **gomidas** автоматически происходит выполнение файла с именем **login.prg**, находящегося в директории, определенной переменной **MID_WORK** (по умолчанию **midwork** в домашней директории пользователя). С помощью этого файла пользователь может определить дополнительные настройки в своей MIDAS сессии.

Приведем пример файла **login.prg** с небольшими комментариями:

```
!+
! Дополнительные определения
!+
CREATE/COMMAND RK READ/KEYWORD !
```

можно определять аббревиатуру для часто используемых команд

```
CREATE/COMMAND WK WRITE/KEYWORD
CREATE/COMMAND RD READ/DESCR
CREATE/COMMAND WD WRITE/DESCR
CREATE/COMMAND XH CREATE/GUI HELP
CREATE/COMMAND SMOOTH/SPECIAL @@ mysmooth !
```

можно определить новую команду

```
!
CREATE/DEFAULT CREATE/GRAPH ? 400,800 !
```

дать определение размеров графического окна и

```
CREATE/DEFAULT CREATE/DISP ? 600,600,400,400 !
```

размера и расположения дисплея изображений

Контрольные вопросы

1. Какие команды ОС добавляются при инсталляции MIDAS и для чего они предназначены?
2. Что позволяют определять параметры команды **`inmidas`**?
3. В чем различие двух режимов работы в MIDAS?
4. Как стартовать MIDAS, чтобы можно было работать без использования графики?
5. Как различить служебные файлы MIDAS, относящиеся к разным сессиям?
6. Можно ли выполнить команду MIDAS из командной строки операционной системы?
7. В каких случаях управляющая программа MIDAS (MIDAS-монитор) «различает» малые буквы в команде?
8. Сколько имеется способов записи параметров в командах MIDAS и чем они отличаются?.
9. Что такое буфер команд и какими способами можно вызвать на выполнение команды из него?
10. Как из MIDAS выполнить команду операционной системы?
11. Для чего в MIDAS используется файл **`login.prg`**?
12. Что определяется переменной **`MID_WORK`**?

I. ЯЗЫК ПРОЦЕДУР MIDAS

1. Процедуры MIDAS

Как уже отмечалось, MIDAS может работать в *интерактивном* и *пакетном* режимах. В первом случае вы просто даете команды этой системы одну за другой, во втором — собираете эти команды в файл, называемый *процедурой*, и затем выполняете ее. Эффект будет один и тот же.

Существует ряд дополнительных команд, помогающих организовать такие процедуры. Эти команды образуют язык процедур MIDAS, называемый *MIDAS Command Language* или сокращенно *MCL*. Дополнительные команды, входящие в MCL, не выполняются в интерактивном режиме и работают только внутри процедур при их запуске в пакетном режиме.

2. Операторы процедур

В MIDAS не надо никаких специальных операторов, чтобы *начать* или *закончить* процедуру. Существуют, правда, операторы, которые желательно помещать в начало процедур, например операторы, определяющие параметры процедуры.

Внутри процедур можно использовать любые команды MIDAS, включая, конечно, команды языка MCL.

3. Переменные — ключевые слова

Начнем с простого вида данных в MIDAS — *ключевых слов*. Как и массивы в Фортране или Си, они имеют *имя*, *тип* и *размер*.

Имя не может быть длиннее 15 символов.

Тип может быть **real**, **integer**, **character** или **double precision**.

Размер означает число элементов и формально не ограничен.

4. Локальные ключевые слова

По умолчанию ключевые слова являются *глобальными*, т. е. сохраняющими свое значение при переходе из одной процедуры в другую. Зачастую это нежелательно, и тогда ключевые слова определяют как *локальные* переменные, т. е. переменные, которые функционируют только внутри данной процедуры. Например следующее определение шести ключевых слов:

```
DEFINE/LOCAL kx/D/1/1 0.1000000000000001
DEFINE/LOCAL werter/I/1/4 2,0,1
DEFINE/LOCAL kxa/R/2/4 2.0,3.0
DEFINE/LOCAL aw/C/1/5 abba
DEFINE/LOCAL kxb/I/1/20 5 all
DEFINE/LOCAL awa/C*5/1/4 abcde all
```

Напомним, что эти операторы выполняются только внутри процедур.

Здесь **I**, **R**, **D** и **C** означают **integer**, **real**, **double precision** и **character**; два числа после этих букв и наклонной черты показывают *индекс первого* элемента и *количество* элементов ключевого слова, начиная с него; и, наконец, числа (символы) после пробела, разделенные запятой, — *начальные значения*. Параметр **all** позволяет присвоить одно и то же значение сразу всем элементам ключевого слова.

Данные нами выше определения дадут следующий результат:

Имя	Тип	Длина	Начальные значения
kx	**double prec.**	1	**kx(1)** = 0.1000000000000001
werter	**integer**	4	**werter(1)** = 2; **werter(2)** = 0; **werter(3)** = 1
kxa	**real**	3(4)	**kxa(2)** = 2.0; **kxa(3)** = 3.0
aw	**character**	5	**aw** = abba
kxb	**integer**	20	**kxb(1) =** 5; **...; kxb(20)** = 5
awa	**character**	20	**awa(1)** = **awa(2)** = **awa(3)** = **awa(4)** = abcde;

Следует сделать несколько замечаний к этой таблице:

1) в документации MIDAS указано, что элементы ключевого словам, которым не присвоены начальные значения (например, **werter(4)**), по умолчанию получают значение равное нулю. Это правило выполняется не всегда. Присваиваемое по умолчанию значение может быть малым, но не равным нулю, в зависимости от операционной системы и версии MIDAS. Настоятельно рекомендуем всегда явно указывать начальные значения для всех элементов ключевых слов;
2) каким бы ни был индекс первого элемента в операторе, ключевое слово по правилам MCL всегда включает элементы, начиная с первого, поэтому **kxa(1)** будет определен и (почти) равен нулю;
3) очевидно различие «числовых» и «нечисловых» ключевых слов при определении начальных значений (в первом случае нескольких начальных значений разделяются запятыми (см., например, **werter**), а во втором — задается просто последовательность символов (см. **aw**), при этом если в начальном значении строкового ключевого слова встречаются пробелы, то необходимо заключать такую строку в кавычки “ ” (пример ключевого слова **awa** показывает определение символьного массива из 4-х строк по 5 символов).

Заметим, что одно только имя ключевого слова по умолчанию означает первый элемент, т. е. **werter** и **werter(1)** представляют собой одно и то же для MIDAS.

Для увеличения размера уже существующего ключевого слова надо его удалить и создать новое с тем же именем, но большего размера. Автоматическое расширение ключевых слов не предусмотрено.

5. Имя и значение переменной

Только что выше мы присвоили начальные значения нескольким переменным (ключевым словам). Важно различать *имена* переменных и их *значения*. Если мы пишем **{kxa}**, то это вызывает значение переменной; а если **kxa**, то имя. Различие будет очевидно в следующем пункте.

6. Ввод и вывод значений переменных

Напечатать что-то на экране можно, используя следующую команду:

```
WRITE/OUT <список_вывода>
```

где **<список_вывода>** может включать текст и значения ключевых слов.

Например, оператор

```
WRITE/OUT kx = {kx}
```

выдаст на экран строку

```
kx = 1.00000E+00
```

Отметим еще раз, что здесь **kx** без скобок **{}** было интерпретировано как имя переменной, а **{kx}** — как ее значение.

Формат чисел, выдаваемых при печати, меняется командой **SET/FORMAT** (см. простой пример ее использования в п. 17). По умолчанию форматы ключевых слов в интерактивном режиме: для целых — **I4** (если число имеет меньше четырех знаков, иначе выводятся все знаки), для вещественных и с двойной точностью — **E15.8**, а в пакетном режиме: для целых — **I4** (или все число), вещественных и с двойной точностью — **E12.5**.

7. Как присвоить значение переменной

Значение ключевому слову можно присвоить, используя команду **WRITE/KEYWORD**. Например, оператор

```
WRITE/KEYWORD kxa/R/3/2 2.7,3.2
```

присвоит значение **2.7** третьему элементу ключевого слова **kxa,** а **3.2** — четвертому.

Кроме задания начального значения, ключевому слову можно присвоить значение и иным образом, например такой командой:

```
kxa(4) = 2
```

но такой способ присвоения значений работает только в пакетном режиме (внутри процедур). При этом нельзя присвоить значение элементу ключевого слова, который выходит за границы, определенные командой MCL **DEFINE/LOCAL** или более общей командой

MIDAS **WRITE/KEYWORD,** т. е. в данном случае нельзя присвоить значение пятому элементу **kxa**.

Подобное присваивание удобно при выполнении вычислений. Здесь в левой части равенства следует помещать имена переменных, а в правой — значения переменных (имена в фигурных скобках) и константы, а также любые включающие их алгебраические выражения и определенные в MIDAS функции

```
kxa(4) = 2 + 10 * {kxa(2)}
```

Ввод значения в ключевое слово можно выполнить с помощью команды

```
INQUIRE/KEYWORD IN_A "Enter name of image:"
```

которая выводит указанную строку диалога, после чего монитор MIDAS ожидает ввода значения, завершающегося нажатием клавиши **Enter**. Например, для ввода символьного ключевого слова **new_word**

```
INQUIRE/KEYWORD new_word/c/1/5 "Give a name:"
```

8. Глобальные ключевые слова

Команда **WRITE/KEYWORD** имеет еще одну *важную функцию* — она может быть использована для того, чтобы определить *глобальное* ключевое слово. Например

```
WRITE/KEYWORD kk/I/1/3 1,4,9
```

порождает *новое* глобальное ключевое слово с начальными значениями **kk(1)** = 1, **kk(2)** = 4, **kk(3)** = 9. При этом определять его командой **DEFINE/LOCAL** не нужно.

9. Алгебраические выражения

В MIDAS можно использовать стандартные *математические* операции: **+ – * /** и *скобки* **()**, чтобы менять порядок операций. Например

```
kxa = (5.6 * ({kxa} + 4) – 1) / 3.1
```

10. Математические функции

Большинство *стандартных* математических функций реализовано в виде встроенных подпрограмм.

```
M$LN (arg1) натуральный логарифм real/double как arg1
M$LOG10 (arg1) десятичный  логарифм real/double как
      arg1
M$EXP (arg1) экспонента real/double как arg1
M$SIN (arg1) синус real/double как arg1
M$COS (arg1) косинус real/double как arg1
M$TAN (arg1) тангенс real/double как arg1
M$ASIN (arg1) арксинус real/double как arg1
M$ACOS (arg1) арккосинус real/double как arg1
M$ATAN (arg1) арктангенс real/double как arg1
M$SQRT (arg1) корень квадратный real/double как arg1
M$ABS (arg1) абсолютная величина integer/real/double
      как arg1
M$NINT (arg1) ближайшее целое integer при real/
      double arg1
```

При этом важно помнить, что в тригонометрических функциях углы измеряются в градусах!

Реализовано также несколько нематематических функций. О них пойдет речь далее, в одном последующих разделов.

11. Циклы

Для циклов используется конструкция

```
DO <счетчик> = <начало> <конец> <шаг>
<команды>
ENDDO
```

где **DO** и **ENDDO** — это служебные слова, **<счетчик>** — целое (**integer**) ключевое слово, **<начало>**, **<конец>**, **<шаг>** — целые алгебраические выражения, **<команды>** — любые команды MIDAS.

Например, следующие операторы дадут значения **a** = 5.0:

```
DEFINE/LOCAL n/I/1/1 0
DEFINE/LOCAL end/i/1 2
DEFINE/LOCAL a/R/1/1 0.0
```

```
DO n = 1 {end} 2
a = {a} + 2.5
ENDDO
```

Заметим, что между **n** и **=**, а также между **=** и **1** в третьей строке должен быть пробел, а переменная-счетчик цикла должна быть обязательно описана.

Следует помнить, что цикл в MIDAS всегда исполняется 1 раз, даже если **<начало>** больше **<конец>**!

12. Условные операторы

Эти операторы сходны с аналогичными операторами в Фортране и возможны в трех вариантах:

```
а) IF <условие> <команда>

б) IF <условие> THEN
   <команда>
   ...
   ENDIF

в) IF <условие> THEN
   ...
   ELSEIF <условие> THEN
   ...
   ELSE
   ...
   ENDIF
```

Разрешается использовать до 8 уровней вложения операторов **IF**. Обратите внимание на необычное слово **ELSEIF**.

В вышеприведенных конструкциях **<условие>** формируется логическими операциями

```
.EQ. (равно) .NE. (не равно)
.LT. (меньше) .LE. (меньше равно)
.GT. (больше) .GE. (больше равно)
```

Например, если **i** равно 5, то следующие операторы дадут **b** = 7.

```
IF i .EQ. 5 THEN
b = 7
```

```
ENDIF
```

Отметим, что пробелы до и после **.EQ.** и т. п. необходимы.

13. Комментарии

Все символы в строке после знака **!** считаются комментарием и не обрабатываются MIDAS.

Два примера:

```
! this is a comment
WRITE/OUT kxa ! this is a comment too (т. е. все
после kxa в этой строке не печатается)
```

Здесь также необходим пробел перед **!** во второй строке.

14. Связь с операционной системой

MIDAS работает в операционных системах типа UNIX, включая Linux, а также Windows XP (на основе пакета cygwin, который эмулирует работу в графической оболочке UNIX/Linux X Windows). Некоторые часто используемые команды Linux даны в *Приложении А*.

Когда вы работаете в MIDAS, то можете давать команды операционной системы, например, чтобы посмотреть, сколько свободного места осталось на диске. Такие команды должны следовать за знаком **$** и могут быть в принципе использованы в MIDAS-процедурах.

Помните, что при старте MIDAS командой **inmidas** вы можете работать с файлами, процедурами и пр. только из *текущей* директории. Если вы хотите сменить текущую директорию, например, чтобы вводить и выводить файлы в другую, то команда **$cd** не поможет. В этом случае следует использовать команду MIDAS **CHANGE/DIRECTORY**.

15. Параметры процедуры

Если ваша процедура должна иметь несколько параметров, их нужно описать в начале процедуры так:

```
DEFINE/PARAMETER P1 <нач_значение> <тип> <промпт>
<мин_значение>,<макс_значение>
```

где **<тип>** может быть **N** (number), **C** (character), **T** (table), **I** (image) или **F** (fit file), что, заметим, отличается от типов переменных (ключевых слов)!

Число параметров может контролироваться самой системой MIDAS, например

```
DEFINE/MAXPAR 2
```

будет проверять, сопровождаете ли вы вызов процедуры добавлением двух и только двух параметров. В ином случае вы получите сообщение с предупреждением.

Вы можете отказаться от присвоения начальных значений и заставить MIDAS всегда проверять, подставляете ли вы параметры описанного типа, когда стартуете процедуру. Для этого нужно поставить знак + вместо **<нач_значение>**.

Остальные параметры команды **DEFINE/PARAMETER** практически не будут нами использоваться.

В MIDAS процедурах имеется возможность определять временные файлы для работы с промежуточными результатами выполнения команд. Эти файлы имеют стандартные имена, начинающиеся с **middumm**. Имя такого файла обозначается в команде префиксом **&** перед именем файла. Пример — небольшая процедура

```
DEFINE/PARAM p1 ? ima "Enter input frame: "
DEFINE/LOCAL lc/c/1/60 {p1}
FILTER/DIGIT {lc} &qq laplace ! результат выполнения этой команды запишется в файл middummqq.bdf
```

Отметим, что эти временные файлы не удаляются автоматически при завершении процедуры, но их можно удалить командой **DELETE/TEMP**.

16. Вызов процедуры

Чтобы выполнить процедуру, следует набрать *@@*, пробел и имя процедуры (имя соответствующего файла) с последующими параметрами разделенными пробелом (см. следующий пункт в качестве примера).

17. Пример процедуры

Напишем простую процедуру, использующую некоторые команды, обсуждавшиеся выше. Процедура вычисляет **z = x**y** и имеет два очевидных входных параметра **x** и **y**.

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! exm1.prg – procedure to calculate z = x**y
! CROSSREF x y
ECHO/FULL
! SET/FORMAT I4
! SET/FORMAT F12.8,E24.16
DEFINE/PARAMETER p1 + NUMBER "Enter x: "
DEFINE/PARAMETER p2 + NUMBER "Enter y: "
DEFINE/MAXPAR 2
DEFINE/LOCAL x/i/1/1
DEFINE/LOCAL y/r/1/1
DEFINE/LOCAL z/d/1/1
WRITE/KEYWORD z1/d/1/1
WRITE/KEYWORD x {p1}
WRITE/KEYWORD y {p2}
WRITE/OUT x = {x} y = {y}
IF x .EQ. 0d0 THEN
z = 0.0
ELSE
z = M$EXP ({y} * M$LN ({x}))
ENDIF
WRITE/OUT z = {z}
z1 = {z}
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

Если вы вошли в MIDAS и вызвали эту процедуру на исполнение посредством

```
@@ exm1.prg 2 2.0
```

то получите в результате ее работы среди нескольких прочих строк следующие:

```
x = 2 y = 2.00000000
z = 4.0000000000000000E+00
```

Информацию о полезных командах **CROSSREF** и **SET/FORMAT**, включенных в процедуру, можно получить, используя справочную систему MIDAS, о которой пойдет речь в следующем разделе.

18. Список команд MCL

Для справки приведем все команды язык процедур MIDAS.

```
BRANCH var comparisons labels

CONTINUE
CROSSREF label1 ... label8
DEFINE/LOCAL keyword data all_flag level_flag
DEFINE/PARAMETER par def type prompt limits

DEFINE/MAXPAR no_par

DO loopvar = begin end step
... command body ...
ENDDO
ENTRY proc

GOTO label

IF par1 op par2 command
IF par1 op par2 THEN
... if-sequence ...
ELSEIF par3 op par4 THEN
... else if-sequence ...
ELSE
... else-sequence ...
ENDIF

label:

PAUSE
RETURN par1 ... par3
ECHO/FULL levela,levelb
ECHO/OFF levela,levelb
ECHO/ON levela,levelb

WRITE/ERROR

! comment
```

И еще несколько команды MIDAS, которые часто используются при написании процедур

```
COMPUTE/KEYWORD reskeyword = expression
SET/FORMAT format_specs
WRITE/OUT text

INQUIRE/KEYWORD key [prompt_string] [flush_opt]
```

Упражнения

1) Написать процедуру, которая решает квадратное уравнение $ax^2 + bx + c = 0$, где a, b и c — параметры, используя формулы $x_1 = (-b + \sqrt{(b^2 - 4ac)})/(2a)$; $x_2 = (-b - \sqrt{(b^2 - 4ac)})/(2a)$. Учесть, что при $b^2 - 4ac < 0$ нет действительного решения и процедура должна печатать соответствующее сообщение.
2) Написать процедуру, которая решает квадратные уравнения при фиксированных значениях b и c (два параметра) и наборе значений a, задаваемом еще тремя параметрами: начальной величиной **a1**, шагом **da** и конечной величиной **an** (например, a, меняющееся от 16 с шагом 0.5 до 24). Использовать свою (возможно модифицированную) процедуру из предшествующего задания, т. е. программа должна вызывать процедуру решения квадратного уравнения. Сохранять все решения уравнений в двух *глобальных* переменных-массивах — ключевых словах **x11** и **x21**.

II. СПРАВКА И ОТЛАДКА В MIDAS

MIDAS имеет хорошо разработанную справочную систему. Рассмотрим сначала ту ее часть, которая доступна в режиме on-line (в текстовой и графической моде — см. п. 8 *Приложения А*), а затем встроенный в систему учебник. После этого будут описаны средства MIDAS, облегчающие отладку процедур.

1. Справка в текстовой моде

Когда вы находитесь в среде MIDAS (т. е. он запущен одним из указанных в разделе 0 способов), список возможных команд может быть извлечен следующей командой без параметров:

```
HELP
```

Систематизированный список команд с некоторыми комментариями приведен нами в *Приложении Б*.

Напомним общую структуру команд MIDAS

```
COMMAND/QUALIFIER par1 par2 ... par8
```

где **COMMAND** — имя команды, **QUALIFIER** — один из ее квалификаторов, **par1** — первый параметр и т. д. Обычно команда обозначает некое действие, а квалификатор — например, то, с каким типом данных это действие выполняется.

Информацию о конкретной команде, например **command,** можно получить так:

```
HELP command
```

Более подробная справка выдается, если указать имя команды и один из ее квалификаторов

```
HELP command/qualifier
```

Заметим, что параметры, приводимые в описании команд в квадратных скобках (например, **[par3]**), являются необязательными.

Если нужна наоборот лишь очень краткая (в одну строчку) информация, показывающая основное назначение и список параметров какой-либо команды, поставьте после нее два вопросительных знака, например

```
WRITE/KEYWORD ??
```

Это даст требуемые сведения о команде **WRITE/KEYWORD**.

Если вы забыли имя команды, но помните его начало (например, **CO**), напишите это с вопросом

```
CO?
```

и высветится список всех команд, имена которых начинаются на **CO**.

2. Справка в графической моде

Выше мы имели дело со справками, получаемыми в текстовой моде. Полезно знать, что можно подключиться к справочной системе MIDAS и в графической моде. При этом вы, конечно, должны находиться в графической оболочке UNIX/Linux, например, **X Windows**. Как запустить эту оболочку, показано в *Приложении А*.

Итак, находясь в среде **X Windows**, дайте следующую команду:

```
CREATE/GUIDE HELP
```

и скоро увидите новое окно со списком команд MIDAS. В этом окне можно искать справочный материал, используя мышку. В графическом режиме легко перемещаться по иерархии справочного текста. Можно в любом месте, где имеется название команды, указать курсором и, нажав клавишу мыши, перейти к описанию этой команды. Меню History сохраняет всю последовательность просмотренных справок, позволяя возвращаться к нужным описаниям.

Аналогичное графическое окно вызывается командой операционной системы **helpmidas** (см. раздел 0).

3. Справка о MCL

Справку о командах языка процедур MIDAS (MCL) можно получить следующим образом:

```
HELP/CL
```

даст полный список таких команд, а

```
HELP/CL command
```

покажет информацию о команде **command**.

4. Встроенный учебник

Иногда установленная версия MIDAS включает также учебник (tutorial), который весьма подробно объясняет различные аспекты работы с MIDAS. В этом случае, например, команда

```
TUTORIAL/HELP
```

разъяснит использование справочной системы MIDAS (т. е. справка о справке). Существуют, конечно, и иные, более полезные квалификаторы команды **TUTORIAL**, но не все они могут быть задействованы на вашем PC.

5. Отладка с ECHO

MIDAS имеет несколько возможностей облегчить отладку процедур. В частности, если вставить команду **ECHO** с квалификаторами **FULL**, **ON** или **OFF** в начале процедуры, то при ее работе:

```
ECHO/ON
```

приведет к выводу на экран всех исполняемых операторов (в дополнение к другой информации);

```
ECHO/FULL
```

выведет на экран ту же информацию, а также дополнительные строки, показывающие какие значения для переменных в скобках **{}** были подставлены системой MIDAS;

```
ECHO/OFF
```

отменяет полностью оба вышеописанных режима.

Например, если запустить следующую процедуру:

```
ECHO/ON
DEFINE/LOCAL a/d/1/1 0d0
a = {a} + 1d0
```

то на экране появятся три строчки (эквивалентные написанным выше). Но если заменить **ECHO/ON** на **ECHO/FULL**, то после соответствующего оператора высветится еще и дополнительная строчка

```
a = 0.00000E+00 + 1d0
```

6. Отладка с DEBUG

Еще более мощной является команда

```
DEBUG/PROCEDURE level_min,level_max switch
```

где **level_min** и **level_max** — некоторые числа, по умолчанию равные 1 и 1 соответственно, **switch** — переключение режима отладки **ON**|**OF.** Если после такой команды запустить процедуру, то она будет выполняться пошагово — оператор (команда) за оператором с паузами между ними. Во время каждой паузы появляется приглашение

```
Mdb>
```

Нажмите **Enter,** и следующий оператор будет исполнен и т. д.

Когда приглашение

```
Mdb>
```

находится в *текущей* (последней) строке на экране, можно использовать специальные *подкоманды*. Их список выдается подкомандой **h**

```
Mdb> h
```

Некоторые из подкоманд приведены в таблице.

Подкоманда	Значение
go	сделать следующий шаг
quit	выйти из исполняемой процедуры
r keyword	показать значение ключевого слова **keyword**
pause	дать возможность запустить команду MIDAS [чтобы вернуться обратно в процедуру, дайте MIDAS-команду **CONTINUE**]

Следует подчеркнуть, что в отличие от обычных команд MIDAS, эти подкоманды пишутся только строчными буквами, т. е. подкоманда **GO** или **Go** работать не будет!

Заметим, что единственный способ узнать значение какого-либо локального ключевого слова — это дать команду MIDAS

```
DEBUG/PROCEDURE 1,3 ON
```

и запустить процедуру. Затем в нужном месте ответить на приглашение следующим образом:

```
Mdb> r my_keyw
```

где **my_keyw** — имя ключевого слова, значение которого следует узнать.

Когда закончите отладку процедур(ы), не забудьте дать команду

```
DEBUG/PROCEDURE level_min,level_max OFF
```

чтобы выйти из режима отладки.

7. Тестовая трансляция

Существует также возможность проверить все команды MIDAS в процедуре на соответствие стандарту (имеется в виду правильность числа параметров и т. п.). Для этого служит команда

```
TRANSLATE/SHOW proc X
```

где **proc** — имя файла, содержащего тестируемую MIDAS-процедуру.

8. Обзор возможностей команды HELP

Следующая таблица дает более полный, чем выше, обзор разных вариантов использования команды **HELP**, являющейся очень важной для любых пользователей MIDAS.

Команда	Назначение	Пример
HELP	Краткий обзор команд MIDAS	
pattern?	Список команд, начинающихся с *pattern*	**CO?**
HELP com	Список всех команд, начинающихся с com, с кратким описанием параметров	**HELP READ**
HELP *command/qualif*	Подробное описание команды *command/qualif*	**HELP READ/KEY**

Команда	Назначение	Пример
command/qualif ??	Краткое описание команды *command/qualif* и ее параметров	**READ/KEY ??**
HELP/QUALIF *qualif*	Список всех команд, имеющих квалификатор *qualif*	**HELP/QUALIF TABLE**
HELP/SUBJECT	Полный список тематических разделов справочной команды	
HELP/SUBJECT *subject*	Подробная информация о командах тематического раздела *subject*	**HELP/SUBJECT TABLE**
HELP/CL	Краткий обзор команд MCL — командного языка MIDAS	
HELP/CL *command*	Справочная информация по команде MCL *command*	**HELP/CL IF**
HELP/KEY *keyword*	Описание ключевого слова *keyword*	**HELP/KEY INPUTI**
HELP *[topic]*	Описание возможностей в тематическом разделе *topic*. Названия некоторых тематических разделов: **DataInput** — форматы для ввода и вывода информации в MIDAS; **ImageDisplay** — информация по теме MIDAS-окно для показа изображений; **Printers** — информация о принтерах в MIDAS; **TapeDevices** — то же, но о накопителях на магнитной ленте; **News** — описание изменений в более свежей версии системы. (Команды должны быть даны, как показано на примере последней.)	**HELP [News]**
HELP/APPLIC	Полный список дополнительных команд	
HELP/APPLIC *applic*	Подробное описание дополнительной команды *applic*	**HELP/APPLIC AUTOCUTS**

Упражнения

Замечание: Задание представляет собой набор вопросов из-за специфики материала — описания справочных и отладочных средств MIDAS.

1) Сколько квалификаторов (qualifiers) имеет MIDAS-команда **WRITE**?
2) Какое максимальное число параметров может иметь команда **OPEN/FILE**?

3) Какое число **IF**-конструкций дано в help'е о MCL-команде **IF**?
4) Существует ли **TUTORIAL** для **PLOT** на **Mercury** (или на том PC, который вы используете)?
5) Сколько строк будет выведено на экран следующей процедурой **TASK4.PRG**:

```
ECHO/FULL
DEFINE/PARAMETER p1 + number
DEFINE/LOCAL a/d/1/1 0
IF {p1} .LT. 0d0 THEN
a = {p1} * (-1)
WRITE/OUT abs(p1) = {a}
ELSE IF {p1} .EQ. 0d0 THEN
WRITE/OUT abs(p1) = 0
ELSE
WRITE/OUT abs(p1) = {a}
ENDIF
```

если она вызывается как

```
@@ TASK4.PRG 1
```

6) Сколько шагов (число строк с **Mdb>**) будет иметь процедура **TASK4.PRG,**

вызванная как

```
@@ TASK4.PRG -2
```

если команда

```
DEBUG/PROCEDURE 1,3 ON
```

дана перед этим?

7) Если вы дали подкоманду

```
Mdb> pause
```

что вернет вас назад (выбрать правильные варианты):

CON, con, cont *или* CONT ?

III. СИСТЕМА ТАБЛИЦ В MIDAS

0. Введение

Сегодня самыми распространенными в астрономии наблюдательными данными являются спектры и изображения. Результаты их анализа чаще всего представляются в виде таблиц. Практически любая система редукции астрономических данных позволяет выполнять разнообразные операции над таблицами, и MIDAS не является исключением. В нем имеется широкий набор команд для выполнения арифметических операций со столбцами таблиц, графического отображения затабулированных значений, объединения таблиц, выделения подмножеств таблицы по определенным пользователем правилам и т. д. Таблицы являются еще одним типом данных, представленным в MIDAS наряду с рассмотренными выше ключевыми словами.

Таблицы используются внутри системы MIDAS в трех случаях: для внутренних вычислений, для импорта/экспорта данных и в пользовательских приложениях. Независимо от этого, все MIDAS-таблицы имеют одну и ту же структуру.

1. Структура таблиц

Таблицы в MIDAS имеют *заголовок* и *элементы*, организованные в *столбцы* (*columns*) и *строчки* (*rows*).

Элементы таблицы в столбце должны быть одного из следующих типов: *целый* (*integer)*, *вещественный* (*real)*, *символьный* (*character)*, а также *массивом* значений одного из указанных типов.

Каждый столбец таблицы описывается *меткой* (*label*) или названием, *форматом* (*format*) представления данных и, возможно, физической *размерностью* (*physical units*) данных. Формат данных представляется так, как принято в Fortran. Например: название — Wavelength, формат — E10.6 и размерность — cm.

В таблицах всегда существуют два дополнительных, *системных* столбца с метками (именами) **SELECT** и **SEQUENCE** (подробнее об этих столбцах см. ниже).

На *столбец* можно сослаться по его *номеру* (например, **#2** для второго столбца в таблице) или по метке (например, **:Flux**); для системных столбцов — только по метке **:SEQUENCE** (или сокращенно **:SEQ**) и **SELECT** (или **SEL**, только без двоеточия!).

Доступ к *строчке* таблицы может быть произведен по ее *номеру* (например, **@3** для третьей строчки таблицы). Обратите внимание на различие префиксов **@** и **#**.

Если элемент не определен, он имеет значение **NULL** и представляется как * для всех типов данных, кроме символьного, когда такой элемент выглядит как пустое место. Здесь и ниже будет называть строки таблиц *строчками*, а переменные символьного типа (character) — *строками*.

Таблицы могут располагаться на диске двумя способами: *записями* (*records*), т. е. строчками, и транспонировано (*transposed*ly) столбцами. По умолчанию используется второй способ.

Таблицы хранятся в файлах со стандартным расширением **.tbl**. Информация представляется во внутреннем формате MIDAS, и поэтому содержимое этих файлов не может быть просмотрено обычными командами системы Unix/Linux (**more**, **less** и др.) или стандартными редакторами (**joe**, **jed**, **vi** и т. п.). Специальные средства для просмотра и редактирования MIDAS-таблиц будут рассмотрены ниже.

2. Создание таблицы

Для того, чтобы создать новую таблицу, дайте команду

```
CREATE/TABLE table ncol nrow
```

где **table** — имя новой таблицы, **ncol** и **nrow** — число столбцов и строчек в ней. Последние числа не имеют большого значения, поскольку таблицы автоматически расширяются, если это оказывается

необходимым. Отметим, что имя таблицы — это имя (без расширения) файла, в котором она хранится, например, имя таблицы **t6** и соответствующего файла **t6.tbl**.

Предыдущая команда создает только часть *заголовка* таблицы. Следует также определить *столбцы*. Это может быть сделано командой

```
CREATE/COLUMN table column unit format type
```

где **table** — имя таблицы, **column** — метка (label) нового столбца, **unit** — физическая размерность значений в столбце, данная в апострофах (например, «sec»), **format** — фортраноподобный формат, в котором данные будут показаны на экране или распечатаны (существует много форматов, кроме обычных **A**n, **F**n.m, **E**n.m), **type** — тип данных (например, **R*4** и **R*8** для вещественных чисел (real) с одинарной и двойной точностью; **I*1**, **I*2**, **I*4** для целых чисел (integer); **C*n** для строк (character) длиной n байт и т. д.).

Например, команды

```
CREATE/TABLE t6 8 8
CREATE/COLUMN t6 lambda "micron" F8.3 R*4
CREATE/COLUMN t6 flux "Jy" E16.8 R*8
```

создадут MIDAS-таблицу с именем **t6** (хранимую в файле **t6.tbl**), имеющую два столбца (кроме **:SEQUENCE** и **SELECT**) с метками **:lambda** и **:flux**.

Сказанного выше достаточно для определения новой таблицы. Если понадобится удалить столбец, используйте

```
DELETE/COLUMN table column
```

где **table** и **column** имеют то же значение, что и в команде **CREATE/COLUMN**, описанной выше.

3. Как поместить данные в MIDAS-таблицу

Это можно сделать несколькими способами: используя ASCII-файл, интерактивный встроенный редактор или специальную команду MIDAS.

3.1. Из ASCII-файла в MIDAS-таблицу

Когда существует обычный (ASCII) файл, содержащий данные, организованные в форме таблицы (т. е. в виде столбцов и строчек), можно преобразовать эти данные в MIDAS-таблицу посредством команды

```
CREATE/TABLE table ncol nrow file
```

где **table** — как обычно, имя таблицы, **ncol** и **nrow** — число столбцов и строчек в ней, **file** — полное имя (с расширением) файла, содержащего данные.

Например, если ASCII-файл **TNK.dat** содержит

```
1 1.0 9
3 4.0 50
2 5 7
05 6 23
4 7 0
```

(заметьте различные позиции чисел в строчках файла), то команда

```
CREATE/TABL t6 2 4 TNK.dat
```

создаст таблицу — файл **t6.tbl**, который будет содержать следующую информацию:

```
Table : t6

Sequence LAB001 LAB002
-------- -------------- --------------
1 1.000000E+00 1.000000E+00
2 3.000000E+00 4.000000E+00
3 2.000000E+00 5.000000E+00
4 5.000000E+00 6.000000E+00
5 4.000000E+00 7.000000E+00
-------- -------------- --------------
```

Отметим, что ранее определенные метки столбцов заменены метками, даваемыми по умолчанию (**LAB00***), т.к. рассматриваемая команда фактически переопределяет таблицу. Другая неожиданная деталь — созданы все 5 строк, а не 4, как указывал третий параметр команды **CREATE/TABL**.

Следует добавить, что внешний (т. е. вне среды MIDAS) обмен данными с MIDAS-таблицами происходит через ASCII-файлы. Мы только что видели перемещение данных из обычных файлов в MIDAS-таблицы. В обратном направлении это может быть произведено двумя командами, выполненными одна за другой

```
ASSIGN/PRINT FILE file
PRINT/TABLE table
```

где **table** — имя MIDAS-таблицы, **file** — имя ASCII-файла, в который будет помещено содержимое таблицы. Следующая команда эквивалентна двум предыдущим:

```
PRINT/TABLE table >file
```

3.2. Интерактивный редактор таблиц

Команда

```
EDIT/TABLE table
```

открывает MIDAS-таблицу с именем **table** и позволяет добавлять, удалять и модифицировать элементы этой таблицы.

Например, таблица **t6.tbl**, заполненная в п.3.1, будет выглядеть на экране примерно так:

```
Sequence?LAB001 ?LAB002 ?
????????????????????????????????????????
1? 1.000000E+00? 1.000000E+00?
2? 3.000000E+00? 4.000000E+00?
3? 2.000000E+00? 5.000000E+00?
4? 5.000000E+00? 6.000000E+00?
5? 4.000000E+00? 7.000000E+00?
```

Используя клавиши, перемещающие курсор, можно выбрать нужную позицию и изменить содержимое таблицы. Переход между элементами таблицы осуществляется клавишей **TAB**.

В редакторе возможно значительное число операций — подкомад. Для того, чтобы использовать их, нажмите одновременно **Ctrl** и **Z**. Внизу на экране должно появиться приглашение

```
????????????????????????????????????????????????????
? ?
? Command: ?
????????????????????????????????????????????????????
```

Наиболее важные подкоманды:

quit — выйти из редактора без сохранения сделанных изменений;
exit — выйти из редактора с сохранением изменений;
help — показать список всех подкоманд.

3.3. Команда для модификации элементов таблицы

Следующая команда MIDAS

```
WRITE/TABLE table col row-sel value
```

помещает константу, заданную как **value**, в элемент MIDAS-таблицы с именем **table** в точке пересечения столбца **col** и строчки **row-sel**.

Например, команда

```
WRITE/TABLE t6 #2 @3 44
```

присвоит значение **44.0** соответствующему элементу (3,2) MIDAS-таблицы **t6**.

Как мы отмечали выше, можно адресоваться к столбцу, используя его метку (label), следующую за двоеточием, или номер после знака **#** (для строчки — номер после знака **@**). Принимая во внимание текущую метку второго столбца (**LAB002**) таблицы **t6**, предыдущая команда эквивалентна следующей:

```
WRITE/TABLE t6 :LAB002 @3 44
```

4. Просмотр MIDAS-таблиц

Редактор таблиц в MIDAS не является самым удобным средством для того, чтобы просматривать содержимое MIDAS-таблиц — вместо него предпочтительнее использовать специальные команды.

Команда

```
SHOW/TABLE table
```

показывает *заголовок* MIDAS-таблицы **table**. С помощью этой команды можно также посмотреть номера колонок.

Команда

```
READ/TABLE table
```

высвечивает на экране все *элементы* таблицы **table** (в текстовом виде). Заметим, что при этом (системный) столбец **:SEQUENCE** всегда виден слева как некий «нулевой» (по порядку) столбец.

Можно представить значения, находящиеся в столбцах таблицы, и в графическом виде (если графическое окно MIDAS не было открыто, это нужно сделать сейчас командой **CREATE/GRAPHICS** без параметров, находясь, разумеется, в графической оболочке X Windows системы Linux — см. *Приложение А*). Следующая команда строит соответствующий график:

```
PLOT/TABLE table plane1 plane2
```

где **plane1** и **plane2** — указатели (метки или номера) столбцов, используемых соответственно для горизонтальной и вертикальной осей.

Команда

```
OVERPLOT/TABLE table plane1 plane2
```

эквивалентна предыдущей команде, но «рисует» поверх предыдущего графика (если он, конечно, был). Команда используется для того, чтобы показывать несколько зависимостей на одном рисунке.

Следует упомянуть несколько полезных параметров перечисленных команд для графического представления данных, заданных таблично. Параметр **p5** позволяет выбрать символы, которыми будут представляться точки (данные в таблице), а параметр **p6** задать тип кривой, соединяющей эти точки. Некоторые значения этих параметров даны в таблице.

p5	Символ	p5	Линия
0	нет	0	нет
1	точка	1	сплошная
2	шестиугольник	2	точечная
3	квадрат	3	короткий пунктир
4	треугольник	4	тире-точка
5	плюс	5	длинный пунктир
6	крест	6	тире-точка-точка
7	звездочка		
8	звезда		

Таким образом, например, команда

```
PLOT/TABLE t6 #2 #1 p5=4 p6=2
```

представит треугольниками значения в столбце **#1** (используемые для оси y, тогда как для оси x будут взяты значения из столбца **#2**) и соединит эти символы линией, состоящей из точек.

5. Преобразования таблиц

«Внешние» преобразования MIDAS-таблиц в ASCII-файлы и обратно были рассмотрены выше в п.3.1.

«Внутренние» преобразования MIDAS-таблиц в *ключевые слова* (*keywords*), *изображения* (*images*) и т. д. реализуются следующими командами:

```
COPY/TK table col row keyword
```

копирует соответствующий элемент (**row,col**) таблицы **table** в ключевое слово **keyword**;

```
COPY/KT keyword table col row
```

копирует кл. слово **keyword** в соответствующий элемент таблицы **table**;

```
COPY/TI table image
```

создает новое изображение **image** из таблицы **table**;

```
COPY/IT image table
```

создает новую таблицу **table** из изображения **image**.

6. Выделение части таблицы

Возможность выделить подтаблицу (в действительности отметить ряд строчек) увеличивает гибкость работы с таблицами в MIDAS. Операция реализуется командой

```
SELECT/TABLE table logical-expression
```

где логическое выражение **logical-expression** может содержать *арифметические* и *логические* операции и *отношения*, а также *логические* и *математические* функции.

Разрешенные *арифметические* операции: + – * / **

Отношения: **.LE. .LT. .GE. .GT. .EQ. .NE.**
Логические операции: **.AND. .OR. .NOT.**
Поддерживаются *математические* функции:
SQRT(:a) **LN**(:a) **LOG10**(:a) **EXP**(:a) **SIN**(:a)
COS(:a) **TAN**(:a) **ASIN**(:a) **ACOS**(:a) **ATAN**(:a)
ABS(:a) **INT**(:a) **MIN**(:a,:b) **MAX**(:a,:b) **MOD**(:a,:b)
и некоторые другие. Заметим, что аргументы тригонометрических функций должны быть в градусах.

Следует подчеркнуть, что в командах **SELECT/TABLE** и **COMPUTE/TABLE** имена функций отличаются от имен, используемых в языке процедур MIDAS *MCL*, где они предваряются символами **m$** !

Если произведено выделение (строчек), например следующим образом:

```
SELECT/TABLE t6 SQRT(:LAB001) .LT. 2 .AND. :LAB002
.GT. 3
```

то в соответствующих строчках будут изменены элементы системного столбца **SELECT** (**0** — если приведенное логическое выражение верно, и **1** — если нет).

Отметим, что вы едва ли увидите столбец **SELECT** на экране, но он существует и, что более важно, действует. Например, теперь, т. е. после выделения, сделанного предыдущей командой, команда

```
READ/TABLE t6
```

покажет вам только *две* (вторую и третью) строчки. Произойдет это потому, что последняя команда принадлежит к группе команд, которые адресуются только к выделенным строчкам, т. е. подтаблицам. Другие члены этой группы команд: **PRINT**, **PLOT**, **OVERPLOT**, **STATISTIC**, **COPY** (input), **MERGE** (input), **PROJECT**, **READ**, **REGRESSION**. Их объединяет то, что все они не изменяют информацию в таблицах.

Команда

```
SELECT/TABLE table ALL
```

удаляет любое выделение. В действительности отсутствие выделения означает, что выделены все строчки, и подтаблица идентична исходной таблице.

Следует также упомянуть, что команда **COMPUTE/TABLE**, которая будет описана ниже, автоматически удаляет любое выделение, и

в этом смысле она действует аналогично только что рассмотренной команде **SELECT/TABLE**.

7. Копирование таблиц

Кроме команд системы UNIX/Linux для копирования/перемещения (**$cp/$mv**) одного файла (включая MIDAS-таблицы) в другой файл (таблицу), существуют и команды MIDAS для перемещения информации из одной MIDAS-таблицы в другую.

Команда MIDAS

```
COPY/TT in-table column1 [out-table] column2
```

копирует столбец **column1** из таблицы **in-table** в столбец **colunm2** в той же таблице (или в иной, уже существующей таблице, если задано ее имя **out-table**). Здесь и ниже необязательные параметры даны в квадратных скобках.

Копия таблицы может быть также сделана командой

```
COPY/TABLE in-table out-table
```

где **in-table** и **out-table** — имена соответственно оригинальной таблицы и ее копии. Если дать эту команду после команды **SELECT**, будет создана новая таблица, являющаяся частью исходной (а не та же таблица с выделенными строчками).

Команда

```
MERGE/TABLE table1 [table2 ...] out-table
```

объединяет общие столбцы (с одинаковыми метками) одной или нескольких таблиц в таблицу **out-table**. Эта таблица будет содержать все те же столбцы, что и первая, исходная таблица **table1**, но дополненные значениями из остальных таблиц **table2**, ...

Еще одна полезная команда

```
PROJECTION/TABLE in-table out-table columnselection
```

позволяет выбрать из входной таблицы столбцы, определенные в **columnselection,** и создать новую таблицу.

Например:

```
PROJECTION/TABLE bigtable smalltable #2
```

создает таблицу **smalltable** с одной, второй колонкой таблицы **bigtable**.

8. Операции со столбцами

Арифметические операции со столбцами таблицы могут быть выполнены командой

```
COMPUTE/TABLE table column = expression
```

где **table** — имя таблицы, **column** означает столбец (существующий или новый, создаваемый командой, причем в последнем случае может быть использована только метка столбца, но не его номер), в который помещаются результаты, **expression** — фортраноподобное выражение, в котором переменными являются указатели на столбцы таблицы.

В этих выражениях можно использовать все арифметические операции и математические функции, упомянутые выше в п.6. Заметим, что команда применима и в случае, когда тип данных в столбце — строка (character).

Поскольку команда играет важную роль, но взаимодействует с командой **SELECT** не всегда очевидным образом, мы приведем несколько иллюстрирующих примеров из лекций по MIDAS А.Князева (лекции на русском языке доступны в библиотеке Астрономического института СПбГУ):

```
SELECT/TABLE cfa :mag .LT. 15 .AND. :vel .GT. 1000
```

отмечает в таблице **cfa.tbl**, содержащей некоторый каталог галактик, все строчки с объектами ярче 15-й звездной величины и со скорость выше 1000 км/с;

```
COMPUTE/TABLE cfa :z = min(:mag,:r) + (1 – SEL) *
:vel
```

создает новый столбец, содержащий минимальные значения из двух столбцов **(:mag и :r)** для всех строчек, но добавляет к этому значение скорости для отмеченных (предыдущей командой) строчек;

```
COMPUTE/TABLE cfa :surf_bright = 2.5 *
log10(10**((0.4 * :mag) / (:d**2)))
```

вычисляет среднюю поверхностную яркость для всех галактик (поскольку предыдущая команда удаляет любое выделение!).

9. Специальные операции с таблицами

В MIDAS реализован ряд наиболее часто выполняемых действий с данными, организованными в виде таблиц.

Простой статистический анализ может быть произведен командой

```
STATISTICS/TABLE table column
```

где **table** и **column** имеют обычный смысл.

В результате действия этой команды будет выдана информация о минимальном, максимальном и среднем значении в данном столбце (и выделенных строчках), а также о дисперсии. Эти данные будут сохранены в глобальном системном ключевом слове **OUTPUTR**, которое затем может быть использовано в процедурах пользователей.

Набор команд с квалификатором **HISTOGRAM** позволяет представить значения в столбце в виде гистограмм, например

```
PLOT/HISTOGRAM table column
READ/HISTOGRAM table column
```

где результаты выполнения команд очевидны.

Линейные и степенные аппроксимации по одной или двум переменным могут быть произведены командой **REGRESSION** с квалификаторами **LINEAR** и **POLYNOMIAL** соответственно. Например,

```
REGRESSION/POLYNOMIAL table y x1[,x2] de-
gree1[,degree2]
```

где **y** означает столбец зависимой переменной (значения функции), **x1** и **x2** — столбцы независимых переменных (аргументы), **degree1** и **degree2** — наибольшие степени аппроксимирующего полинома по осям *x1* и *x2* соответственно.

Коэффициенты аппроксимирующего полинома сохраняются в глобальном вещественном ключевом слове **OUTPUTD**, остальная информация в глобальных ключевых словах **OUTPUTR** (имеет иную структуру, чем в команде **STATISTICS/TABLE** !), **OUTPUTI** и **OUTPUTC** (более подробно см. **HELP** данной команды).

Часто бывает желательно вычислить значения аппроксимации и поместить их в отдельный столбец таблицы (скажем, чтобы позднее нанести на рисунке и исходные данные, и их аппроксимацию). Это может быть сделано следующей группой команд:

```
REGRESSION/POLYNOMAIL table :z :x,:y 2,3
SAVE/REGRESSION table test
COMPUTE/REGRESSION table :my_fit = test
```

которая аппроксимирует двумерную функцию *z(x,y)*, содержащуюся в таблице **table**, полиномом с максимальной степенью *x*, равной 2, и степенью *y*, равной 3; т. е. $p(x, y) = ax^2y^3 + bx^2y^2 + cxy^3 + ...$

Команда

```
SORT/TABLE table column
```

изменяет положение всех строчек в таблице **table** так, чтобы значения в столбце **column** оказались в возрастающем порядке.

В MIDAS реализовано также несколько более специфичных операций с таблицами (см., например, **REBIN, INTERPOLATE** и т. п.).

10. Форматирующие файлы

Зачастую бывает полезно, а иногда даже необходимо вместе с командой **CREATE/TABLE** использовать специальный форматирующий MIDAS-файл (table format file). Такие файлы имеют расширение **.fmt** и содержат следующую команду для *каждого* столбца:

```
DEFINE/FIELD pos1 pos2 type [format] label [unit]
```

где **pos1** и **pos2** — позиции начала и конца поля (данного столбца); **type** — тип **R**, **D**, **I** или **C**; **format**, **label** и **unit** имеют тот же смысл, что и в команде **CREATE/COLUMN** (см. подробнее п. 2 выше).

Например, для уже использовавшего выше файла **TNK.dat**, содержащего

```
1 1.0 9
3 4.0 50
2 5 7
05 6 23
4 7 0
```

и файла **TK.fmt**, имеющего 2 строчки:

```
DEFINE/FIELD 1 3 I I3 :TIME "sec"
DEFINE/FIELD 4 9 R F5.2 :FLUX "Jy"
```

команда

```
CREATE/TABL t6a 2 4 TNK.dat TK.fmt
```

создаст таблицу **t6a** (соответственно файл **t6a.tbl**), который будет хранить следующую информацию:

```
Table : t6a

Sequence TIM FLUX
-------- --- -----
1 1 1.00
2 3 4.00
3 2 5.00
4 5 6.00
5 4 7.00
-------- --- -----
```

Сравните этот результат с приведенным в п.3.1 (полученным без форматирующего файла) и отметьте различия в метках столбцов и форматах данных, появившиеся из-за использования форматирующего файла.

11. Список команд для работы с таблицами

Большинство таких команд приведены в нижеследующем списке. Более подробную информацию об этих командах и их параметрах можно получить, используя **HELP command/qualifier .**

```
COMPUTE/TABLE table column = expression

COMPUTE/REGRESSION table column = name[(ind-vars)]

COMPUTE/HISTOGRAM image = table column

COMPUTE/HISTOGRAM table/TABLE = table column

CONVERT/TABLE image = table indv[,indv] depv refima
method [par]

COPY/KT keyword table column row

COPY/TK table column row keyword

COPY/TI in-table out-table

COPY/IT in-image out-table

COPY/TT in-table column [out-table] column
```

```
COPY/TABLE in-table out-table

CREATE/COLUMN table column [unit] [format] [type]

CREATE/TABLE table ncol nrow filename [formatfile]

DELETE/COLUMN table column [...]

EDIT/TABLE table [ncol nrow]

IDENTIFY/CURSOR table identifier x [y] [tolerance]

IDENTIFY/GCURSOR table identifier x [y] [tolerance]

INTERPOLATE/IT out-table i,d in-image [s] [degree]

INTERPOLATE/TI out-image in-table refima [s] [de-
gree]

INTERPOLATE/TT out-table i,d in-table i,d [s] [de-
gree]

JOIN/TABLE tab1 col1,col2 tab2 col1,col2 outtab
tol1,tol2

LOAD/TABLE table col1 col2 [ident] [symbol] [size]
[color] [flag]

MERGE/TABLE table1 [table2 ...] out-table

NAME/COLUMN table column [column] [unit] [format]

OVERPLOT/HISTOGRAM tab col [offset] [log] [opt]
[bin[,min[,max]]] [exc] [log] [opt]

OVERPLOT/TABLE tab col1 col2
[x_sc,y_sc[,x_off,y_off]] [symbols] [lines]
[flag_dir]

PLOT/HISTOGRAM tab col [x_sc,y_sc[,x_off,y_off]]
[bin[,min[,max]]] [exc] [log] [opt]

PLOT/TABLE table column1 column2 [sc-x,sc-y]

PRINT/HISTOGRAM table column [bin [min [max]]]

PRINT/TABLE table [column1 ...] [row1 [row2]] [file
[format]]

PROJECTION/TABLE in-table out-table column [column
...]
```

```
READ/HISTOGRAM table column [bin [min [max]]]
READ/TABLE table [column1 ...] [row1 [row2]] [for-
mat]
REBIN/IT out-table i,d[,b] in-image [func] [parm]
[intop]
REBIN/TI out-image in-table i,d[,b] refima [func]
[parm] [intop]
REBIN/TT out-table i,d[,b] in-table i,d[,b] [func]
[parm] [intop]
REGRESSION/LINEAR table dep-var ind-var1[,ind-var2]
REGRESSION/POLYNOMIAL table dep-var ind-var1[,ind-
var2] degree1[,degree2]
RETRO/TABLE table
SAVE/REGRESSION table name
SELECT/TABLE table logical-expression
SET/REFCOLUMN table column
SHOW/TABLE table
SORT/TABLE table column
STATISTICS/TABLE table column
WRITE/TABLE table column row value
```

Упражнения

Написать процедуру, имеющую 5 параметров (**p1, p2, p3, p4, p5**) и выполняющую следующие действия:

0) перевести данные из файла **TAB1.dat** в MIDAS-таблицу **TAB1.tbl**, используя форматирующий файл **TAB1.fmt**. Файлы **TAB1.dat** и **TAB1.fmt** должны быть созданы заранее и содержать: первый — две следующие строчки

```
0 0
3.14159265 3.14159265
```

второй, форматирующий файл должен включать команды, необходимые для преобразования первого файла в MIDAS-таблицу.

Учесть, что названия столбцов таблиц **TAB1.tbl** и **TAB2.tbl** (см. ниже) должны совпадать;

1) создать таблицу **TAB2.tbl** с двумя столбцами: **TIME** (в днях) и **FLUX** (в эрг/с/см^2);
2) поместить в столбец **TIME** числа от **p1** (по умолчанию 0.5) до **p2** (9.5) с шагом **p3** (0.5);
3) поместить в столбец **FLUX** значения, равные **TIME** + **sin(TIME*pi/180*p4)**, где **p4** — параметр (по умолчанию 3.0);
4) в первой строчке поместить особое значение **FLUX = -p5**, где **p5** — параметр (по умолчанию 5.0);
5) объединить таблицы **TAB1.tbl** и **TAB2.tbl** в одну с именем **TAB.tbl**;
6) упорядочить строчки в таблице **TAB.tbl** так, чтобы значения в столбце **TIME** оказались в возрастающем порядке;
7) выполнить 2 линейные аппроксимации: первую для всех точек в **TAB.tbl**, вторую — для всех точек за исключением отрицательного значения **FLUX** (используйте команду **SELECT**);
8) поместить значения, соответствующие этим аппроксимациям, в столбцы **FIT1** и **FIT2**;
9) нанести на один график прямые, показывающие обе аппроксимации (одну — пунктиром, другую — непрерывной прямой), а также точки, соответствующие значениям FLUX;
10) исключить столбец **FIT2**;
11) напечатать на экране содержимое таблицы **TAB.tbl**.

IV. АППРОКСИМАЦИЯ В MIDAS

0. Введение

Аппроксимация сложной функции/зависимости более простой функцией — задача, которая часто встречается в различных науках, включая, конечно, и астрономию.

Напомним формулировку этой задачи. Предположим, есть функция $y = f(x)$, заданная в форме таблицы (значения x и y), и некая выбранная для аппроксимации функция $f_a(x, a, b, ...)$, имеющая набор параметров ($a, b, ...$) в дополнение к независимой переменной x. Требуется найти значения параметров, при которых минимальна разность табличных значений и значений аппроксимирующей функции в тех же узлах.

Существует много методов решения данной задачи, и выбор наиболее подходящего способа зависит от ряда аспектов. Более подробно ознакомиться с проблемой можно в учебниках по методам вычислений.

Ранее мы уже рассматривали аппроксимацию значений, заданных в столбцах таблиц, *степенной* функцией, применяя команду **`REGRESSION/POLYNOMIAL`**. MIDAS дает возможность пользователям аппроксимировать значения, данные в MIDAS-таблице или изображении (image), используя широкий набор *нестепенных* функций.

1. Аппроксимирующие функции

Для аппроксимации в MIDAS можно использовать *линейные комбинации* функций. Последние могут быть как стандартными функциями, «встроенными» в MIDAS, так и функциями, определенными самим пользователем.

1.1. Стандартные аппроксимирующие функции

POLY(X;A,B,C,...) $= a + bx + cx^2 + ...$
POLY(X,Y;A,B,...) $= a + bx + cy + ...$
LOG(X;A,B,C) $= a \ln(b + cx)$
EXP(X;A,B,C) $= a \exp(b + cx)$
SIN(X;A,B,C) $= a \sin(b + cx)$
TAN(X;A,B,C) $= a \tan(b + cx)$
GAUSS(X;A,B,C) $= a \exp\{-\ln 2\,[2(x-b)/c]^2\}$
LORENTZ(X;A,B,C,D) $= a\,\{1 + [2(x-b)/\mathrm{c}]^2\}^{-d}$

и так далее.

1.2. Аппроксимирующие функции, определяемые пользователем

Эти функции представляются обычными *фортрановскими* подпрограммами, написанными и оформленными по *специальным правилам*. Ниже приведен пример такой подпрограммы, ее использование осуждается ниже, в п.2).

```
======================================================================
C+
C.NAME
C USER00
C
C.DESCRIPTION
C One dimensional polynomial of NPAR degree.
C
C.INPUT ARGUMENTS:
C NIND INTEGER Number of independent variables
C  X (NIND) REAL Array with values of these vari-
   ables
C NPAR INTEGER Number of parameters
C  PARAM (NPAR) DOUBLE PRECISION Array with values
   of the parameters
C
C.OUTPUT ARGUMENTS:
C Y DOUBLE PRECISION Value of the function
```

```
C DERIV (NPAR) DOUBLE PRECISION Array with values
  of derivatives
C-
SUBROUTINE USER00 (NIND,X,NPAR,PARAM,Y,DERIV)
IMPLICIT NONE
C
INTEGER NIND,NPAR,I
DOUBLE PRECISION Y
C
REAL X(NIND)
DOUBLE PRECISION PARAM(NPAR),DERIV(NPAR)
C
Y = 0.0d0
DO I = 1, NPAR
Y = Y + PARAM(I) * X(1)**(I-1)
DERIV(I) = X(1)**(I-1)
ENDDO
C
RETURN
END
```

При этом следует использовать:

1) только следующие имена фортрановских подпрограмм: **USER00**, **USER01**, ..., **USER09**;
2) соответствующие имена, содержащих их файлов: **user00.for**, **user01.for**, ..., **user09.for**;
3) фиксированное число, порядок и тип входных/выходных параметров;
4) никаких операторов **IMPLICIT**, лишь циклы **DO**—**ENDDO** и так далее.

Команда

```
CREATE/FUNCTION func
```

где **func** — это **user00**, **user01**, ... или **user09**, транслирует фортрановскую подпрограмму, находящуюся в файле **func.for,** и добавляет ее в *библиотеку стандартных аппроксимирующих функций* MIDAS (см. п. 1.1).

2. Выбор аппроксимирующей функции

Команда

```
EDIT/FIT name
```

открывает *встроенный редактор* и предлагает модифицировать *fit-файл* с именем **name** (подобные файлы имеют расширение **.fit**). Если файл **name.fit** не существует, команда создает его. На экране будут видны примерно следующие строки:

```
EDIT/FIT                                12-APR-1984 JDP

                        name_fit
Sequence?FUNCTIONS                                  ?
??????????????????????????????????????????????????.......
        1?_                                         ?
        2?                                          ?
        3?                                          ?
        4?                                          ?
        5?                                          ?
        6?                                          ?
        7?                                          ?
        8?                                          ?
        9?                                          ?
        10?                                         ?
...
```

В каждой строке можно записать одну функцию — или стандартную, или определенную пользователем. Таблица заполняется, используя клавиши, управляющие движением курсора, примерно так же, как при редактировании таблиц в команде **EDIT/TABLE** (см. III.3.2). Выход из редактора происходит после одновременного нажатия клавиш **Ctrl** и **z**. Возможны и подкоманды, аналогичные упомянутым в п.III.3.2.

Строки таблицы соединены невидимыми знаками плюс, что создает линейную комбинацию функций. Коэффициенты этой комбинации должны быть включены непосредственно в функции. Например, линейная комбинация

```
F(x;a,b,c) = a*sin(-b*x) + 3*exp(x/c+b)
```

может быть записана в fit-файл **my_fit** как

```
1?SIN(X;A1,B1,C1)
2?EXP(X;A2,B2,C2)
```

где подразумевается, что **A1** = a, **B1** = 0, **C1** = $-b$, **A2** = 3, **B2** = b, **C2** = $1/c$ (обратите внимание на **A1** и **A2**). Заметьте, что в скобках должны быть указаны все параметры функций (даже тождественно равные 0), причем не числами, а символами (о присвоении символам числовых значений см. ниже).

Редактор позволяет также задать начальные значения свободных параметров, обычно необходимые для поиска их оптимальных значений. Нажмите несколько раз клавишу, сдвигающую курсор вправо. Когда он покинет последний символ, записанный вами в строке, произойдет несколько событий:

i) заголовок **FUNCTIONS** будет замещен на **PARAMETERS**;
ii) функция, которую вы набрали, исчезнет, и курсор переместится в начальную позицию.

Теперь можно задать начальные значения параметров, например для нашей комбинации функций **F(…):**

```
1?A1=a B1=0.@ C1=-b
2?A2=3.@ B2=C1 C2=d
```

где **a**, **b**, **c** — некоторые выбранные вами числа, а **d** = 1/**c**.

Отметим два очень важных момента:

1. Соотношение **A2=C1** означает, что **A2** и **C1** *не являются независимо варьируемыми* параметрами (см. выражение для **F(x;a,b,c)**, данное выше, где **b** входит в обе функции).
2. Символ **@** указывает на то, что число перед ним является *константой* и данный параметр *не будет варьироваться* при поиске наилучшего приближения.

Аналогично используются функции, определенные пользователем. Например, можно записать

```
1?USER00(X;A,B,C)
```

и продолжать далее как для стандартных функций. Заметим, что для функции **USER00**, определенной выше, такая строка будет автомати-

чески означать присвоение NIND=1 и NPAR=3 и, следовательно, следующую аппроксимирующую функцию:

```
USER00(X;A,B,C) = a + b*x + c*x*x
```

3. Опции аппроксимации

Когда аппроксимирующая функция (или комбинация) составлена, можно, если необходимо, переопределить некоторые опции MIDAS. Это делается командой

```
SET/FIT METHOD=method PRINT=n WEIGHT=w FUNCT=f FCTDEF=u
```

где опции/параметры обычно задаются явно, в форме равенств ОПЦИЯ=значение (а не как в большинстве примеров команд выше — в определенном порядке и без названия параметра. Впрочем, в MIDAS эти способы задания параметров эквивалентны). Наиболее важные опции перечислены в таблице.

Опция	Значение	Ломментарии
METHOD	определяет метод решения численной проблемы (аппроксимации)	**method**=**NR** — метод Ньютона—Рафсона (иные могут не работать в АИ СПбГУ)
PRINT	печать промежуточных результатов после **n** итераций	**n** — любое целое число
WEIGHT	выбор весовой функции (см. подробнее **HELP** команды)	**w**=**S** — статистические веса **w**=**C** — постоянные и т. д.
FUNCT	указывает аппроксимирующую функцию	**f** — имя соответствующего fit-файла, содержащего нужную функцию
FCTDEF	использование функций, определенных пользователем	**u**=**USER**, если есть; **u**=**SYST**, если нет

Например, для fit-файла **my_fit**, рассмотренного выше (см. п. 2), команда

```
SET/FIT METHOD=NR PRINT=1 WEIGHT=C FUNCT=my_fit
```

определит соответствующим образом указанные опции.

Значения опций можно просмотреть командой

```
SHOW/FIT
```

4. Процесс подгонки

Для того, чтобы выполнить подгонку функции, представленной таблицей, следует дать команду

```
FIT/TABLE nfeval[,prec[,metpar]] table depcol indcol
```

где **nfeval** — максимальное число итераций (используемых в методе Ньютона, если **method=NR**), **prec** — точность значений параметров, которая должна быть достигнута, **metpar** — дополнительный параметр (например, в методе **NR** — релаксационный множитель — по умолчанию он равен 1, но для улучшения сходимости может быть меньше — 0.3–0.9), **table** — имя MIDAS-таблицы, **depcol** и **indcol** — столбцы со значениями зависимой переменной (функции) и независимой переменной (аргументы), соответственно. Например,

```
FIT/TABLE 30,0.001 my_table :flux :lambda
```

выполнит подгонку значений, заданных в столбце **:flux**, и определит параметры аппроксимации с относительной точностью 0.001 (если это возможно за 30 итераций).

Отметим, что процесс подгонки далеко не всегда сходится. Иногда, если аппроксимирующая функция нелинейная и достаточна сложная, то для сходимости процесса вычислений необходимо задавать (фактически угадывать!) начальные значения ее параметров с точностью порядка 10% от искомых значений. Этот факт является причиной многих затруднений при создании аппроксимаций.

5. Значения аппроксимирующей функции

Итак, мы определили коэффициенты аппроксимирующей функции. Теперь требуется сопоставить эту функцию с первоначальной, аппроксимируемой. И чтобы вычислить значения аппроксимирующей функции, воспользуемся командой

```
COMPUTE/FIT table outcol = f(indcol)
```

где **table** — имя соответствующей MIDAS-таблицы, **f** — имя fit-файла (аппроксимирующая функция), **indcol** — столбец с независимой переменной (аргументы), **output** — (новый) столбец со значениями аппроксимирующей функции. Отметим, что между именем **f** и скобкой **(** не должно быть пробела.

Заметим, что если в fit-файле задана (линейная) комбинация функций, можно выбрать для вычисления отдельные ее компоненты. Например, для fit-файла **my_fit** команды

```
SELECT/FUNCTION my_fit 2
COMPUTE/FIT my_table :approx = my_fit(:x1)
```

вычислят значения второй функции в **my_fit.fit** (т. е. **EXP(X;A2,B2,C2)**) для значений параметров (**A2**, **B2**, **C2**), определенных последней процедурой подгонки (командой **FIT/TABLE**).

6. Tutorial для FIT

Учебник (tutorial), который может быть запущен (желательно в графической моде Linux) командой

```
TUTORIAL/FIT
```

является весьма полезной иллюстрацией для рассмотренного материала. Однако необходимо сделать несколько пояснений для лучшего понимания того, что произойдет и отобразится на экране.

Tutorial демонстрирует, как аппроксимировать одномерное изображение (спектр), состоящее из двух перекрывающихся гауссиан, наложенных на нелинейный фон с шумом. Заметим, что аппроксимация изображений (image) и MIDAS-таблиц (table) таблиц похожа за исключением нескольких несущественных деталей.

После старта tutorial две функции копируются в вашу текущую директорию: **TEST** — для генерации искусственного изображения и **FUNCTION** — с моделью для подгонки. Искусственное изображение создается следующим образом. Сначала определяется изображение **REF**, чтобы задать пределы изменения независимых переменных. Затем команда **COMPUTE/FUNCTION** создает фрейм (image) с гауссовскими профилями, наложенными на фон. И, наконец, добавляется некоторый шум (случайная функция). Полученное изображение **PROFILE** показывается на экране.

Изображение **PROFILE** аппроксимируется модельной функцией **FUNCTION**. Соответствующий fit-файл копируется в вашу директорию. Этот файл можно просмотреть и изменить, используя команду для редактирования fit-файлов

```
EDIT/FIT FUNCTION
```

следующим образом:

```
1 GAUSS(X;A1,A2,A3) A1=50. A2=95. A3=45.
2 GAUSS(X;A4,A5,A6) A4=A1 A5=135. A6=A3
3 POLY(X;A,B,C) A=0. B=0. C=0.
```

где два параметра второй гауссианы (высота и FWHM) связаны с соответствующими параметрами первой.

Различные методы подгонки применяются после изменения опций командой **SET/FIT**. Если иные методы, кроме **NR**, работают в ваших версиях MIDAS, то можно сравнить их эффективность, точность и т. п.

Наконец, значения аппроксимации вычисляются и наносятся на график поверх оригинальных данных. Отдельные компоненты этой аппроксимации (одна гауссиана + фон) также затем показываются на этом графике.

7. Список команд, связанных с аппроксимацией

В этом списке вы найдете некоторые команды, которые не обсуждались выше. По именам этих команд (и параметров) легко догадаться об их назначении. Использование команды **HELP** позволит получить подробную информацию.

```
COMPUTE/FIT outima [= funct[(refima)]]

COMPUTE/FIT table :out[,:error] [=
    funct[(:col1,...)]]

COMPUTE/FUNCTION outima = funct(refima)

COMPUTE/FUNCTION table :out = funct(:col1,...)

CREATE/FUNCTION userfunct1[,...]

EDIT/FIT [funct]

FIT/IMAGE [nfeval[,prec[,metpar]]] [image[,wgt]]
    [funct]

FIT/TABLE [nfeval[,prec[,metpar]]] table dep[,wgt]
    ind [funct]

MODIFY/FIT table seqno [funct]

REPLACE/FUNCTION userfunct1[,...]
```

```
SAVE/FIT table seqno [funct]
SELECT/FUNCTION funct number[,...]
SELECT/FUNCTION funct ALL
SET/FIT [METHOD=mname] [PRINT=iter]
    [WEIGHT=wgttype] [FUNCT=fname]
[FCTDEF=where]
SHOW/FIT
```

Упражнения

1) Написать процедуру, создающую MIDAS-таблицу, содержащую 2 столбца и 100 строк. В первый столбец (x) должны быть занесены все числа от 1 до 100; во второй — значения следующей функции:

$y(x) = 40\,[\exp(-\ln 2\,(2\,(x - \mathbf{p1})/\mathbf{p2})^2) + \exp(-\ln 2\,(2\,(x - \mathbf{p3})/\mathbf{p2})^{**}2)] + \mathbf{p4}^*x + \mathbf{p5}^*x^2 + \mathbf{p6}^*\sin(60x)$

где **p1**, ..., **p6** — параметры процедуры. Задайте по умолчанию: **p1** = 60.01, **p2** = 5.01, **p3** = 80.01, **p4** = 1.01, **p5** = 0.01, **p6** = 0.

Замечание: При вычислении *y(x)* в процедуре можно разбить данное выражение на несколько частей и присвоить их значения локальным ключевым словам, затем уже их суммы поместить в столбец.

2) Создать fit-файл, содержащий две гауссианы (стандартные аппроксимирующие функции **GAUSS**) и полином 2-й степени (**POLY**), а также начальные значения параметров, близкие к заданным по умолчанию в процедуре из п.1, но не совпадающие с ними.

3) Написать еще одну процедуру (можно объединить ее с процедурой из п.1), которая производит необходимую настройку опций MIDAS, выполняет подгонку значений, данных в столбцах таблицы, созданной в п.1, и показывает на графике исходные данные (точками) и их аппроксимацию (кривой). Параметры процедуры — точность определения параметров и далее по вашему усмотрению.

4) Сравнить параметры аппроксимирующей функции (**p1**, **p2**, ..., **p5**), полученные для двух значений **p6**: 0 и 0.1 (начальные значения остальных параметров задать как в п.1).

V. СТРУКТУРЫ ДАННЫХ В MIDAS

0. Введение

Полный список структур данных, используемых в MIDAS, таков:

— **изображения** или **фреймы** (images, frames)
— **таблицы** (tables)
— **fit-файлы** (fit-files)
— **каталоги** (catalogs)
— **дескрипторы** (descriptors)
— **ключевые слова** (keywords)

Некоторые из этих структур (ключевые слова, таблицы, fit-файлы) уже рассматривались нами ранее, все остальные будут обсуждаться ниже. Перед этим, однако, настоятельно рекомендуется прочитать определения структур, приведенные в нашем мини-глоссарии (см. *Приложение Г)*.

А. ИЗОБРАЖЕНИЯ

1. Спектры и многомерные изображения

Результатами наблюдений часто бывают одно-, двух- или трехмерные массивы чисел. В MIDAS они называются изображениями или фреймами и являются основным типом данных, которые обрабатываются системой.

2. Структура изображений

Изображение (или фрейм) в MIDAS имеет *заголовок* и одно-, двух- или трехмерный *массив элементов*.

Заголовок содержит некоторые дескрипторы (подробнее см. ниже), а все *элементы* массива должны иметь один и тот же тип. MIDAS-изображения обычно хранятся на диске в виде бинарных файлов с расширением **.bdf** (по умолчанию).

3. Доступ к элементам

К элементам изображения можно обращаться так же, как элементам массива, например элемент трехмерного изображения **my_frame** записывается как

```
my_frame[x1,y1,z1]
```

где **x1**, **y1** и **z1** — числа (индексы), характеризующие положение элемента массива (может быть как номер столбца, строки и т. п., так и вещественное число, когда используются физические координаты — см. ниже п.4).

Прямоугольная часть изображения обозначается, например, так:

```
my_frame[x1,y1,z1:x2,y2,z2]
```

где **x1, y1**, **z1** и **x2**, **y2**, **z2** — индексы, описывающие положение левого нижнего и правого верхнего углов выбранной области. Возможна также эквивалентная запись с использованием в качестве разделителя двух точек ..

```
my_frame[x1,y1,z1..x2,y2,z2]
```

Специальные символы **<** и **>** обозначают начальное и конечное значения индексов для выбранной оси изображения. Например,

```
my_frame[<,2,1:2,>,<]
```

представляет собой часть изображения **my_frame** с координатами (*x, y, z* приняты как обозначения осей) в интервалах: *x* от 1 до 2; *y* от 2 до последнего значения, *z* от 1 до 1.

4. Типы координат

Оси MIDAS-изображений могут быть связаны с двумя системами координат: 1) с системой, непосредственно привязанной к (исходному) массиву *пикселов (точнее к устройству, на котором этот массив получен)* и использующей соответствующие номера столбцов, строк и т. д. и 2) с системой, не зависящей от приемного устройства, имеющей определенный физический смысл и называемой в MIDAS «*мировыми*» (world) координатами.

В *«пиксельной»* системе, следует ставить символ **@** перед координатами, так

```
my_frame2[@1:@5]
```

означает первые 5 элементов изображения **my_frame2**. Заметим, что такие координаты (числа после **@**) должны быть целыми.

В *мировых* координатах MIDAS использует два системных дескриптора, имеющие имена **START** и **STEP** и ассоциированные с конкретным изображением. Значения координат вычисляются по формулам

```
world = START + STEP * (@i – 1)
```

где **@i** — координата элемента по данной оси в пиксельной системе. Заметим, что мировые координаты могут быть целыми и не целыми (вещественными). Например, при выборе для **my_frame2** значений **START** = 120 (скажем, нанометров) и **STEP** = 0.5 (нанометра) **my_frame2[120:122.5]** будет также означать первые 5 элементов. Заметьте отсутствие **@** для мировых координат.

5. Основные команды при работе с изображениями

5.1. Создание и модификация изображений

Любой элемент изображения может быть модифицирован, используя конструкцию

```
frame[x,y,z] = expression
```

где **x**, **y**, **z** — координаты элемента. Выражение **expression** может содержать арифметические действия и некоторые функции (подобные выражения использовались нами при работе с MIDAS-таблицами — см. III.6). Например

```
tt[@10,@10] = 5.0
```

где все пробелы обязательны.

Другая возможность сделать нечто подобное предоставляется командой

```
WRITE/IMAGE frame [pix_specs] data
```

где **frame** — имя модифицируемого изображения, **pix_specs** — набор чисел, определяющих положение элемента, **data** — набор значений, разделенных запятыми.

В частности, команда

```
WRITE/IMAGE m_f [<,@3:@2,4] 1.,3.,2.,33.
```

должна привести к следующим значениям: **m_f[1,3] = 1.0; m_f[2,3] = 3.0; m_f[1,4] = 2.0; m_f[2,4] = 33.0.** Для того, чтобы увидеть широкие возможности данной команды, обратитесь к **HELP WRITE/IMAGE**.

Еще более мощной является команда

```
COMPUTE/IMAGE frame = expression
```

которая вычисляет выражение **expression** и присваивает результат изображению с именем **frame**. Выражение может иметь до 21 операнда, которые могут быть *функциям*, *изображениями* и/или *константами* в обычной алгебраической трактовке.

Например,

```
COMPUTE/IMAGE r = sqrt(c+5.-log10(b))+abs(aa)
```

вычисляет элементы изображения **r.bdf** согласно приведенному выражению, включающему элементы изображений **c.bdf**, **b.bdf** и **aa.bdf**. И еще один пример

```
COMPUTE/IMA EXP(20./3.4)+SIN(1.2)
```

где вычисляется выражение, и его результат запоминается в стандартном ключевом слове OUTPUTR(1).

Отметим, что изображения, используемые в подобных выражениях, должны иметь *одинаковые шаги* по осям (с относительной погрешностью 0.0001) и *начало* (с абсолютной точностью 0.05 от величины шага). Расчеты выполняются только для пересекающихся (общих) областей изображений!

5.2. Работа с частью изображения

Часть изображения может быть сделана новым изображением при помощи команды

```
EXTRACT/IMAGE out = in intval
```

где **out** и **in** — имена исходного и получаемого изображений, **intval** определяет (так же, как выше в п.3) левый нижний и правый верхний углы выделяемой области исходного изображения. Например:

```
EXTRACT/IMAGE aa = dd
    [23:43:30,18:21:20..23:43:40,18:21:50]
```

где из двумерного изображения **dd** выделяется область **aa**. Для обозначения выделяемой области используются мировые координаты, представленные в привычной астрономической нотации: прямое восхождение (часы, минуты, секунды, разделенные двоеточием) и аналогично склонение.

Следует отметить, что все команды MIDAS, в которых изображения могут быть параметром, позволяют адресоваться к части изображения и иным образом. Например,

```
COMPUTE/IMAGE my_frame = 0
```

записывает ноль во все элементы изображения **my_frame**, тогда как

```
COMPUTE/IMAGE my_frame[@2,@3:@50,@4] = 0
```

делает это только для соответствующей части этого изображения.

5.3. Просмотр изображений

Команда

```
READ/IMAGE frame
```

показывает на экране элементы изображения **frame** в форме таблицы.

В частности команды

```
READ/IMAGE gal @200,<,20
READ/IMAGE gal [@200,<:@219,<]
```

выведут на экран 20 пикселей изображения **gal.bdf**, начиная с 200-го по оси *x* и первого по оси y. Эти две формы записи команды равнозначны.

Команда

```
READ/IMAGE CURSOR
```

выведет на экран все пиксели в области, определенной курсором.

В большинстве случаев затруднительно рассматривать изображение в табличной форме, и графический способ представления является более предпочтительным. Команда

```
LOAD/IMAGE frame
```

открывает специальное графическое окно — дисплей изображений, отличное от того, которое появляется при работе с таблицами, и показывает в нем изображение **frame**. Заметим, что дисплей изображений нужных размеров можно создать (определить) командой **CREATE/DISPLAY** до использования команды **LOAD/IMAGE**.

Команды

```
PLOT/COLUMN frame [x_coord] [y_start,y_end]
PLOT/ROW frame [y_coord] [x_start,x_end]
```

дают сканы изображения **frame** для указанных координат (первая команда вдоль столбцов, вторая — вдоль строк) в графическом окне, которое используется при просмотре MIDAS-таблиц. Заметим, что это окно должно быть заранее открыто командой **CREATE/GRAPHICS**.

Добавим, что **x_coord** и **y_coord** — мировые координаты или номера столбцов и строк (по умолчанию первый столбец/строка); а **y_start,y_end** и **x_start,x_end** — интервалы по осям, которые будут показаны на рисунке.

5.4. Копирование/удаление изображений

Изображение может быть скопировано/перемещено/удалено командами Linux в среде MIDAS **$cp/$mv/$rm**. То же самое может быть выполнено и командами MIDAS.

Сделать копию изображения можно командой

```
COPY/II source_frame dest_frame dest_format flags
```

где **source_frame** и **dest_frame** — имена исходного и нового изображений, **dest_format** — формат данных, по которому они могут быть преобразованы. Возможные форматы: **I1** (байты), **I2** (16-битные целые), **UI2** (16-битные целые без знака), **I4** (32-битные

целые), **R4** (32-битные вещественные), **R8** или **D** (64-битные вещественные — двойная точность); по умолчанию формат — **R4**. Наконец, **flags** — некий признак.

Например

```
COPY/II gal galr R8 d
```

скопирует изображение **gal** в **galr**, преобразовав данные в числа двойной точности (64-бита). По признаку **d** после копирования изображение **gal** будет удалено.

Удалить изображение можно командой

```
DELETE/IMAGE frame
```

Для того чтобы переименовать изображение, используйте комбинацию двух приведенных команд.

5.5. Создание изображения

Обычно мы получаем изображения из наблюдений. В редких случаях, когда требуется искусственное изображение, оно может быть создано командой

```
CREATE/IMAGE frame [dim_specs] [frame_specs]
    [func_type] [coefs]
```

Здесь **frame** — имя нового MIDAS-изображения. Возможный параметр **dim_specs** — последовательность чисел, разделенных запятыми и представляющих собой значения системных дескрипторов **NAXIS**, **NPIX(1)**, ..., **NPIX(NAXIS)**, означающих число осей (от 1 до 3) и число пикселов по каждой из них.

Аналогично, **frame_specs** — последовательность чисел, разделенных запятыми и представляющих собой значения системных дескрипторов **START(1)**, ..., **START(NAXIS)**, **STEP(1)**, ..., **STEP(NAXIS)**, являющихся началом (START) и шагом (STEP) по каждой из осей всемирной системы координат.

Наконец, **func_type** — имя одной из поддерживаемых в MIDAS функций, которая может быть использована для создания искусственных изображений, **coefs** — параметры этой функции (подробнее о том, какие функции возможны и каковы их параметры, см. **HELP CREATE/IMAGE**).

В виде примера приведем команду

```
CREATE/IMAGE new 2,200,300 0.,1.25,1.1,0.3 ELLIPS
    50,70,20,7
```

которая создает двумерное изображение размером 200x300 (200 пикселов по одной оси и 300 пикселов по другой). Оси связаны с некоторыми мировыми координатами (предположим **x** и **y**) следующим образом:

```
x = 1.1 * (@i – 1), y = 1.25 + 0.3 * (@j – 1)
```

где **i** и **j** — значения соответствующих координат в пикселах. Эллипс (содержательная часть изображения, т. е. сигнал) будет иметь полуоси размером 50 и 70 пикселов; значения сигнала внутри эллипса 20 и вне его — 7.

6. Преобразование в таблицы

Изображение может быть переведено в MIDAS-таблицу (элемент в элемент) командой

```
COPY/IT inframe outable
```

где **inframe** и **outable** — имена исходного изображения и новой таблицы.

Преобразование в обратном направлении может быть выполнено командой

```
COPY/TI intable outimage
```

где **intable** и **outframe** — имена исходной таблицы и нового изображения.

7. Операции над изображениями

Существует множество команд, предназначенных для работы с изображениями в MIDAS. Следуя упоминавшимся лекциям А. Князева, разделим их на группы.

7.1. Работа с координатами

Набор команд для считывания координат изображения или определения центра:

```
CENTER/GAUSS CENTER/MOMENT GET/CURSOR GET/GCURSOR
```

7.2. Координатные преобразования

Программы преобразований: вращение, извлечение части и помещение ее в изображение, переход к новому шагу и т. п. —

ALIGN/IMAGE EXTRACT/IMAGE FLIP/IMAGE TRANSPOSE/IMAGE

GROW/IMAGE INSERT/IMAGE REBIN/II

REBIN/LINEAR REBIN/ROTATE REBIN/SPLINE REBIN/IT

ROTATE/CLOCK ROTATE/COUNTER_CLOCK ROTATE/1DIM XCORR/IMAGE

7.3. Арифметика

Программы, выполняющие арифметические операции над изображениями (включая усреднение):

AVERAGE/AVERA AVERAGE/COLUMN AVERAGE/ROW AVERAGE/IMAGES

AVERAGE/KAPPA AVERAGE/WEIGHT

COMPUTE/ROW COMPUTE/COLUMN COMPUTE/IMAGE COMPUTE/PIXEL

7.4. Фильтрация

Программы фильтрации различного рода по всему изображению или его части:

CONVOLVE/IMAGE DECONVOLVE/IMAGE CREATE/FILTER

FFT/IMAGE FFT/INVERSE FFT/POWER FFT/FREQ

FILTER/COSMIC FILTER/DIGITAL FILTER/GAUSS FILTER/MAX

FILTER/MIN FILTER/MEDIAN FILTER/SMOOTH FILTER/ADAPTIV

7.5. Создание изображений

Программы для создания новых изображений (в том числе и путем извлечения части из существующих):

CREATE/IMAGE CREATE/RANDOM EXTRACT/IMAGE EXTRACT/ CURSOR

EXTRACT/SLIT EXTRACT/CTRACE EXTRACT/RTRACE EXTRACT/ LINE

EXTRACT/TRACE EXTRACT/REFERENCE_IMAGE EXTRACT/ ROTATED_IMAGE

7.6. Преобразование пиксельных значений

Программы, модифицирующие значения (в областях, отмеченных курсором или выбранных на основе задаваемых критериев):

FIT/FLAT_SKY ITF/IMAGE MODIFY/CURSOR MODIFY/GCURSOR

MODIFY/PIXEL MODIFY/AREA REPLACE/IMAGE REPLACE/ POLYGON

7.7. Определение характеристик

Программы для получения характеристик анализируемых изображений:

FIND/MINMAX INTEGR/APERTURE INTEGR/LINE STATIST/ IMAGE

MAGNITUDE/CIRCLE MAGNITUDE/RECTANGLE

7.8. Аппроксимация

Программы подгонки имеющихся распределений различными функциями:

COMPUTE/FIT COMPUTE/FUNCTION EDIT/FIT FIT/IMAGE

FIT/TABLE READ/FIT SET/FIT SHOW/FIT

SELECT/FIT REGRESSION/POLYNOMIAL SAVE/REGRESSION

Информацию обо всех этих командах можно получить, используя команду **HELP**.

Б. MIDAS-ТАБЛИЦЫ

8. Таблицы и изображения

Структура таблиц в MIDAS и команды, выполняющие различные операции над ними, уже были рассмотрены нами ранее (см. раздел III). Здесь следует лишь добавить, что MIDAS-таблицы во многих аспектах подобны MIDAS-изображениям (структура, возможные дескрипторы, многие команды и т. п.). Знание этого факта позволяет лучше ориентироваться в системе MIDAS и ее многочисленных командах.

В. FIT-ФАЙЛЫ

9. Fit-файлы и изображения

Fit-файлы, их структура и назначение также обсуждались нами ранее (см. раздел IV). Часто говорят, что fit-файлы представляют собой «вырожденные» изображения, содержащие только дескрипторы без каких-либо данных. Действительно можно заметить некоторое подобие fit-файлов и изображений (некоторые команды, дескрипторы), однако различие структур настолько существенно, что идея об их подобии не представляется продуктивной.

Г. КАТАЛОГИ

10. Назначение и структура

Очень часто пользователь имеет группу сходных наблюдательных данных (например, изображения нескольких галактик или спектры звезд в каком-то поле, данные одного наблюдательного сета, для которых необходимо выполнить одну и ту же первичную обработку и т. п.). Такие данные можно объединить в MIDAS-каталог и затем запускать команды/процедуры только один раз сразу для всех данных.

Каталог в MIDAS представляет собой ASCII-файл, содержащий список включенных данных (изображений, таблиц или fit-файлов) и имеющий по умолчанию расширение **`.cat`**. В частности, записывается следующая информация о файлах, входящих в каталог: номер (**`No`**), имя файла (**`Name`**), идентификация типа файла (**`Ident`**), число осей (**`Naxis`**) и число пикселей (**`Npix`**).

11. Создание каталога

Команды

```
CREATE/ICAT catname [dir_spec] [descr]
CREATE/TCAT catname [dir_spec] [descr]
CREATE/FCAT catname [dir_spec] [descr]
```

создают каталог с именем **catname**, содержащий в зависимости от команды изображения/таблицы/fit-файлы, находящиеся в текущей директории. По умолчанию все файлы с расширением **.bdf/.tbl/.fit** включаются в каталог. Параметр **dir_spec**, представляющий собой опции LINUX-команды **$ls** (см. подробнее описание из Linux, выдаваемый в MIDAS командой **$man ls** или п.5 *Приложения А*), позволяет собирать в каталог не все, а только необходимые файлы. Например,

```
CREATE/TCAT my_cat ss*.tbl
```

поместит в каталог **my_cat** только MIDAS-таблицы с именами, начинающимися на **ss**, а

```
CREATE/ICAT galaxies log060906,:files
```

создаст каталог изображений **galaxies.cat**, имена которых записаны в столбце **:files** таблицы **log060906.tbl**.

Отметим, что ниже для краткости мы будет писать одну команду с квалификатором **xCAT** (где **x** = **I**, **T** или **F**) вместо трех версий команды с квалификаторами **ICAT**, **TCAT** и **FCAT**.

12. Изменение каталога

Новый объект добавляется в каталог командой

```
ADD/xCAT catname framelist
```

где **catname** — имя каталога, **framelist** — одно или несколько разделенных запятыми имен объектов, которые нужно включить в данный каталог.

Удалить объект из каталога можно, используя команду

```
SUBTRACT/xCAT catname framelist
```

где **catname** — имя каталога, **framelist** — одно или несколько разделенных запятыми имен объектов, которые нужно исключить.

Очень опасной является команда

```
DELETE/xCAT catname
```

поскольку удаляет (правда, после подтверждения) элементы (объекты) не только из каталога **catname**, но и из всей текущей директории диска!

Следующей командой можно сделать каталог *активным*, что позволит не указывать название каталога в последующих командах (по умолчанию они будут обращаться к этому активному каталогу)

```
SET/xCAT catname
```

где **catname** — имя каталога. Активность каталога отменяется командой

```
CLEAR/xCAT catname
```

В принципе, можно выполнять сортировку содержимого каталога по полю **Ident,** которое описывает файл в каталоге (см. выше п.10) командой

```
SORT/xCAT catname
```

и производить поиск объектов в каталоге по совпадению символов (не обязательно полному), заданных строкой **search_string**, в поле **Ident**,

```
SEARCH/xCAT catname search_string
```

Первое совпадение записывается в стандартное ключевое слово **OUT_A.**

13. Просмотр каталога

По аналогии с MIDAS-таблицами команда

```
SHOW/xCAT catname
```

дает общую информацию о каталоге с именем **catname**, а команда

```
READ/xCAT catname
```

показывает содержимое этого каталога.

14. Операции с каталогами

Основное достоинство каталогов в MIDAS заключается в возможности применять MIDAS-команды и процедуры сразу к группе сходных объектов. Это можно сделать командой

```
EXECUTE/CATALOG proc p1 p2 ... p7
```

где **proc** — имя MIDAS-процедуры (имя файла с расширением **.prg**) или MIDAS-команды, а **p1** ... **p7** — ее параметры (не более 7).

Часто прежде, чем применить эту команду, приходится определять несколько системных ключевых слов (например, **CATLI** и **CATLC**). О том, как это сделать, информирует команда

```
INFO/SETUP CATALOG
```

Используя полученные сведения, можно определить нужные ключевые слова при помощи команды

```
WRITE/SETUP CATALOG par1 par2 ...
```

назначение параметров **par1**, **par2** и т. д. также объясняются предыдущей командой.

В простых случаях можно избежать чтения online-документации. Например, если был создан каталог **my_cat**, включающий несколько таблиц, и требуется выполнить сходные вычисления для каждой из них (скажем, удвоить значения в столбце), то это можно реализовать командой

```
EXECUT/CATALOG COMPUTE/TABLE my_cat.cat :z = :x * 2
```

где **:x** и **:z** — метки соответствующих столбцов таблиц.

Д. ДЕСКРИПТОРЫ

15. Аналогия с ключевыми словами

Дескрипторы во многом сходны с ключевыми словами, которые мы рассматривали ранее (см. I.4-8). В частности, дескрипторы так же, как и ключевые слова, подобны фортрановским массивам и имеют *имя*, *тип* и *длину*. Мы разделим дескрипторы на две группы: *определяемые пользователем* и *системные*.

16. Дескрипторы пользователей

Имя такого дескриптора не должно превышать 72 символов.

Тип может быть *integer* (**I**), *real* (**R**), *double precision* (**D**) или *character* (**C**).

Длина ограничена 32767 элементами.

16.1. Как создать дескриптор

Это можно сделать командой

```
WRITE/DESCRIPTOR frame descr data
```

где **frame** — имя (файла) изображения/таблицы/fit-файла, **descr** — имя (нового) дескриптора с последующей информацией о нем *имя/тип/первый_индекс/послед_индекс* (мы встречали такую конструкцию при определении ключевых слов), **data** — значения, разделенные запятыми. Следует подчеркнуть, что для таблиц и fit-файлов необходимо давать имя соответствующего файла с расширением!

Например,

```
WRITE/DESCRIPTOR my_table.tbl DATE/I/1/4 03,04,2000
```

добавит дескриптор **DATE** (типа integer с 4 элементами DATE(1),DATE(2),...,DATE(4)) к MIDAS-таблице **my_table** и присвоит указанные значения первым трем его элементам.

Заметим, что все элементы дескриптора должны быть одного типа и нельзя, например, записать название месяца в **DATE** словом.

16.2. Как просмотреть дескриптор

Команды

```
SHOW/DESCRIPTOR frame [descr_list]
READ/DESCRIPTOR frame [descr_list]
```

действуют, как и с другими квалификаторами и показывают соответственно *общую информацию* и *содержимое* дескриптора из списка **descr_list** (имена дескрипторов, разделенные запятыми) или всех дескрипторов (если список отсутствует), ассоциированных с объектом **frame**.

16.3. Как удалить дескриптор

Это может быть выполнено командой

```
DELETE/DESCRIPTOR frame descr
```

с обычным смыслом **frame** и **descr**.

17. Дескрипторы системы

Список всех системных дескрипторов может быть получен командой

```
HELP [Descr]
```

данной, именно так, как она написана.

Имена этих дескрипторов не превышают 8 символов, и имена дескрипторов, определяемых пользователями, обязаны от них отличаться.

E. КЛЮЧЕВЫЕ СЛОВА

18. Ключевые слова, определяемые пользователем

Этот тип ключевых слов уже подробно рассматривался нами ранее (см. I.4-8). Здесь следует лишь еще раз отметить большое сходство ключевых слов и дескрипторов в MIDAS и обратить внимание на то, что пользователь не может называть свои ключевые слова именами ключевых слов системы MIDAS.

19. Системные ключевые слова

Список таких ключевых слов и информация об их назначении может быть получена по команде

```
HELP [Key]
```

данной именно так, как она написана.

20. Дескрипторы и ключевые слова

Дескрипторы и ключевые слова в MIDAS хотя и похожи, но все же различаются в ряде моментов. Начнем с того, что все дескрипторы *глобальны*, тогда как ключевые слова могут быть и *локальны* (зона действия ограничена одной процедурой).

Ключевые слова обычно используются для сохранения численной информации, которая обрабатывается позднее специальным образом. С другой стороны, информация, хранимая в дескрипторах, используется для описания (что отражено в самом названии этой структуры) некоторого объекта MIDAS (и часто является нечисло-

вой). Неудивительно, что дескрипторы *всегда связаны* с каким-то MIDAS-объектом, а ключевые слова — *никогда*.

Теперь становится понятно, почему мы обсуждали так много MIDAS-команд, имеющих дело с вычислениями, использующими ключевые слова, и рассматривали лишь создание, изменение и удаление дескрипторов.

Упражнения

Написать MIDAS-процедуру, которая:

1. создает MIDAS-таблицу размером 30 на 30 (со столбцами 0001, 0002, 0003 и т. д.) и заполняет ее значениями функции $F(i, j) = 2 + 3i + i*j + i^2/10 + j$, где i и j номера строк и столбцов;
2. копирует эту таблицу в MIDAS-изображение;
3. показывает изображение на экране (в графической моде);
4. создает каталог и включает в него данное изображение;
5. делает 2 копии изображения;
6. в одной из копий сначала меняет значение 36 элементов (от [1,1] до [6,6]) на 400.0, а затем изменяет значение первого элемента [1,1] на 1000.0;
7. показывает видоизмененное изображение в графической моде и в текстовой моде (достаточно первой строки), а также показывает эту строку на графике (используйте **`PLOT/ROW`**);
8. добавляет оба новых (скопированных) изображения в каталог;
9. удаляет из каталога первоначальное изображение;
10. показывает содержимое каталога;
11. создает дескриптор **`MY_DATE`** (с 3 элементами, содержащими какую-нибудь дату: день, месяц и год) во всех изображениях, входящих в каталог;
12. удаляет дескриптор в одном из изображений, входящих в каталог;
13. показывает значение дескриптора для всех изображений каталога.

Параметрами процедуры могут быть размер таблицы (30) и коэффициенты функции $F(i, j)$.

VI. ФОРМАТЫ ВВОДА/ВЫВОДА В MIDAS

0. Введение

Независимо от структуры все данные представляются в MIDAS в виде файлов в некотором *внутреннем* формате. Этот формат обычно включает заголовок, содержащий дескрипторы, и собственно цифровой массив. Имеется набор стандартных дескрипторов, которые обязательно присутствуют в заголовке. Без этих дескрипторов команды MIDAS не могут работать. Например, для изображений к таким необходимым дескрипторам относятся: **`NAXIS`** (число осей), **`NPIX`** (размерность осей), **`STEP`** (шаг по осям) и т. д. Внутренний формат MIDAS не меняется от одной операционной системы к другой, но представление чисел зависит от типа компьютера и операционной системы. Поэтому обычно рекомендуется наблюдательные данные и результаты их обработки хранить не во внутреннем формате MIDAS, а преобразовывать в специальный формат, служащий для хранения и распространения астрономических данных. Таким форматом является FITS (Flexible Image Transport System), рекомендованный IAU (International Astronomical Union) для использования в качестве стандарта для представления данных и обмена цифровой информацией между астрономическими учреждениями.

Кроме FITS, существуют и другие форматы, используемые при вводе/выводе данных в MIDAS: PostScript (PS), ASCII и т. д. Кратко рассмотрим основные форматы и связь между ними.

1. FITS-формат

FITS-формат используется научными организациями и правительственными агентствами для хранения астрономических изображений, полученных как с телескопов на спутниках, так и с

наземных телескопов. Формат поддерживается практически всеми средствами обработки астрономических данных и архивными системами.

Первоначально FITS-формат (basic FITS) разрабатывался для передачи изображений, т. е. цифровых массивов. Позднее были добавлены другие типы данных. Их правила описания и представления называются расширениями FITS-формата. К ним относятся:

— **`FITS random groups`** *(случайные группы данных)*, что используется для данных, состоящих из серии массивов (каждый из которых сопровождается набором связанных с массивом параметров);
— **`FITS tables`** *(таблицы)* — для представления ASCII-таблиц;
— **`FITS binary tables`** *(двоичные таблицы)* — для таблиц, в которых каждое значение ячейки таблицы может быть вектором.

Полезно помнить, в какие структуры MIDAS преобразуются различные представления данных в FITS-формате:

basic FITS → MIDAS-image
FITS-tables → MIDAS-table
BINTABLE (binary table) → MIDAS-table
random groups → MIDAS-image + MIDAS-table

1.1 Структура FITS-файлов

FITS-файл может состоять из последовательности блоков, каждый из которых включает *заголовок* и *группу данных*, называемую HDU (Header and Data Unit). В этом смысле структура блоков сходна со структурой файла, содержащего MIDAS-изображение.

Заголовок и данные записываются как логические записи длиной по 2880 байт. HDU включает целое число таких записей. Заголовок содержит описание данных, данные идут сразу после заголовка. HDU может повторяться несколько раз, а может не содержать данных (только заголовок). Структура расширений FITS-формата такая же.

Каждая логическая запись заголовка содержит 36 80-символьных ASCII-строк. В строке записывается: ключевое слово = значение/комментарий. Ключевое слово занимает позиции с 1 по 8 и выравнивается по левому краю. Допустимые значения символов в ключевом слове: цифры, латинские символы в верхнем регистре,

подчеркивание, дефис. Индикатор значения — знак равенства в 9-10 позиции, значение/комментарий помещаются в 11-80 позиции.

Следующие ключевые слова: **SIMPLE**, **BITPIX**, **NAXIS**, N**AXIS1**, …, **NAXISn**, **END** — являются обязательными и должны появляться в заголовке в фиксированном порядке.

— Значениями ключевого слова **SIMPLE** могут быть **T** и **F** (**T**, если формат файла согласуется полностью со стандартом, и **F** в противном случае).
— Число битов в одном значении данных определяется **BITPIX.** Оно может быть равным: 8 (беззнаковое целое), 16 (целое), 32 (целое), 32 (вещественное), 64 (с двойной точностью).

Число битов в изображении вычисляется по формуле **NBITS = |BITPIX|*(NAXIS1*NAXIS2*....*NAXISm)**, где **NAXIS**- число осей массива данных, 0 — 999, (если 0, то данных нет, имеется только заголовок)

NAXIS1 — число точек первой оси и т. д., **END** — последнее ключевое слово конец заголовка). Если до конца 2880 байтовой записи еще остается место, то оно заполняется пробелами.

Для расширений первым словом в заголовке является **XTENSION**. Его значение — символьная строка, содержащая название расширения, например, **XTENSION = BINTABLE** (для двоичных таблиц).

Итак, заголовок определяет реальный формат и размер последующей группы данных, а также содержит некоторую другую информацию. Заголовки можно просматривать, например, командами LINUX **$less** или **$more**. Более подробную информацию о ключевых словах FITS можно найти в *MIDAS User Guide* (volume A) и в работах, на которые там даны ссылки.

Заметим, что в отличие от заголовков данные в группах (HDU) нельзя просмотреть, используя стандартные редакторы или команды **less** и **more** операционной системы LINUX.

1.2. вод из FITS-файлов

Преобразование файлов, находящихся на диске в формате FITS, во внутренний формат MIDAS может быть сделано командой

```
INDISK/FITS in_files [out_spec]
```

где **in_files** определяет FITS-файл, а **out_spec** — имя выходного файла в формате MIDAS. Например:

```
INDISK/FITS xyz.fits
```

Преобразует FITS-файл xyz.fits в toto0001.bdf, или toto0001.tbl, или toto0001.fit в зависимости от того, что содержится в файле xyz.fits. Заметим, что если выходное имя не указывается, то файлу назначается имя по умолчанию, начинающееся с toto и с порядковым номером.

Можно вводить файлы из каталога и записывать в каталог, например:

```
INDISK/FITS in.cat out.cat
```

при этом FITS-файлы из каталога **in.cat** преобразуются в файлы внутреннего формата MIDAS с соответствующими именами из каталога **out.cat.**

Еще пример:

```
INDISK/FITS galaxy* ROOT=ngc
```

Все FITS-файлы, начинающиеся с galaxy, будут преобразованы в ngc0001.bdf, ngc0002.bdf и т. д.

Аналогичный перевод FITS-файлов, записанных на магнитной ленте или диске, можно выполнить командой

```
INTAPE/FITS file_list id device
```

где **file_list** — список номеров файлов, которые будут считаны, **id** — префикс (не более 4 символов) имен FITS-файлов, **device** — имя ленточного устройства или префикс создаваемых файлов на диске (по умолчанию расширение **.bdf**).

Это несколько устаревшая команда из-за того, что ленточные устройства сейчас редко используются для транспортировки и хранения данных.

Например, команда

```
INTAPE/FITS 2,5-7 st im
```

считывает FITS-файлы **im0002.mt**, **im0005.mt**, **im0006.mt**, **im0007.mt**, конвертирует их и создает MIDAS-изображения **st0002.bdf**, **st0005.bdf**, **st0006.bdf**, **st0007.bdf**;

```
INTAPE/FITS 1-2 st image
```

которая читает файлы **image0001.mt, image0002.mt** и создает **st0001.bdf и st0002.bdf** (предполагается, что FITS-файлы являются изображениями).

Заметим, что в принципе изображения (но не таблицы или fit-файлы) в формате FITS могут быть прямо использованы в MIDAS (т. е. без выполнение команд **INDISK/FITS** и т. п.). Однако в этом случае изображения будут полностью храниться в оперативной памяти компьютера. И хотя при работе с одним или двумя изображениями проблем обычно не возникает, при выполнении команд, выполняющих вычисления, типа **AVERAGE/IMAGE,** захватывается значительная память, что замедляет работу.

1.3. Вывод в FITS-файлы

Преобразование MIDAS-файлов в FITS-файлы производится командой

```
OUTTAPE/FITS cat device [flag] [dens,block] [type]
```

где **cat** — имя каталога, содержащего MIDAS-изображения, или просто имя изображения, а **device** может быть именем выходного файла в формате FITS.

И снова для записи на диск более удобной представляется команда

```
OUTDISK/FITS in_files [out_spec] [option] [out_type]
```

где **in_files** определяет файлы, переводимые в формат FITS, а **out_spec** — имя выходного FITS-файла. Например, команда

```
OUTDISK/FITS xyz
```

преобразует Midas-изображение **xyz.bdf** в FITS файл **toto0001.mt**, отметим, что расширение **.bdf** входного файла и имя **toto0001.mt** выходного файла берется по умолчанию.

Здесь также можно использовать MIDAS-каталоги, например:

```
OUTDISK/FITS in.cat out.cat
```

преобразует все файлы из каталога **in.cat** в FITS-файлы с соответствующими именами**.**

2. Работа с ASCII-файлами

Преобразование ASCII-файлов в объекты MIDAS: изображения, таблицы, ключевые слова и т. д. (и обратно) уже обсуждалась нами ранее. Здесь мы лишь напомним соответствующие команды.

2.1. Ввод из ASCII-файлов

Следующей командой из ASCII-файла создается Midas-изображение

```
INDISK/ASCII in_file [out_file] [npix_string]
```

где **in_file** — имя входного ASCII-файла, **out_file** — имя выходного Midas файла (максимально 3D, по умолчанию **toto.bdf**), **npix_string** — строка с числом осей и размерами изображения по каждой оси (по умолчанию — одномерное изображение). Например:

```
INDISK/ASCII spectr.asc ima
```

Из файла **spectr.asc** создается одномерное изображение **ima.bdf**.

Построить изображение по данным из ASCII-файла можно командой

```
CREATE/IMAGE frame [dim_specs] [frame_specs] AS-
    CII_FILE [coefs]
```

где **frame** — имя нового изображения, **dim_specs** — спецификация размерностей создаваемого изображения, **frame_specs** — дополнительные дескрипторы, **coefs** — функция, задающая значения пикселей создаваемого изображения. Более детальную информацию можно получить, используя справку в MIDAS (см. также раздел V п.5.1).

Следующая команда создает таблицу из ASCII-файла:

```
CREATE/TABLE table ncol nrow file [format_file]
[organization]
```

Здесь **table** и **file** — имена новой MIDAS-таблицы и исходного файла соответственно, остальные параметры были описаны нами ранее при рассмотрении таблиц в MIDAS (см. раздел III п.2).

Команды

```
OPEN/FILE filename READ file_control_key
READ/FILE file_id cbuf_key [maxrd]
```

позволяют считывать символьные (*character*) данные в ключевые слова MIDAS из ASCII-файлов. Получить информацию о приведенных параметрах можно, используя **HELP** для этих команд. Отметим, что перед тем, как вводить команду **READ/FILE,** надо открыть ASCII-файл командой **OPEN/FILE**, например:

```
OPEN/FILE outputc
..READ/FILE output {fctr(1)} 20
```

считывает 20 символов из файла **outputc** в ключевое слово **fctr(1).**

2.2. Вывод в ASCII-файлы

Это преобразование обсуждалось для MIDAS-таблиц в разделе III.3.1. Оно состоит из двух команд.

Первая из них —

```
ASSIGN/PRINT FILE file
```

где **file** — имя выходного ASCII-файла; вторая зависит от структуры данных:

```
PRINT/IMAGE frame_specs [pixel_specs]
PRINT/TABLE table [column...]
PRINT/KEYWORD [key_list]
```

где **frame_specs** — имя изображения, **pixel_specs** описывает положение и размер части изображения, предназначенной для печати; **table** — имя MIDAS-таблицы, **column** — ссылка на столбец; **key_list** — одно или несколько имен ключевых слов, разделенных запятыми (без пробелов!).

Одномерное изображение можно записать в ASCII-файл командой

```
OUTDISK/ASCII in_file out_file
```

где **in_file –** имя изображения, **out_file** — имя ASCII-файла.

Записать строку символов или значение символьного ключевого слова в ASCII-файл, открытый предварительно **OPEN/FILE**, можно командой

```
WRITE/FILE file_id charbuf
WRITE/FILE file_id,KEY charkey
```

Например, команда

```
WRITE/FILE 7 Picture of NGC 1022
```

записывает в ASCII-файл строку «Picture of NGC 1022»

Вторая форма записи команды применяется для ключевых слов, длина которых превышает 80 символов.

3. Вывод на экран (terminal)и ввод с клавиатуры

Можно направить всю выходную информацию на экран монитора командой

```
ASSIGN/PRINT TERMINAL
```

Иногда может быть полезна и команда

```
WRITE/OUT "text_string"
```

которая печатает на экране заключенную в апострофы строку текста **text_string**.

Ввести с клавиатуры значение в ключевое слово можно командой

```
INQUIRE/KEYWORD key [prompt_string] [flush_opt]
```

key — ключевое слово, **prompt_string** — подсказка, **flush_opt** — флаг обнуления ключевого слова перед вводом нового значения. Например:

```
INQUIRE/KEY petrita/c/6/5 "Give me a name:" flush
```

вводит в символьное ключевое слово **petrita** новое значение, предварительно его очистив.

4. PostScript-формат

Как известно, большинство научных журналов принимают рисунки в формате *PostScript*. В MIDAS можно выводить данные в этом формате. Существует различие при выводе графиков и изображений в PostScript-файлы.

4.1. Вывод графика в PS-файл

Здесь можно использовать следующие команды. Сначала настройка, назначающая вывод в PostScript-файл (теперь **GRAPH**, а не **PRINT**)

```
ASSIGN/GRAPH POSTSCRIPT
```

затем непосредственно команда, строящая график, например,

```
PLOT/COLUMN frame
```

В результате этого в рабочей директории будет создан PostScript-файл со *стандартным* именем **postscript.ps**. Его следует переименовать, например, командой Linux

```
$mv postscript.ps my_plot.ps
```

где вместо **my_plot.ps** может быть любое имя файла, расширение **.ps** разумно сохранить. Очевидно, что переименование необходимо, если выводится более одного PS-файла.

Отметим, что после того, как вывод в PostScript-файлы завершен, следует не забывать о восстановлении нормальной моды (**GRAPH**). Для этого служит команда

```
ASSIGN/GRAPH g,0
```

4.2. Вывод изображения в PS-файл

Сначала строим изображение в соответствующем окне MIDAS, например, командой

```
LOAD/IMAGE frame
```

а затем дадим команду

```
COPY/DISPLAY p5=noprint
```

В результате этого в текущей директории будет создан PostScript-файл со *стандартным* именем **screen00.ps.** Отметим, что название файла содержит номер сессии MIDAS, т. е., если MIDAS был запущен командой **inmidas 01**, то тогда стандартное имя файла будет **screen01.ps.** Его также следует переименовать, например, командой Linux, как было показано выше.

Очень важно при этом, чтобы MIDAS-окно с изображением было ***полностью*** видно на экране монитора, иначе последняя команда

работать не будет! Кроме этого, перед печатью окно не должно быть закрыто другими окнами Linux (X Windows).

В системах Linux, Windows и других просмотр PostScript-файлов обычно осуществляется командой **Ghostview** (**Ghostscript**). Некоторые замечания о работе с ней в Linux приведены в *Приложении А*.

5. Перенаправление ввода/вывода

В MIDAS можно использовать перенаправление ввода-вывода так же, как это делается в Linux (см. *Приложение А*), то есть любая информация, выдаваемая на терминал программой, может быть сохранена в файле или передана как ввод другой программе.

Например, копирование содержимого таблицы в текстовый файл выполняется двумя командами

```
ASSIGN/PRINT file startable.dat
PRINT/TABLE startable
```

Первой командой назначаем вывод в файл, а второй — выводим таблицу.

Это же действие можно выполнить, используя *перенаправление* ввода/вывода, так:

```
WRITE/TABLE startable >startable.dat
```

В результате в текущей директории будет создан файл **startable.dat**.

```
WRITE/TABLE startable >>startable.dat
```

В этом варианте информация дописывается в уже существующий файл.

Отметим, что в отличие от синтаксиса перенаправления ввода/вывода в Linux, в MIDAS надо писать знаки **>** и **<** , указывающие перенаправление, всегда после пробела, но имя файла пробелом не отделяется!

Еще несколько полезных примеров. Можно записать данные в файл с одновременным выводом их на терминал, используя команду типа

```
WRITE/TABLE mytable >mytable.dat+terminal
```

Вывод на терминал можно отменить, применяя перенаправление вывода на специальное устройство (нуль-устройство)

```
WRITE/TABLE mytable >Null
```

.

В MIDAS можно использовать программный канал (pipe). Это обозначает, что вывод MIDAS-программы можно направить на вход UNIX-программе, и наоборот. Дадим два примера:

```
READ/DESCRIPTOR mama | $grep CUNIT
```

эта команда выдает стандартные дескрипторы изображения **mama.bdf**, причем вывод команды направляется на вход команды Linux **grep**, которая ищет совпадение по шаблону **CUNIT**.

```
$pwd | WRITE/KEYWORD INPUTC
```

команда записывает название текущей директории в стандартное ключевое слово **INPUTC**.

Заметим, что при использовании программного канала между несколькими командами Linux знак **$** ставится только один раз, например:

```
$ls | grep gal
```

что позволяет просмотреть содержимое текущей директории и найти файлы, в названии которых встречается **gal**.

Упражнение

Написать MIDAS-процедуру, которая:

1) создает MIDAS-изображение (любым способом, например, как в упражнении к разделу V) и копирует его в MIDAS-таблицу;
2) записывает как изображение, так и таблицу на диск в виде FITS-файлов;
3) считывает изображение и таблицу из этих файлов в соответствующие структуры MIDAS с именами, отличающимися от первоначальных;
4) трансформирует таблицу в ASCII-файл;
5) переводит изображение в PostScript-файл;
6) один скан изображения (например, любую строку) представляет в виде еще одного PostScript-файла.

Параметрами процедуры могут быть размеры изображения/таблицы и имена файлов.

VII. ПРОГРАММЫ НА ФОРТРАНЕ И СИ В MIDAS

0. Введение

Система команд MIDAS достаточно развита, чтобы пользователь мог выполнить практически все необходимые операции над данными, не прибегая к созданию новых программ. Тем не менее, в MIDAS существует возможность использовать программы, написанные на Фортране и Си.

Весьма просто в MIDAS работать с программами, которые *не* предполагают какого-либо взаимодействия с данными, представляемыми во внутреннем формате MIDAS (иными словами с изображениями, таблицами и т. д.). В противоположном случае, когда программа должна оперировать с такими данными, доставляет обычно много хлопот. И этот случай будет рассмотрен нами достаточно подробно. Условно программы первого типа будем называть *простыми*, а второго — *сложными*.

A. ПРОСТЫЕ ПРОГРАММЫ НА ФОРТРАНЕ ИЛИ СИ

Если программа не работает со структурами данных MIDAS, она компилируется обычными командами Linux, например:

`f77 myprog.f -o myprog.exe` (для программ на Фортране)
`g77 myprog.f -o myprog.exe`
`cc myprog.c -o myprog.exe` (для программ на Си)

и т. п.

Запустить полученный в результате компиляции файл (**myprog.exe**), можно командой MIDAS

```
RUN myprog.exe
```

Б. СЛОЖНЫЕ ПРОГРАММЫ

Когда возникает необходимость произвести нестандартную обработку данных или использовать возможности MIDAS, например, в системе сбора данных, которая управляет процессом наблюдений. В этих случаях обычно приходится писать собственную программу на Фортране или Си. Ниже мы последовательно рассмотрим *правила написания* таких программ, *специальные MIDAS-интерфейсы* (стандартные подпрограммы), которые необходимо использовать, а также *особенности компиляции*.

1. Правила написания программ

Наличие особых правил диктуется тем, что программа, написанная на Фортране или Си, перед компиляцией должна быть обработана специальным *препроцессором* MIDAS. Поэтому при написании таких программ на Фортране следует учитывать следующее:

а) создатели MIDAS приняли за правило, что *любая программа в системе содержать специальный блок комментариев.* Блок помещается в начале программы и содержит описание имени программы, ее назначения, параметров и т. п. (см. пример программы, приведенный в конце этого раздела);

б) *нельзя использовать оператор IMPLICIT для описания переменных.* После комментариев в начале программы ставится оператор

```
IMPLICIT NONE
```

и *все* переменные должны быть описаны явно;

в) *должны быть подключены специальные (include-)файлы.* Для использования интерфейсных подпрограмм MIDAS в программу включаются два include-файла, которые объявляют системные глобальные переменные и присваивают им соответствующие значения. Это производится следующими операторами:

```
INCLUDE 'MID_INCLUDE:ST_DEF.INC'
```

```
INCLUDE 'MID_INCLUDE:ST_DAT.INC'
```

г) *должен присутствовать специальный COMMON блок* с именем **VMR,** необходимый для эмуляция указателей/ссылок (в языке Си они определяются явно);

д) *нельзя использовать* оператор READ с бесформатным вводом и т. п.;

е) *приняты за правило отступы в группах операторов.* Для каждого оператора в группах, заключенных между **IF, ELSE и ENDIF**, а также между **DO** и **ENDDO** или **CONTINUE**, отступ слева должен быть увеличен как минимум на два пробела (см. пример в пункте **B** ниже);

ж) *запрещены некоторые номера меток.* Не следует употреблять метки в интервале **80000–81000**, поскольку он используется препроцессором MIDAS;

Сходные требования предъявляются и к программам, написанным на Си. При этом, конечно, учитывается специфика этого языка. Например, подключение include-файла *(в)* выглядит так:

```
#include <midas_def.h>
```

а ряд требований отпадает (*г*, *ж* и др.).

Добавим, что, кроме выше перечисленных ограничений, есть и дополнительные возможности, предоставляемые препроцессором. Например, комментарии в программе можно помещать в любом месте после знака !, как это принято в MIDAS.

2. Интерфейсные подпрограммы MIDAS

MIDAS имеет ряд интерфейсов — специальных подпрограмм, которые позволяют программам пользователя получать доступ к структурам данных MIDAS.

Для того чтобы открыть доступ в среду MIDAS, в начале программы на *Фортране* следует включить вызов подпрограммы

```
STSPRO ('progname')
```

где **progname** — имя программы (см. пример в пункте **B**).

По окончании работы в среде MIDAS следует вызвать подпрограмму

```
STSEPI
```

которая не имеет параметров.

Для программ на языке *Си* имена соответствующих процедур **SCSPRO** и **SCSEPI**. Отметим стандартное изменение префикса (с **ST** для Фортрана на **SC** для Си) и в дальнейшем будем давать лишь названия подпрограмм для Фортрана.

2.1. Ключевые слова

Значения ключевого слова, имеющего MIDAS-тип real (**R**), могут быть *считаны* в фортрановскую переменную (массив) подпрограммой

```
STKRDR (key, felm, maxs, acts, vals, kun, knul, stat)
```

Здесь *входные парамеры*: **key** (тип character) — имя ключевого слова (макс. 8 знаков) в апострофах, **felm** и **maxs** (оба integer) — первый элемент и число элементов для передачи в фортрановскую программу.

Выходные параметры: **acts** (integer) — число переданных элементов, **vals** (real) — массив, в который передаются значения ключевого слова, **kun** и **knul** (оба integer) не используются, **stat** (integer) равен нулю при правильной передаче данных.

Значения ключевых слов, имеющих тип double precision (**D**), integer (**I**) и character (**C**), передаются соответственно подпрограммами: **STKRDD, STKRDI** и **STKRDC**. Эти подпрограммы имеют (практически) те же параметры, что и **STKRDR**. Исключение составляет лишь тип **vals**, который меняется в соответствии с типом ключевого слова: integer для **STKRDI** и т. д.

В дальнейшем вместо перечисления всех подпрограмм, например **STKRDR**, **STKRDD**, **STKRDI** и **STKRDC**, будем использовать *обозначения* типа **STKRDx**, подразумевая, что **x = R, D, I, C**.

Обратное преобразование — *запись* значений массива в ключевое слово производится подпрограммами

```
STKWRx (key, vals, felm, maxs, kun, stat)
```

где параметры имеют тот же смысл, что и выше, в подпрограммах **STKRDx**.

2.2. Изображения

Следующая подпрограмма *считывает* все (включая системные) дескрипторы изображения и делает его доступным для фортрановской программы:

```
STIGET (name, dattyp, iomode, filtyp, maxdim,
naxis, npix, start, step, ident, cunit, pntr, no,
stat)
```

При этом *входные параметры*: **name** (тип character) — имя изображения, переведенное подпрограммой **STKRDC** в фортрановскую переменную (см. замечание ниже!); **dattyp, iomode** и **filtyp** (все integer) — тип данных, мода и тип файла, которые обычно обозначаются переменными, определенными в include-файлах, например **D_R4_FORMAT, F_I_MODE, F_IMA_TYPE** (см. также пример в пункте **B**); **maxdim** (integer) — размерность изображения (1, 2 или 3).

Выходные параметры: **naxis** (integer) — размерность (число осей); **npix** (integer) — массив, показывающий число пикселов вдоль каждой из осей; **start** и **step** (оба double precision) — массивы, содержащие начальное значение и шаг для каждой оси; **ident** и **cunit** (оба character) — идентифицируют изображение; **pntr** (integer) — аналог указателя, работу которого иллюстрирует пример в пункте **B**; **no** (integer) — идентификатор изображения; **stat** (integer) — то же, что и в предыдущих подпрограммах. Отметим, что в программе необходимо задать начальные значения параметров **ident** и **cunit** в виде пробелов операторами

```
DATA ident/' '/
DATA cunit/' '/
```

Создать новое MIDAS-изображение можно фортрановской подпрограммой

```
STIPUT (name, dattyp, iomode, filtyp, naxis, npix,
start, step, ident, cunit, pntr, no, stat)
```

где значения параметров те же, что и в подпрограмме **STIGET** выше.

Заметим, что определенную проблему представляет передача имени изображения в фортрановскую программу и обратно. Делает-

ся это, используя какое-либо ключевое слово типа character и одну из подпрограмм, рассмотренных выше, а именно:

```
STKRDC (key, felm, maxs, acts, vals, kun, knul,
stat)
```

где **key** — имя ключевого слова, содержащего имя изображения, **vals** — переменная типа character, которая фигурирует как параметр name в подпрограммах **STIGET** и **STIPUT** (см. также пример в пункте **В**).

2.3. Дескрипторы

Значения дескриптора могут быть *считаны* в фортрановскую переменную (массив) подпрограммами

```
STDRDx (no, dsc, felm, maxs, acts, vals, kun, knul,
stat)
```

Здесь *входные парамеры*: **no** (тип integer) — идентификатор изображения, с которых связан дескриптор, определенный ранее подпрограммой **STIGET** или **STIPUT**; **dsc** (character) — имя дескриптора в апострофах (макс. 15 символов); **felm** и **maxs**, а также все *выходные параметры* те же, что и в подпрограммах **STKRDx** для ключевых слов.

Записать значения дескриптора можно подпрограммами

```
STDWRx (no, dsc, vals, felm, maxs, kun, stat)
```

где все параметры (за исключением stat) являются входными и имеют тот же смысл, что и выше.

2.4. Прочее

Вызов подпрограммы

```
STETER (errno, text)
```

с входными параметрами **errno** (тип integer) и **text** (character) останавливает выполнение фортрановской программы и выдает в MIDAS сообщение (об ошибке) в виде номера errno и текста text.

3. Компиляция сложных программ

Усложнение вызвано необходимостью работы препроцессора MIDAS и подключения библиотек интерфейсных подпрограмм на Фортране (Си), имеющихся в MIDAS. Процесс компиляции состоит из нескольких шагов.

Следующая команда Linux запускает препроцессор MIDAS для *фортрановских* программ:

```
/$midas/$98NOV/system/exec/esoext.exe
    -I/$midas/$98NOV/incl -f myprog.for
```

где **$midas** — абсолютный путь в директорию, в которой содержится версия MIDAS (например, с именем 95NOV), **$98NOV** — номер версии MIDAS на вашем компьютере (95NOV, 98NOV и т. д.), всегда являющийся и именем вышеупомянутой директории, **myprog.for** — имя файла с фортрановской программой. Результат работы препроцессора — файл с тем же именем, но с расширением **.f** (в данном случае **myprog.f**).

На втором шаге производится обычная трансляция

```
g77 -c myprog.f
```

Ее результат — объектный модуль с тем же именем, но с расширением **.o** (в данном случае **myprog.o**).

Третий шаг — компиляция с подключением библиотек подпрограмм MIDAS

```
g77 myprog.o -L/$midas/$98NOV/lib -lmidas -o myprog.exe
```

Результат — исполняемый файл (**myprog.exe**), который можно запустить на выполнение только командой MIDAS RUN (см. пункт **A**).

Для программы, написанной на языке *Си*, процесс компиляции состоит из двух шагов:

```
cc -I$midas/$98NOV/incl -c myprog.c
cc myprog.o -L$midas/$98NOV/lib -lmidas -o myprog.exe
```

где обозначения те же, что и выше. Если в программе используются стандартные математические функции, то нужно добавить еще опцию **-lm** в конце второй строки.

4. Вызывающая процедура

Исполняемый файл запускается командой **RUN** (см. пункт **А**). Однако лучше это делать из процедуры MIDAS, например так:

```
!===================================
ECHO/ON
DEFINE/PARAMETER P1 ? I
DEFINE/PARAMETER P2 ? I
DEFINE/PARAMETER P3 10 N
DEFINE/PARAMETER P4 H C
WRITE/KEYWORD IN_A {P1}
WRITE/KEYWORD OUT_A {P2}
WRITE/KEYWORD MY_KEYW1/R/1/3 {P3}
WRITE/DESCR {IN_A} MY_DATE/I/1/3 22,04,2000

READ/KEYWORD MY_KEYW1
READ/DESCR {IN_A} MY_DATE
LOAD {IN_A}

RUN myprog.exe

READ/KEYWORD MY_KEYW1
READ/DESCR {OUT_A} MY_DATE
LOAD {OUT_A}
!===================================
```

Данная процедура имеет три параметра: имена исходного и создаваемого изображений и начальное значение ключевого слова. Процедура передает имена изображений в ключевые слова, что необходимо для программы (**myprog.exe**), создает новый дескриптор, выводит значения ключевого слова и дескриптора и показывает изображение до и после исполнения программы **myprog.exe**, описанной в следующем пункте. Заметим, что попытка назвать новый дескриптор **DATE** вызвала бы серьезные осложнения, поскольку такое имя имеет системный дескриптор.

В. ПРИМЕР ПРОГРАММЫ НА ФОРТРАНЕ

Данный ниже пример иллюстрирует правила написания программ, отмеченные в п. 1, и работу с интерфейсными подпрограммами MIDAS, обсуждавшимися в п. 2. Программа должна быть помещена в файл **myprog.for** и откомпилирована в соответствии с указаниями из п. 3.

В начале программы **myprog** находится блок комментариев и описание всех используемых переменных, а также подключаются два include-файла, задаются начальные значения двух параметров и открывается доступ в среду MIDAS. Затем считывается, изменяется и записывается снова значение ключевого слова; определяются исходное и создаваемое изображения; вызывается фортрановская процедура **inverse**, которая делает нечто вроде негатива изображения (процедура находится в конце программы); наконец считывается значение дескриптора исходного изображения; это значение изменяется и записывается как дескриптор создаваемого изображения.

```
C @(#)myprog.for ------------------------------
C
C.IDENTIFICATION
C
C.KEYWORDS
C
C.PURPOSE
C
C.ALGORITHM
C
C.INPUT/OUTPUT
C the following keywords are used:
C IN_A/R/1/60 input frame
C OUT_A/R/1/60 output frame
C ...
C-----------------------------------------------
C
c PROGRAM myprog.for

IMPLICIT NONE
```

```
INTEGER NAXISA, NPIXA(2), IAV, STAT, IRAD,
    IMNOA, IMNOC, PNTRA
INTEGER KNULL, KUNIT(1), DUN, DNUL, IDISCR(4),
    MADRID(1), PNTRC
CHARACTER*60 FRAMEA, FRAMEC
CHARACTER CUNITA*64, IDENTA*72, DEFAUL*1
DOUBLE PRECISION STEPA(2), STARTA(2)
REAL INPUTR(3)

COMMON /VMR/ MADRID

INCLUDE 'MID_INCLUDE:ST_DEF.INC'
INCLUDE 'MID_INCLUDE:ST_DAT.INC'
DATA IDENTA/' '/
DATA CUNITA/' '/

C---- initialization
CALL STSPRO ('myprog')
c---- get, change, and pass a keyword value
CALL STKRDR ('MY_KEYW1', 1, 3, IAV, INPUTR,
    KUNIT, KNULL, STAT)
inputr(1) = 13.
inputr(2) = 100.
CALL STKWRR ('MY_KEYW1', INPUTR, 1, 2, KUN,
    STAT)
c---- define input image
CALL STKRDC ('IN_A', 1, 1, 60, IAV, FRAMEA,
    KUNIT, KNULL, STAT)
CALL STIGET (FRAMEA, D_R4_FORMAT, F_I_MODE,
    F_IMA_TYPE, 2,
& NAXISA, NPIXA, STARTA, STEPA, IDENTA, CUNITA,
    PNTRA, IMNOA,
& STAT)
c---- define output image
CALL STKRDC ('OUT_A', 1, 1, 60, IAV, FRAMEC,
    KUNIT, KNULL, STAT)
CALL STIPUT (FRAMEC, D_R4_FORMAT, F_O_MODE,
    F_IMA_TYPE,
```

```
& NAXISA, NPIXA, STARTA, STEPA, IDENTA, CUNITA,
    PNTRC, IMNOC,
& STAT)
c---- change image in a Fortran procudure
call inverse (MADRID(PNTRA), MADRID(PNTRC),
    NPIXA(1), NPIXA(2))
c---- get, change and pass a descriptor value
call STDRDI (IMNOA, 'MY_DATE', 1, 3, IAV,
    IDISCR, DUN, DNUL, STAT)
IDISCR(1) = 12
call STDWRI (IMNOC, 'MY_DATE', IDISCR, 1, 3,
    DUN, STAT)
c---- leaving the program
CALL STSEPI
END
c----
subroutine inverse (a, b, n, m)
implicit none
real a(1), b(1), s
integer n, m
s = 0
do 1000 j = 1, m*n
if (a(j).gt.s) s = a (j)
1000 continue
do 2000 j = 1, m*n
b(j) = s – a (j)
2000 continue
return
end
c----------------------------------------------------------------
```

Г. ПРИМЕР ПРОГРАММЫ НА СИ

Для иллюстрации приведем сходной пример программы на Си, которую написали студенты О. Галай и О. Барсунова.

```
/* ------------------------------------------------
.COPYRIGHT (c) 2000 St.Petersburg University
```

```
.IDENT demo.c
.AUTHOR Galay Oleg, Barsunova Olga
.KEYWORD MIDAS.
.LANGUAGE C.
.PURPOSE Demo program for practice
.VERSION 1.0 September 2000 .
.RETURNS Output a sum of images.
.ENVIROMENT
.COMMENT Run by demo.prg.
---------------------------------------------- */
#include<string.h>
#include<ctype.h>
#include<math.h>
#include<midas_def.h>

int main()
{
int i, nval, null;
int onpix[2];
char text[84], ima1[84], ima2[84], imaout[84],
    ident[85], cunit[84];
int unit;
double ostart[2], ostep[2];
double *outdata;
double *ima1data;
double *ima2data;
int ima1id, ima2id, outid;
double DescVal[2];
double DescVal2[2];
double DescVal1;

SCSPRO("Demo program");
sprintf(text,"**********************************
    ******************");
SCTPUT(text);
sprintf(text, "This program creates two images
    and sums them.");
SCTPUT(text);
```

```
sprintf(text,"**********************************
    ******************");
SCTPUT(text);

sprintf(text, "Reading MIDAS-keywords in C-
    variables.");
SCTPUT(text);
SCKGETC("INA", 1L, 60L, &nval, ima1);
SCKGETC("INB",1L, 60L, &nval, ima2);
SCKGETC("OUT",1L, 60L, &nval, imaout);

sprintf(text,"Image parameters.");
SCTPUT(text);
onpix[0] = 500;
onpix[1] = 500;
ostart[0] = 1.32;
ostart[1] = 1.32;
ostep[0] = 1.0;
ostep[1] = 1.0;

/* Create two images */
sprintf(text,"Open two images !");
SCTPUT(text);
SCIPUT( ima1, D_R8_FORMAT, F_O_MODE, F_IMA_TYPE,
    2, onpix, ostart, ostep, "First
image", "DATA: UNIT; AXIS: PIXEL", &ima1data,
    &ima1id );
SCIPUT( ima2, D_R8_FORMAT, F_O_MODE, F_IMA_TYPE,
    2, onpix, ostart, ostep, "Second
image", "DATA: UNIT; AXIS: PIXEL", &ima2data,
    &ima2id );

sprintf(text,"Put data in two images.");
SCTPUT(text);
for (i=0; i < onpix[0]*onpix[1]; i++)
{
ima1data[i] = sin(3.14*i/72);
ima2data[i] = cos(3.14*i/100);
```

```
}

/* Read and sum the images */
/* SCIGET( ima1, D_R8_FORMAT, F_IO_MODE,
    F_IMA_TYPE,2, 2, onpix, ostart, ostep,
ident, cunit, &ima1data, &ima1id ); */
/* SCIGET( ima2, D_R8_FORMAT, F_IO_MODE,
    F_IMA_TYPE,2, 2, onpix, ostart, ostep,
ident, cunit, &ima1data, &ima2id ); */
sprintf(text,"Open output image.");
SCTPUT(text);
SCIPUT( imaout, D_R8_FORMAT, F_O_MODE,
    F_IMA_TYPE, 2, onpix, ostart, ostep, "Output
image", "DATA: UNIT; AXIS: PIXEL", &outdata,
    &outid );

sprintf(text,"Sum two images.");
SCTPUT(text);
for (i=0; i < onpix[0]*onpix[1]; i++)
{
outdata[i] = (ima1data[i] + ima2data[i]);
}
sprintf(text,"Change descriptors.");
SCTPUT(text);
DescVal[0] = 145.756;
DescVal[1] = -145.756;
SCDWRD(ima1id, "START", &DescVal, 1, 2, &unit);
DescVal1 = 555.756;
SCDWRD(ima2id, "DENSITY", &DescVal1, 1, 1,
    &unit);
SCDRDD(ima2id, "START", 1, 2, &nval, DescVal2,
    &unit, &null);
sprintf(text,"Read START = %8.3f,%8.3f \n",
    DescVal2[0], DescVal2[1]);
SCTPUT(text);
DescVal2[1] = 300;
SCDWRD(outid, "START", &DescVal2, 1, 2, &unit);

SCFCLO(ima1id);
```

```
SCFCLO(ima2id);
SCFCLO(outid);
SCSEPI();
}
```

Процедура, вызывающая программу, может быть такой.

```
!================================================
define/parametr P1 ? ? "Enter first image :> "
define/parametr P2 ? ? "Enter second image :> "
define/parametr P3 ? ? "Enter output image :> "
def/local INA/C/1/20 "{P1}"
def/local INB/C/1/20 "{P2}"
def/local OUT/C/1/20 "{P3}"
RUN demo.exe
!================================================
```

Упражнение

Написать MIDAS-процедуру (и программу на языке Фортран или Си), которая:

1) создает два разных MIDAS-изображения (любым способом, например, как в упражнении из раздела V);
2) снабжает одно из изображений каким-либо новым дескриптором вещественного типа и присваивает ему некое значение;
3) показывает значение этого дескриптора и исходные изображения на экране;
4) в фортрановской/Си-программе складывает поэлементно изображения и результат помещает в новое изображение;
5) в фортрановской/Си-программе частично изменяет значения дескриптора и присоединяет его к новому изображению;
6) показывает на экране MIDAS новое изображение, а также значение дескриптора.