COMPUTATIONAL MATHEMATICS
BANACH CENTER PUBLICATIONS, VOLUME 13
PWN–POLISH SCIENTIFIC PUBLISHERS
WARSAW 1984

ПАКЕТ ПРИКЛАДНЫХ ПРОГРАММ САФРА

М. М. ГОРБУНОВ-ПОСАДОВ, В. Я. КАРПОВ, Д. А. КОРЯГИН

Институт Прикладной Математики им. М. В. Келдыша АН СССР, Москва, СССР

1. Программное обеспечение вычислительного эксперимента

В настоящее время одним из основных методов изучения сложных физических и инженерно-технических проблем является метод вычислительного эксперимента, т.е. моделирование интересующего исследователя процесса на вычислительной машине.

Для физических задач основные этапы вычислительного эксперимента состоят в следующем:

1. Выбор физического приближения и формулировка математической модели (как задачи математической физики).

2. Выбор дискретной модели, аппроксимирующей исходную математическую задачу (например, построение разностной схемы) и разработка вычислительного алгоритма для решения уравнений.

3. Создание программы для реализации вычислительного алгоритма.

4. Проведение расчетов и обработка полученной информации.

5. Анализ результатов, сравнение с физическим экспериментом, пересмотр и уточнение физической модели и, если нужно, повторение всех этапов сначала.

С точки зрения программирования вычислительный эксперимент характерен тем, что для каждой физической модели с целью установления соответствия между физическим экспериментом и вычислительным экспериментом необходимо решать большое число вариантов (варьируя определяющие параметры задачи) и, кроме того, менять (уточнять) саму физическую модель. Эта особенность („многовариантность" и „многомодельность") вычислительного эксперимента проявляется в многократных изменениях программы, реализующей вычислительный алгоритм, причем эти изменения касаются как структуры

программы в целом, так и отдельных фрагментов программной реализации алгоритма.

Фактически, разработка программы для вычислительного эксперимента выливается в создание крупной программной системы (объемом порядка $10^5 \div 10^6$ команд), характеризующейся большим числом компонент и многообразием их взаимодействия. По-видимому, единственный реальный путь обеспечения вычислительного эксперимента программными средствами, позволяющими быстро собирать программы и моделировать вычислительный алгоритм и физическую модель, заключается в использовании модульного принципа программирования в сочетании с развитыми системными средствами. Плодотворность модульного принципа программирования обеспечивается тем важным обстоятельством, что описание одних и тех же физических процессов входит в различные комплексные проблемы. Кроме того, различные физические процессы часто описываются одними и теми же уравнениями. Например, одними и теми же уравнениями описываются такие процессы как диффузия, теплопроводность, намагничивание. Различие между процессами проявляется лишь в физическом смысле коэффициентов уравнений и искомой функции.

Что касается системных средств, то можно сказать, что разрабатывавшиеся до сих пор системные средства, несмотря на их практическую направленность, как правило, были ориентированы на некоего „обобщенного" пользователя. Это, конечно, обеспечивало их весьма широкую применимость, но в то же время приводило к тому, что с точки зрения каждой конкретной прикладной деятельности они оказывались не вполне адекватными. Иначе говоря, возможности, реализуемые системными средствами, определялись исходя из учета общих, а не специфических прикладных интересов. По мере расширения сфер прикладного программирования и совершенствования его методов выяснилось, что услуг, обеспечиваемых универсальными системными средствами (например, средствами отладки) не достаточно для успешного проведения работ, а также, что более характерно, использование таких средств для специальных целей (в частности, редактирования, модификации и сборки программ) сопряжено с трудоемкими операциями. На наш взгляд, это следствие сепаратизма, присущего разработкам прикладного и системного программирования, на современном этапе становится тормозом развития машинных приложений, связанных с решением комплексных проблем.

Проведенные рассмотрения позволяют сделать вывод о том, что одним из главных направлений работ в современном программировании должно стать создание проблемно-ориентированных программных систем, называемых пакетами прикладных программ. Причем успех на этом направлении может быть достигнут только путем объе-

динения усилий коллективов прикладных и системных программистов.

2. Функциональное и системное наполнения пакета прикладных программ

Под пакетом прикладных программ мы понимаем комплекс взаимосвязанных прикладных и системных программ, обеспечивающих адекватное покрытие некоторой прикладной деятельности. Поясним несколько подробнее использованные здесь термины.

Прежде всего отметим, что всякая конкретная прикладная деятельность характеризуется двумя факторами. Во-первых, предметной областью, то есть тем, на что направлен труд программиста, представляющую собой совокупность решаемых им прикладных задач, и, во-вторых, дисциплиной работы, т.е. совокупностью приемов, правил, принятых при разработке, отладке и эксплуатации программ. Адекватность покрытия означает достаточную полноту средств, связанных как с первым, так и со вторым факторами, определяющими конкретную прикладную деятельность. Другими словами, достаточно полным должен быть как состав модулей, служащих материалом для сборки программ из данной предметной области, так и состав средств хранения, редактирования модулей, сборки программ и выполнения других видов принятых в данной прикладной деятельности работ. При этом в общем случае адекватность покрытия следует рассматривать во времени, то есть всякое изменение в прикладной деятельности должно учитываться путем включения в пакет новых прикладных или системных программ. Таким образом, можно говорить о „динамической" адекватности покрытия.

Рассмотрим в качестве примера Систему Автоматизации Физических Расчетов (сокращенно САФРА), созданную в Институте прикладной математики им. М. В. Келдыша АН СССР. Этот пакет в существующем сейчас состоянии предназначен для расчетов задач одномерной газовой динамики с теплопроводностью с учетом различных физических процессов, таких как магнитные поля (задачи о пинчах), двухтемпературность, перенос излучения, поглощение света лазера, учет термоядерных реакций (задачи лазерного термоядерного синтеза). Всё это определяет предметную область пакета САФРА.

Кроме предметной области в пакете зафиксирована определенная технология разработки и эксплуатации программ решения больших физических задач. Эта технология выбрана на основе изучения предшествующего опыта создания таких программ и, в частности, опыта разработки и эксплуатации программ с помощью системы OLYMPUS,

разработанной под руководством доктора Робертса в Калэмской лаборатории (Великобритания).

Характерной особенностью работ, связанных с программами решения этих задач, является отсутствие чётко выраженной хронологической границы между этапами разработки и эксплуатации таких программ. Кроме того, содержанию работ на обоих этапах присуща известная технологическая общность. В связи с этим средства пакета САФРА выбраны таким образом, что они в равной мере могут быть использованы как при разработке, так и при эксплуатации программ.

Таким образом, организационно пакет прикладных программ можно представить состоящим из двух частей: функционального и системного наполнений.

Функциональное наполнение отражает специфику предметной области пакета и включает, например:

— совокупность модулей, используемых при составлении программ для решения задач данной предметной области;

— набор стандартных схем счета, определяющих модули, из которых состоит программа решения той или иной типичной задачи;

— набор описаний, отражающих различные функциональные и программно-эксплуатационные характеристики модулей и стандартных схем счета, входящих в пакет.

Требования на форму представления и организацию элементов функционального наполнения устанавливаются обычно при определении системных средств пакета.

Системное наполнение является административным органом пакета, отражающим дисциплину работы с пакетом. Системное наполнение может включать, например, такие компоненты:

— язык заданий, являющийся средством общения пользователя с пакетом;

— архив, являющийся системой хранения элементов функционального наполнения и служебной информации пакета;

— монитор, представляющий собой совокупность программных средств, обеспечивающих операционные возможности пакета.

Следует отметить, что системное наполнение может быть в известной степени инвариантным относительно предметной области, то есть системное наполнение, разработанное для некоторой предметной области, может оказаться легко адаптируемым для других предметных областей.

Рассмотрим подробнее основные компоненты пакета САФРА.

3. Архив пакета САФРА

Архив предназначается в первую очередь для хранения накапливаемого программного фонда и формулируемой на языке заданий ин-

формации о структуре собираемых программ. Хранимые в архиве единицы информации называются *модулями* пакета и являются основными элементами обработки в пакете прикладных программ САФРА. Каждый модуль характеризуется наличием у него *архивного имени* и возможностью независимого доступа. Всякий доступ из задания пользователя к модулю в архиве осуществляется только посредством указания в задании архивного имени этого модуля, т.е. имени, под которым этот модуль хранится в архиве. Специфика проблем сборки и стремление к логической простоте и экономичности языка заданий предопределили отход от традиционного понимания модуля, которое обычно ассоциируется с законченной программной единицей, допускающей автономную трансляцию. В системе САФРА в модули выделяются функционально связные части программируемого алгоритма или задания на сборку вне зависимости от того, будут ли они допускать автономную трансляцию или нет. Архивное имя модуля является важным объектом в системе САФРА, поскольку после определенных автоматических преобразований задание на сборку сводится к перечислению в нужном порядке архивных имен модулей, из которых составляются программа и ее начальные данные. Гибкость использования программного фонда и удобство широкого варьирования сочетаний модулей в программах основываются на том, что в программах на языке заданий указываются не архивные имена модулей, с которыми требуется установить программные связи, а функциональные имена, отражающие функциональный смысл того или иного нужного модуля и определяющие место такого модуля в общей структуре программы. Каждое функциональное имя относится к некоторому классу модулей, имеющих сходное значение. В языке заданий предусмотрены средства указания соответствий между функциональными и архивными именами модулей для задания конкретных вариантов сборки.

В системе САФРА различаются модули следующих типов:

1. Программная единица (PRUNIT),
2. Вставка (MACRO),
3. Схема счета (SCHEME),
4. Версия (VERSION),
5. Вариант (VARIANT).

Модуль *программная единица* служит аналогом программной единицы языка программирования. (Примерами программных единиц являются основная программа, подпрограмма либо функция-подпрограмма языка ФОРТРАН, процедура модульных версий языка АЛГОЛ и т.п.). В отличие от программной единицы языка программирования, модуль пакета может содержать не только предложения языка прог-

раммирования, но и специальные управляющие предложения языка заданий пакета, которые, например, могут указывать на то, что в данное место модуля нужно вставить некоторую последовательность предложений, которая в пакете описана в виде модуля вставки.

Модуль *вставка* предназначается для оформления одного или нескольких предложений языка программирования или языка заданий, которые могут служить вставками, например, в программные единицы или в модули других типов. Обычно в виде вставок оформляются последовательности операторов, имеющие относительно самостоятельное значение и определяющие те локальные решения, которые могут изменяться в различных программных комбинациях. Например, для ФОРТРАН'а имеет смысл оформлять в виде вставок изменяемые в зависимости от масштабности решаемой задачи операторы DIMENSION или COMMON. Поскольку на одну и ту же вставку разрешается ссылаться из различных модулей пакета, то сопоставление одному функциональному имени вставки нового архивного имени позволяет без дополнительных затрат труда произвести на уровне сборки нужную модификацию всех модулей, окончательный вид которых зависит от данного локального решения. В модуле вставка могут содержаться ссылки на другие вставки. Вставка не должна явно или неявно через другие вставки ссылаться на себя.

Модуль *схема счета* предназначен для запоминания перечня основных функциональных элементов, на которые расчленяется алгоритм. Этому перечню элементов соответствует набор программных единиц, участвующих в решении некоторого класса задач. Примером такого набора с фиксированным порядком вызовов может служить схема счета системы OLYMPUS. Схема счета составляется в терминах функциональных имен в расчете на широкую свободу варьирования конкретных программ в рамках задаваемого перечня общих для этих программ функциональных элементов. При сборке конкретной программы каждому функциональному имени ставится в соответствие некоторое архивное имя модуля. Аналогичная принципиальной блок-схеме в содержательных наименованиях, схема счета хранится в архиве для того, чтобы на ее основе системой могли пользоваться различные люди, работа которых сводится к внесению в эту схему необходимых для конкретной реализации уточнений.

Модуль *версия* служит для описания конкретной версии программы, т.е. для установления соответствия между отдельными функциональными именами, указанными в схеме счета, и архивными именами соответствующих программных единиц.

Модуль *вариант* предназначен для описания конкретного счетного варианта программы. В нем должны быть конкретизированы все

вставки в программу, данные, необходимые для работы операторов ввода, ресурсы, требуемые программой, и т.д.

4. Дисциплина работы в пакете САФРА

Рассмотрим основные моменты в подходе к разработке и эксплуатации программ при использовании пакета САФРА.

Предполагается, что разработка программы начинается с составления общей схемы счета, которая представляет собой перечисление функциональных имен модулей, отражающих основные этапы решения задачи. Например, в системе OLYMPUS для решения задач эволюционного типа зафиксирована схема счета, представленная на рис. 1.

```
COTROL
        DATA
        INITAL
        OUTPUT(1)
        STEPON
        OUTPUT(2)
        TESEND
        ENDRUN

STEPON
        STIT
        MOTION
        ENERGY
        FIELD
        CVERGE
```

Рис. 1. Схема счета

COTROL есть программная единица, осуществляющая управление всем расчетом. Она последовательно вызывает подпрограмму DATA задания основных параметров конкретного варианта;

— подпрограмму INITAL расчета физических величин на первом временном слое;

— подпрограмму OUTPUT со значением параметра-единица для выдачи на печать значений начальных данных.

Затем вызывается подпрограмма STEPON, результатом работы которой должно быть значение функций на новом временном слое.

После того, как функции на новом временном слое вычислены, вызовом OUTPUT(2) производится пошаговая выдача данных и подпрограммой TESEND проверяется, нужно ли продолжать расчет, или нужно его закончить. В первом случае производится возврат на STEPON, т.е. переход к расчету нового временного слоя, а во втором случае оформляется окончание расчета.

Каждая из указанных выше программных единиц может обращаться к другим подпрограммам. Например, в задачах магнитной гидродинамики для расчета решения на новом временном слое (подпрограмма STEPON) мы должны решить последовательно уравнение движения MOTION, уравнение энергии ENERGY, уравнение поля FIELD и проверить CVERGE после этих расчетов, сошлись ли итерации. Если итерации не сошлись, мы снова последовательно вызываем указанные подпрограммы, а в противном случае подпрограмма STEPON свою работу заканчивает.

Таким образом, в процессе разработки внутренней структуры отдельного модуля в нем выделяются такие относительно самостоятельные части, которые предполагается оформить тоже в виде модулей. Такое выделение и накопление модулей различных уровней называется модуляризацией программы. При составлении программы из модулей обращения к модулям могут оформляться двумя различными с технологической точки зрения способами. Первый способ предполагает обращение к модулю с помощью соответствующей конструкции языка программирования, осуществляющей вызов подпрограммы (например, оператором CALL), и в этом случае вызываемый модуль оформляется как самостоятельная программная единица (например, SUBROUTINE). При втором способе обращение оформляется в виде специальной конструкции, являющейся по существу макровызовом. При таком способе вызываемые модули оформляются как вставки (макроопределения). Выбор конкретного способа оформления делается исходя только из соображений удобства и эффективности программной реализации. Работы по отладке программы могут вестись одновременно в нескольких направлениях, каждое из которых, по существу, связано с отладкой одного или группы модулей из числа определенных на данный момент в результате модуляризации программы. Каждому направлению ставится в соответствие версия программы, которая представляет собой перечень соответствий между функциональными именами модулей и архивными именами их реализаций. В число этих реализаций на стадии разработки программы могут входить модули двух типов. Во-первых, „реальные" модули, полностью отвечающие функциональному назначению, ассоциируемому с данным функциональным именем, и, во-вторых, „имитирующие" модули, отвечающие функциональному назначению лишь

в той мере, в какой это диктуется соображениями отладки. При этом этап разработки программы заключается в создании в архиве „реального" модуля, отвечающего некоторому функциональному имени, причем для одного функционального имени может быть получено несколько различных по программной реализации модулей. Такая организация разработки дает возможность, по-существу, исключить этап автономной отладки модулей, поскольку наличие в версии зафиксированных соответствий между функциональными и архивными именами обеспечивает имитацию работы отлаживаемого модуля в контексте всей программы. Логическим завершением этого процесса, напоминающего программирование задач по принципу „сверху вниз", является получение версии или версий, в которых не будет ссылок на „имитирующие" модули. Если такая версия не содержит свободных функциональных имен, то совокупность упоминаемых в ней архивных имен определяет единственный эксплуатационный вариант программы. Если же в версии содержатся свободные функциональные имена, то с ее помощью задается некоторый класс эксплуатационных вариантов программы. Каждый из этих вариантов получается путем задания определенной совокупности архивных имен, соответствующих свободным функциональным именам в версии.

5. Язык заданий

Работа с пакетом ведется посредством выполнения задания, написанного пользователем на языке заданий. Задание состоит из совокупности пунктов задания, выполняемых последовательно и определяющих такие действия, как

— внесение изменений в архив (запись и исключение модулей);

— препроцессирование (т.е. предварительную обработку) текстов отдельных модулей путем замены ссылок на вставки соответствующими текстами вставок;

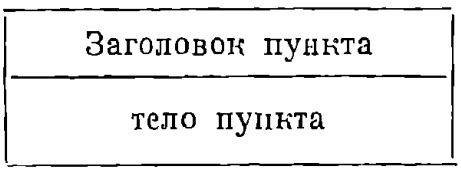
— трансляцию модулей;

— сборку готовой программы из транслированных модулей;

— передачу программы на счет;

— а также некоторые другие виды работ.

Каждый пункт задания состоит из заголовка пункта задания либо из заголовка пункта и тела пункта задания.



Заголовок пункта представляет собой строку (перфокарту), содержащую до 80 символов и имеющую следующий формат записи:

$$\left\langle \begin{matrix}\text{признак}\\ \text{заголовка}\end{matrix} \right\rangle \langle\text{операция}\rangle \left\langle \begin{matrix}\text{тип}\\ \text{модуля}\end{matrix} \right\rangle : \left\langle \begin{matrix}\text{архивное имя}\\ \text{модуля}\end{matrix} \right\rangle .$$

Признак заголовка начинается с первой позиции строк. Признак заголовка должен быть одинаковым для всех пунктов одного задания и может меняться от задания к заданию. Признаком заголовка может служить любая последовательность длиной не более трех символов, составленная из следующих символов:

$$+, \quad -, \quad *, \quad /$$

Признак заголовка не должен содержать пробелов. Далее в настоящем описании в качестве признака заголовка мы используем символ +. Принципиальная возможность разнообразия признаков вводится для того, чтобы можно было отличать заголовки пунктов языка заданий от других предложений в рамках кодировки, принятой в той или иной вычислительной системе.

После признака заголовка может следовать произвольное число пробелов (в частности, ни одного), затем следует тип пункта, указывающий операцию, для выполнения которой служит данный пункт задания, далее следует один или несколько пробелов или запятых, затем указывается тип модуля, над которым должна быть проведена данная операция, затем символ двоеточие и после него указывается имя модуля. Позиции строки с номерами 73–80 игнорируются. Архивным именем модуля может служить идентификатор, т.е. любая комбинация из букв или цифр длиной не более шести символов, начинающаяся с буквы. В имени не могут содержаться пробелы.

Тело пункта — это последовательность предложений языка заданий, либо другая информация, заключенная между заголовком данного пункта и заголовком следующего пункта, или концом задания, если данный пункт последний.

В любом месте задания могут быть записаны предложения = комментарии. Они имеют следующий формат записи

$$\left\langle \begin{matrix}\text{признак}\\ \text{заголовка}\end{matrix} \right\rangle \text{C} \qquad \langle\text{текст}\rangle .$$

Комментарии могут служить для пояснения отдельных конструкций задания и не оказывают никакого влияния на его выполнение. Комментарии, встретившиеся в теле пункта операции STORE, предназначенного для записи в архив нового модуля, запоминаются и воспро-

изводятся при распечатке текста данного модуля, но исключаются при сборке из модулей конкретной задачи.

Протоколом выполнения задания является его листинг, печатающийся на АЦПУ. Если в процессе анализа некоторого пункта выявляется какая-либо ошибка, в листинге печатается соответствующее сообщение, и, в зависимости от серьезности ошибки, отключается выполнение данного и части последующих пунктов. Для каждого из невыполненных пунктов при этом печатается сообщение:

!!!!! БЫЛИ ОШИБКИ — ПУНКТ НЕ ВЫПОЛНЕН

С другой стороны, после каждого выполненного пункта печатается сообщение о результате его работы, например:

+++++ НОВЫЙ МОДУЛЬ ДИСПАК СОЗДАН

или

+++++ СФОРМИРОВАНА ЗАДАЧА ШИФР 031100

Занести в архив новый модуль. (Операция STORE)

Если пользователь хочет завести в архиве новый модуль определенного типа с определенным архивным именем, он должен указать об этом системе. Для этой цели служит пункт задания, заголовок которого имеет вид

$$+\text{STORE}\ \left\langle \begin{matrix}\text{тип}\\ \text{модуля}\end{matrix} \right\rangle : \left\langle \begin{matrix}\text{архивное имя}\\ \text{модуля}\end{matrix} \right\rangle$$

Телом данного пункта задания является текст создаваемого модуля.

В результате выполнения данного пункта задания в архиве появится модуль указанного в заголовке пункта типа с данным архивным именем. Записанный в архив модуль можно использовать в последующих пунктах того же задания и в последующих заданиях. Любые использования этого модуля осуществляются путем ссылки на его архивное имя.

Тип модуля в заголовке пункта STORE может быть опущен. В этом случае считается, что создаваемый модуль имеет тип MACRO.

Пример. Занесение в архив программной единицы.

```
+STORE PRUNIT: OLMU 04
      SUBROUTINE RVAR (KNAME, PVALUE)
C
C 0.4 ПЕЧАТЬ ИДЕНТИФИКАТОРА И ЗНАЧЕНИЯ
C     ПЕРЕМЕННОЙ
```

```
C
      COMMON /COMBAS/
     1 ALTIME, CPTIME, NLEDGE, NLEND, NLRES, NONLIN,
     2 NOUT, NPRINT, NREAD, NREC, NRESUM, NSTEP
      LOGICAL
     1 NLEND, NLRES
C  - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      DIMENSION KNAME(2)
C  - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      WRITE(NOUT, 9900), (KNAME(J), J=1,2), PVALUE
      RETURN
9900  FORMAT(4X,2A6,2H =,E12.4)
      END
```

В результате выполнения этого пункта задания в архив занесется модуль типа программная единица с именем OLMU 04, тело которого представляет собой фортранную подпрограмму с именем RVAR.

Исключить модуль из архива. (Операция DELETE)

Операция DELETE позволяет исключить модуль из архива. Заголовок пункта задания имеет вид

$$+\ \text{DELETE}\ \left\langle \begin{matrix}\text{тип}\\ \text{модуля}\end{matrix} \right\rangle : \left\langle \begin{matrix}\text{архивное имя}\\ \text{модуля}\end{matrix} \right\rangle$$

Тело данного пункта пусто.

Пример. В результате выполнения пункта задания

```
+DELETE PRUNIT : OLMU 04
```

модуль программная единица с именем OLMU 04 будет из архива исключен.

Вставить текст. (Предложение INSERT)

При работе с пакетом часто оказывается полезным выделить некоторые части, общие для нескольких модулей, и рассматривать их как самостоятельные объекты. Такое выделение можно реализовать посредством модуля типа MACRO, служащего для выполнения макроподстановок.

Пример. Блок общей памяти COMBAS в модуле OLMU 04 нужен не только для подпрограмм RVAR, но и для целого ряда других подпрограмм. Выделим его в модуль типа MACRO:

```
+STORE MACRO : OLMC 11
      COMMON /COMBAS/
```

```
      1 ALTIME, CPTIME, NLEDGE, NLEND, NLRES, NONLIN,
      2 NOUT, NPRINT, NREAD, NREC, NRESUM, NSTEP
        LOGICAL
      1 NLEND, NLRES
```

Если некоторая часть текста модуля оформлена в виде другого модуля, то перед выходом на выполнение модуля, например, на трансляцию программной единицы, система должна произвести вставку в текст модуля. Указание на необходимость вставок осуществляется предложением INSERT, имеющим формат, аналогичный формату заголовка пункта задания:

$$+\ \text{INSERT}\ \left\langle \begin{matrix}\text{тип}\\ \text{модуля}\end{matrix}\right\rangle : \left\langle \begin{matrix}\text{функциональное имя}\\ \text{модуля}\end{matrix}\right\rangle$$

Предложение INSERT должно быть записано в теле модуля в том месте, в котором нужно произвести вставку.

Пример. Модуль OLMU 04 после оформления модуля OLMC 11 можно записать в архив следующим образом:

```
+STORE PRUNIT : OLMU 04
      SUBROUTINE RVAR(KNAME, PVALUE)
C
C 0.4 ПЕЧАТЬ ИДЕНТИФИКАТОРА И ЗНАЧЕНИЯ
C     ПЕРЕМЕННОЙ
C
+INSERT MACRO : OLMC 11
C  - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      DIMENSION KNAME(2)
C  - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      WRITE(NOUT, 9900) (KNAME(J), J=1,2), PVALUE
      RETURN
9900  FORMAT(4X,2A6,2H =,E12.4)
      END
```

Преимущество аппарата вставок заключается в том, что подстановка участка текста производится системой только при выполнении модуля. Поэтому, например, замена вставки приводит фактически к изменению целого ряда модулей, имеющих ссылку на эту вставку.

Использование вставок позволяет выделить и обрабатывать как самостоятельные модули такие части программ, оформление которых в виде отдельных программных единиц, допускающих автономную трансляцию, нежелательно по каким-либо причинам. (Например, из-за синтаксических затруднений или же из соображений экономии расхода машинного времени на вызовы программных единиц). Кроме

того, вставки, в совокупности с модулями типа вариант, представляют собой довольно гибкое средство формирования текстов из набора написанных ранее макрочастей.

Работа с вариантом. (Предложение MATCH)

Одной из важнейших особенностей пакета является возможность составления различных вариантов сложных программ из существующего набора модулей. Основным средством описания конкретных вариантов программ являются предложения MATCH.

Предложения MATCH устанавливают соответствие между именами, содержащимися в предложениях INSERT (т.е. функциональными именами) с одной стороны и именами модулей в архиве — с другой.

Предложение MATCH имеет формат

$$\text{MATCH}\ \left\langle \begin{matrix}\text{функциональное}\\ \text{имя}\end{matrix} \right\rangle = \left\langle \begin{matrix}\text{архивное}\\ \text{имя}\end{matrix} \right\rangle$$

Пример. Пусть в программе имеется обращение к подпрограмме RVAR, описанной в приведенном ранее примере. Вообще говоря, в архиве пакета может храниться несколько программных единиц с одним и тем же фортранным именем RVAR, но, естественно, с различными архивными именами. Если нам нужно использовать подпрограмму RVAR, оформленную в виде модуля типа PRUNIT под именем OLMU 04, мы должны записать следующее предложение

MATCH RVAR = OLMU 04.

Последовательность предложений MATCH можно оформить в виде отдельного модуля типа вариант.

Пример. Пусть основная программа записана в архив следующим бразом:

```
+STORE PRUNIT : OLM 000
                PROGRAM MAIN
C
C 0.0 ОСНОВНАЯ ПРОГРАММА
C
+ INSERT MACRO : COMBAS
C  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
C    ВРЕМЯ, ЗАКАЗАННОЕ ДЛЯ РАБОТЫ
     CALL JOBTIM(ALTIME)
```

```
C
C     ПРИСВАИВАНИЕ НАЧАЛЬНЫХ ЗНАЧЕНИЙ ПАРАМЕТ-
C     РАМ УПРАВЛЕНИЯ
            CALL BASIC
C
C     УПРАВЛЕНИЕ РАСЧЕТОМ
          CALL COTROL
C
            STOP
            END
```

Она содержит обращение к подпрограммам JOBTIM, BASIC, COTROL, которые хранятся в архиве под именами OLMU 25, OLM001 и OLM002. Мы можем составить следующий модуль, описывающий вариант сборки программы (строки, начинающиеся с +C, — это строки комментария):

```
+C
+C
MATCH COMBAS = OLMC 11
+C
+C ПОДПРОГРАММЫ
MATCH JOBTIM = OLMU 25
MATCH BASIC = OLM 001
MATCH COTROL = OLM 002
```

Если имя, содержащееся в операторе вызова программной единицы языка программирования или в предложении INSERT, не встречается ни в одном предложении MATCH, оно рассматривается как имя модуля в архиве. Таким образом, уточнение функциональных имен заменой их на архивные имена может производиться либо явно, посредством предложения MATCH, либо неявно, по умолчанию, когда функциональное имя совпадает с архивным.

Если некоторое функциональное имя встречается в левых частях нескольких предложений MATCH, то ему в соответствие ставится архивное имя, указанное последним. Поэтому модуль типа вариант, выполняясь после модуля, задающего версию, детализацией которой он является, может вносить изменения в эту версию, предписывая своими предложениями MATCH некоторые замены функциональных имен на архивные имена, отличающиеся от тех архивных имен, которые ставятся в соответствие тем же функциональным именам предложениями MATCH, содержащимися в модуле версии.

6. Общая схема обработки задания

Выполнением задания для системы САФРА управляет монитор, представляющий собой совокупность программных средств, обеспечивающих операционные возможности пакета. Эти средства базируются на штатном программном обеспечении вычислительной машины, т.е. с системной точки монитор пакета имеет статус обычной задачи. Монитор интерпретирует задание пункт за пунктом, выполняя работы, предписываемые каждым очередным пунктом задания до тех пор, пока не будет исчерпано все задание, либо пока не встретится какой-либо „криминальной" пункт, выполнение которого окажется невозможным из-за его некорректного оформления, либо из-за несоответствия реальному состоянию архива.

Основными компонентами системного наполнения пакета являются: интерпретатор языка заданий, препроцессор текстов, внутренняя информационная служба, генератор заданий для мониторной системы „Дубна". Отметим, что в языке заданий нет понятия модуля загрузки, поскольку пакет избавляет пользователя от необходимости оперировать этим понятием и берет на себя всю службу по организации хранения, учета и использования транслированных программных единиц. При выполнении каждого нового задания те конфигурации модулей, которые транслировались ранее, не транслируются заново, а вместо них из системы хранения автоматически подставляются оттранслированные программные единицы.

Выполняя задание пользователя, монитор взаимодействует с архивом системы, организуя внесение туда изменений, а также посылает запросы на выборку из архива нужной информации. Он контролирует и другие внутрисистемные связи, посылая задания на распечатки текстов и взаимодействуя с информационной и планирующей службами системы. Собственная функциональная деятельность монитора состоит в выполнении предписанных процедур по сборке с указанной в задании заменой функциональных имен модулей на архивные имена. Кроме того, монитор организует внешние связи системы с компонентами штатного математического обеспечения ЭВМ (операционной системой, трансляторами и т.д.).

7. Информационная система

С точки зрения пользователя пакет прикладных программ есть вместилище знаний об алгоритмах решения задач из данной предметной области. Однако, зачастую воспользоваться этими знаниями бывает очень сложно, поскольку информация о предметной области и дисциплине работы с программным фондом развитого пакета очень обширна,

а по своей структуре чрезвычайно разнородна. Для работы с пакетом пользователю приходится изучать объемную программную документацию, вследствие чего эффект использования пакета значительно уменьшается, так как эта работа бывает порой сравнима с работой по созданию вычислительного алгоритма. Для того, чтобы достигнуть высокого уровня организации работ при проведении вычислительного эксперимента средствами пакета прикладных программ, необходимо создать средства информационного обслуживания пакета, позволяющие накапливать „знания о знаниях" и производить логическую обработку информации, связанной с проведением вычислительного эксперимента.

Информационная система пакета САФРА решает четыре следующие основные задачи:

1. *Задача ознакомления со структурой пакета.* Цель этой задачи заключается в том, чтобы пользователь, впервые обратившийся к информационной системе, имел возможность выяснить, описания каких объектов заложены в информационную базу и каковы наборы характеристик каждого объекта, или, другими словами, мог выяснить что и о чем можно спрашивать у системы.

2. *Задача изучения характеристик объектов.* Эта задача наиболее важна с практической точки зрения. Действительно, основными объектами, которыми оперирует пакет САФРА, являются архивные модули. Уже сейчас архив пакета содержит около 1000 модулей, основную часть которых составляют модули типа программная единица. Хотя модули типа схема счета, версия, вариант, вставка составляют меньшую часть, но и они несут значительную функциональную нагрузку. Большую долю всех вопросов, возникающих у пользователя в процессе эксплуатации пакета, составляют вопросы типа: каковы эксплуатационные характеристики данной программной единицы, каково назначение данной схемы счета и из каких модулей она состоит, чем характеризуется данный вариант программы, какие изменения локализованы в текстовой вставке и т.д. Цель решения задачи изучения характеристик объектов в первую очередь состоит в получении ответов на эти и аналогичные вопросы относительно характеристик архивных модулей. Попутно при этом решается задача слежения за текущим состоянием архива, что является важным для организации архивной службы пакета.

3. *Задача ознакомления со структурой программы и структурой ее данных.* При желании пользователя подробно ознакомиться с программной реализацией алгоритма, он должен иметь возможность получить ответы на вопросы типа:

— в каком модуле запрограммирован тот или иной алгоритм;
— откуда вызывается данная программная единица;
— каковы входные и каковы выходные параметры данного модуля;
— где определяется и где используется в программе данная переменная, и так далее.

Подобная информация нужна в частности при попытке использовать имеющийся фонд пакета для создания новой версии программы.

4. *Задача конструирования программы из модулей функционального наполнения.* Система, решающая эту информационную задачу, представляет собой собеседника, который осведомлен о том, какие схемы счета, версии и варианты программ имеются в архиве пакета, и какие математические модели они реализуют. Она также, имея детальное представление о назначении и устройстве программных единиц, отвечая на вопросы пользователя, поможет ему выбрать и включить в вычислительную цепочку необходимый алгоритм. При этом формирование задания для пакета на выборку и включение модуля в цепочку может производиться самой информационной системой, так что полученная программа будет заведомо корректной в смысле программирования.

8. Опыт эксплуатации пакета

Приведем некоторые статистические данные. Программный фонд, хранящийся в архиве САФРА, содержит в настоящее время свыше 1000 модулей суммарным объемом свыше 70 000 предложений различных алгоритмических языков. За два года эксплуатации пакета проведены расчеты более 8000 вариантов программ.

Практика работы с пакетом показала, что его применение имеет определенные преимущества по сравнению с использовавшейся ранее организацией работ по конструированию больших программ. В частности, раньше существовал некоторый порог в размерах программы, при переходе через который разработчик ее должен был „связать с ней свою судьбу", т.е., с одной стороны, программа не могла эффективно эксплуатироваться без разработчика, а с другой стороны, основная часть трудовой деятельности разработчика посвящалась сопровождению программы, внесению в нее изменений и организации расчетов конкретных вариантов. Использование пакета САФРА позволило в значительной мере преодолеть упомянутые трудности.

Литература

[1] В. Я. Карпов, Д. А. Корягин, А. А. Самарский, *Принципы разработки пакетов прикладных программ для задач математической физики*, Ж. вычисл. мат. и мат. физ. 18 (1978), 458.

[2] K. V. Roberts, *An introduction to the OLYMPUS system*, Comp. Phys. Communications 7 (1974), 237.

[3] J. P. Christiansen, K. V. Roberts, *OLYMPUS, a standard control and utility package for initial-value Fortran Programs*, ibid. 7 (1974), 245.

[4] М. М. Горбунов-Посадов, В. Я. Карпов, Д. А. Корягин, В. В. Красотченко, В. В. Мартынюк, *Пакет прикладных программ САФРА. Системное наполнение*, Препринт ИПМ АН СССР 85 (1977).

[5] М. М. Горбунов-Посадов, Л. А. Хиздер, *Система ОХРА. Основные понятия. Базовые операции*, Препринт ИПМ АН СССР 67 (1979).

[6] —, —, *Работа с регионами и файлами в системе ОХРА*, Препринт ИПМ АН СССР 81 (1979).

[7] В. Я. Карпов, В. В. Красотченко, *Задачи информационного обеспечения вычислительного эксперимента*, Препринт ИПМ АН СССР 127 (1979).

[8] С. А. Гайфулин, В. Я. Карпов, Т. В. Мищенко, *САФРА. Функциональное наполнение, система OLYMPUS*, Препринт ИПМ АН СССР 27 (1980).

[9] М. М. Горбунов-Посадов, Д. А. Корягин, В. В. Красотченко, *Реализация системного наполнения пакета прикладных программ САФРА (Версия 2.0)*, Препринт ИПМ АН СССР 186 (1979).

Presented to the Semester
Computational Mathematics
February 20 — May 30, 1980