РОССИЙСКИЙ НАУЧНЫЙ ЦЕНТР
"КУРЧАТОВСКИЙ ИНСТИТУТ"

А.Д. Панов

ИАЭ-6019/15

ПАКЕТ ПРОГРАММ ОБРАБОТКИ СПЕКТРОВ SPRO И ЯЗЫК ПРОГРАММИРОВАНИЯ SL.

Препринт ИАЭ-6019/15. М., 1997

Москва • 1997

УДК 539.1.078

Ключевые слова: Математическая обработка данных, пакет программ, язык программирования, интерпретатор

Описывается пакет программ SPRO, предназначенный для обработки данных электронной и ядерной спектроскопии. Дается описание пользовательского интерфейса и возможностей пакета. Пакет SPRO имеет встроенный язык программирования SL, позволяющий дополнять содержимое пакета новыми алгоритмами обработки данных. Дается описание основ языка SL, приводится состав встроенной библиотеки.

The program system SPRO, intended for electron and nuclear data processing, is described. The user's interface and contents of the program system is described. The program system SPRO has a build-in programming language SL, which allows to add the contents of system by new algorithms of data processing. The foundations of SL and contents of build-in library are described.

Оглавление

1 Введение

Пакет программ SPRO предназначен для обработки данных, получаемых с помощью многоканальных анализаторов: обычно это данные электронной или ядерной спектроскопии. Невозможно все разнообразие возникающих задач вместить на постоянной основе и в состав одного пакета, поэтому в среде пакета SPRO был определен встроенный язык программирования SL (сокращение от SPRO Language). Работа с пакетом SPRO строится следующим образом. Войдя в пакет пользователь имеет под рукой большое количество базовых процедур обработки спектров и разнообразного сервиса, с помощью которых можно решить многие стандартные задачи. Если, однако, возникает проблема, которая не может быть решена с помощью стандартных средств, то пользователь для ее решения может дописать свой собственный модуль на языке SL и пользоваться им по мере необходимости, сохраняя при этом под рукой доступ ко всему стандартному функциональному наполнению пакета и сервису. Более того, в SL-программах имеется возможность использовать стандартный пользовательский интерфейс пакета SPRO (меню, графические окна, контекстную помощь и проч.), поэтому SL-программы по своему внешнему оформлению и предоставляемому сервису могут ничем не уступать постоянным программам пакета SPRO. Из SL-программ могут так же вызываться любые программы стандартного функционального наполнения пакета SPRO (вместе с присущим им пользовательским интерфейсом). Поэтому одним из самых тривиальных и часто используемых приложений языка SL является создание пользовательских меню, в которых могут быть собраны наиболее часто используемые пользователем программы.

Во втором разделе будут описаны основные компоненты пользовательского интерфейса SPRO, в третьем разделе кратко описано функциональное наполнения пакета, в четвертом разделе — основы языка SL.

2 Пользовательский интерфейс и основные характеристики пакета

Пакет SPRO написан на языке Си для операционной системы DOS начиная с версии 3.30. Программа может быть выполнена на любом IMB PC-совместимом компьютере начиная с XT, имеющем графический адаптер EGA (256 Кбайт) и выше. На магнитном диске весь пакет занимает немного меньше 1 Мбайт памяти. SPRO может одновременно работать с 10-ю спектрами любой длины до 8192 каналов. Нормально спектры хранятся в компактных двоичных файлах (*.bsp), но пакет поддерживает еще три различных текстовых формата файлов и имеет средства для преобразования всех форматов друг в друга. Кроме того, имеется отдельный SL-модуль, преобразующий спектры в формат, совместимый с системой GRAPHER.

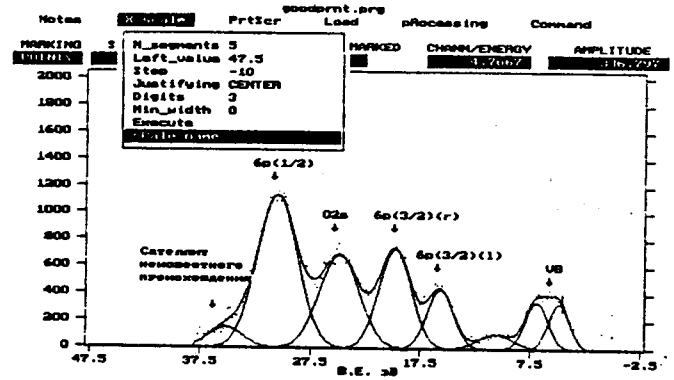

Рисунок 1: Типичный вид экрана пакета SPRO

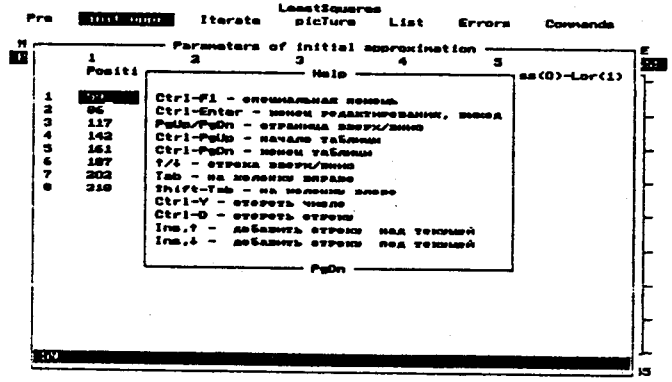

Рисунок 2: Табличный редактор и контекстная помощь.

Пользовательский интерфейс основан на трех основных элементах: иерархической системе меню, графическом окне и редакторе таблиц. Типичный вид экранов при работе с пакетом SPRO приведен на рис. 1 и рис. 2, на которых видны основные элементы интерфейса пользователя.

Меню могут быть горизонтальными, которые размещаются в верхней части экрана, падающими, которые раскрывают содержание пунктов горизонтального меню, и всплывающими, которые похожи на падающие, но появляются независимо от горизонтальных меню. Падающие и всплывающие меню могут содержать параметры, или информацион-

ные строки (см. рис. 1), таким образом, они совмещают в себе свойства обычных меню и диалоговых окон. Все типы меню, в том числе и горизонтальные, могут неограниченное число раз вкладываться друг в друга.

Спектры, которые обрабатываются пакетом, размещаются в десяти разделах памяти. Для просмотра спектров (или, что то же самое — содержимых разделов) определено единственное графическое окно. Клавиша F10 переводит пакет из системы меню в графическое окно (в любой ситуации, даже из вложенного несколько раз падающего меню) и обратно. В режиме графического окна могут быть вызваны на просмотр или спрятаны содержимые различных разделов, и возможны разнообразные манипуляции со спектрами: могут быть размечены каналы или области, изменен масштаб, изменен тип шкалы (линейная или логарифмическая), спектры могут быть растянуты и сжаты по горизонтали, прокручены по горизонтали (если не помещаются целиком), изменен тип представления (точки или гистограмма), смещены вверх-вниз друг относительно друга и др. Разметка спектра и некоторые другие операции осуществляются с помощью графического курсора (мышь не поддерживается). Одновременно на экране может присутствовать участок спектра длиной не более 512 каналов. На экран может быть выдано любое число находящихся в обработке спектров, но при этом только один из них может быть активным. Установка/снятие активности так же происходит в режиме графического окна. Активный спектр выделяется цветом, и подразумевается, что именно он будет обрабатываться, если пользователь воспользуется какой-нибудь из программ пакета, остальные спектры выведены только для просмотра (впрочем, в SL-программах это соглашение может быть нарушено).

Табличный редактор вызывается программами пакета, когда нужно ввести какие-то данные для последующей обработки (например, начальное приближение для разложения спектра или параметры линий для генерации модельной кривой), и для представления результатов вычислений. На рис. 2 показан типичный экран при работе с табличным редактором.

В любой ситуации при работе с пакетом SPRO имеется доступ к контекстно-зависимой помощи (клавиша F1). Из контекстной помощи можно узнать подробности действий по каждому пункту всех типов меню, горячим клавишам, доступным в данной ситуации, и т. д. На рис. 2 на фоне редактируемой таблицы открыто окно помощи с описанием клавиш редактирования. При редактировании таблиц кроме общей помощи имеется специальная помощь по характеру информации конкретной таблицы (Ctrl-F1).

Система SPRO предназначена прежде всего для обработки спектров, и не может заменить специализированные системы графического представления табличных данных типа системы GRAPHER. Тем не менее в состав системы входит несложная система подготовки изображений для печати (реализована в виде внешнего SL-модуля `GOODPRNT.PRG`), которая позволяет нанести надписи на графики и от-

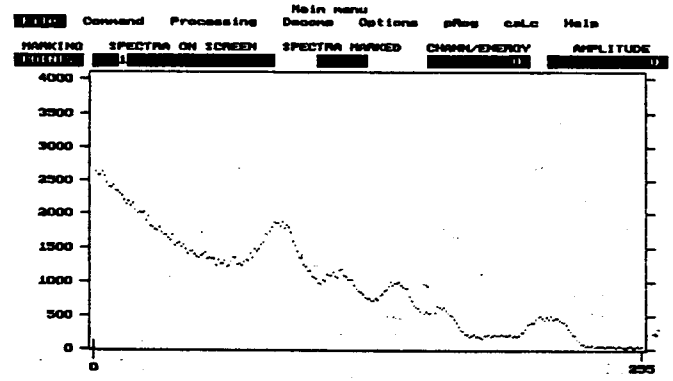

Рисунок 3: Головное меню пакета SPRO.

редактировать вид шкал. На рис. 1 показан вид экрана при работе с модулем `GOODPRNT.PRG`, хотя такое оформление экрана характерно и для пакета SPRO в целом.

3 Функциональное наполнение пакета

В данном разделе мы опишем основные компоненты постоянного функционального наполнения пакета SPRO (версия 2.1), следуя иерархической системе меню. Приводимой информации достаточно для ориентации в пакете, все необходимые подробности могут быть получены при работе с пакетом из контекстной помощи.

Описание начнем с головного меню (рис. 3), которое открывается сразу после запуска пакета (`SPRO.EXE`), и будем последовательно раскрывать содержимое пунктов меню, двигаясь слева направо и сверху вниз.

3.1 File

Программы спасения и загрузки спектров, преобразования форматов файлов, генерация модельных спектров и операции с файлом протокола. Программа раскрывает собственное горизонтальное меню.

`File|Load` — загрузка файла спектра (`*.bsp`) с диска в один из разделов памяти, просмотр комментария файла, и связанные с этим опции.

`File|Save` — Запись спектра из некоторого раздела на диск, определение комментария, и связанные с этим опции.

File|History — Выдача на экран истории загрузки файлов спектров (20 последних загрузок).

File|formatTransform — преобразование двоичных файлов спектров в текстовые и обратно. Поддерживаются следующие текстовые форматы: список одних только амплитуд (*.cts), двухколоночная таблица канал–амплитуда (*.sts), таблица в свободном формате (произвольно расположенная последовательность пар канал–амплитуда, *.lts).

File|phAnton — программа генерации модельных спектров. Эта программа может генерировать широкий набор модельных спектров, которые нужны, прежде всего, для отладки и проверки различных алгоритмов обработки спектров. Программа может:

- Генерировать спектр, состоящий из пиков асимметричной гаусс-лоренцовской формы, и из пиков произвольной, заданной по точкам формы;
- Генерировать многокомпонентные экспоненциальные и гиперболические распадные кривые;
- Генерировать кусочно-линейные сплайны (до 50 вершин);
- Эмулировать разные типы фонов, в том числе фоны неупругого рассеяния электронов с пиками характеристических потерь или более простые;
- Генерировать сумму любого набора перечисленных элементов;
- Вместо самого модельного сигнала генерировать его интеграл или производную;
- Зашумлять результирующий спектр случайными отклонениями с пуассоновским или нормальным распределением.

Параметры модельных кривых могут спасаться в виде файлов специального формата и загружаться с диска.

File|Protocol — позволяет открыть файл протокола. Если файл протокола открыт, то при закрытии любой таблицы и по подтверждению пользователя содержимое этой таблицы вместе с текущей датой, временем и, возможно, комментарием, в текстовом формате будет записано в файл протокола и продолжит уже имеющиеся там записи. Так как результаты обработки спектров всегда выдаются в виде таблиц, то, тем самым, автоматически будет генерироваться отчет о сеансе обработки спектров.

3.2 Command

Разные сервисные операции со спектрами, в том числе калибровка шкалы. Программа открывает собственное горизонтальное меню.

Command|Copy — копирование спектров из раздела в раздел.

Command|Move — перемещение спектров из раздела в раздел.

Command|Delete — обнуление содержимого раздела.

Command|deleteAll — очистка всех разделов.

Command|calibraTion — несколько способов калибровки шкалы и сброс калибровки. Возможна как линейная калибровка (энергия линейно зависит от номера канала), так и индивидуальная калибровка каждого канала из специального файла.

Command|Goto — переход на заданный канал или энергию (обычно имеет смысл для длинных спектров, не помещающихся целиком в окно).

Command|PrtScr — распечатка текущего содержимого графического окна на принтере (поддерживаются матричные 9- и 24-игольчатые принтеры).

Command|View — временное сжатие длинного спектра, таким образом, чтобы он целиком поместился в графическое окно.

Command|Undo — аннулирование последней операции меню **Command** (восстановление стертого спектра и т. д.).

3.3 Processing

Набор базовых относительно простых процедур обработки спектров. Открывает собственное горизонтальное меню.

3.3.1 Processing|coRrect/transform

Набор тех процедур обработки, которые изменяют внешний вид спектра. Открывает собственное горизонтальное меню.

Processing|coRrect/transform|setVal — позволяет вручную поменять значение в нескольких каналах спектра.

Processing|coRrect/transform|Outr — программа удаления выбросов (выбросы размечаются вручную, и на их место вписывается полусумма амплитуд в соседних каналах).

Processing|coRrect/transform|Back — набор программ вычитания фонов (падающее меню). Поддерживается вычитание постоянного и линейного фона, метод Ширли [1], модифицированный метод Ширли с линейной функцией рассеяния, и модифицированный метод Ширли с экспоненциальной функцией рассеяния [2].

Processing|coRrect/transform|Zero — обнуление заданных участков спектра.

Processing|coRrect/transform|shift/Norm — сдвиг спектров (в том числе на дробное число каналов) и нормировка постоянным множителем (может быть, отрицательным).

Processing|coRrect/transform|Add/Sub — сложение спектров с одновременной нормировкой положительным или отрицательным множителем и со сдвигом.

Processing|coRrect/transform|Smooth — сглаживание методом парабол по любому нечетному числу соседних каналов от 5 до 51.

Processing|coRrect/transform|Rough — оргубление спектра, т. е. замена исходного спектра на сжатый в несколько раз путем суммирования по группам каналов.

Processing|coRrect/transform|rEvrs — инвертирование спектра относительно его центра.

Processing|coRrect/transform|Undo — аннулирование последней операции программы coRrect/transform.

3.3.2 Processing|caLculations

Набор процедур обработки спектров, которые сводятся к некоторым вычислениям и не меняют вид спектров. Открывает собственное горизонтальное меню.

Processing|caLculations|Squares — позволяет вычислить площади нескольких размеченных участков спектров.

Processing|caLculations|Positions — падающее меню. Две очень простые программы для определения только положений пиков. В одной программе положения выдаются по ручной разметке каналов, в другой определяются автоматически по положению нуля производной.

Processing|caLculations|Peaks — "эвристическая" программа для анализа формы пиков. Позволяет по нескольким размеченным областям выдать положения, амплитуды, ширины, площади и асимметрии соответствующих пиков.

Processing|caLculations|ConstBase — аппроксимация заданного участка спектра горизонтальной линией (вычисление постоянного фона). Вычисляется так же дисперсия и ошибка среднего.

3.3.3 Processing|Derivations

Позволяет вычислить первую и вторую производную спектра по заданному шаблону, а так же позволяет вычислить интеграл от спектра слева направо или справа налево. Программы выполняют масштабирование результата для удобного просмотра в графическом окне. Имеет собственную опцию Undo. Программа открывает собственное падающее меню.

3.4 Decomp

Этот пункт меню содержит две программы для разложения спектров на перекрывающиеся линии: метод дифференциальных моментов и метод χ^2 (или метод наименьших квадратов).

3.4.1 Decomp|DiffMom

Программа метода дифференциальных моментов. Метод введен А. П. Кучеровым в работе [3]. В работах [4, 5] алгоритм метода был нами усовершенствован (введена обратная связь между итерациями и улучшена сходимость) и распространен на пики асимметричной формы. В данной реализации метод распространен на асимметричные пики гаусс-лоренцовской формы и на пики произвольной формы, задаваемой таблицей. Помимо этого введено вычисление статистических ошибок параметров разложения методом Монте-Карло. Преимуществом метода дифференциальных моментов является очень большая скорость и возможность работать с большим количеством пиков. Однако даже в простых случаях метод дифференциальных моментов дает результаты, неудовлетворительные в смысле критерия χ^2, хотя визуально результаты работы этой программы обычно кажутся приемлемыми. В настоящее время этот метод надо признать устаревшим, так как на современных компьютерах (с процессором 80386 и старше) гораздо лучше использовать более обоснованный метод χ^2 (см. ниже). Метод дифференциальных моментов сохраняет свое значение только при использовании компьютеров IBM XT и AT-80286, поэтому здесь подробно рассматриваться не будет.

3.4.2 Decomp|LeastSquare

Программа метода χ^2-разложения спектров на линии (возможен так же вариант ее использования как простого метода наименьших квадратов).

Спектр описывается набором 5-параметрических линий. Параметрами являются положение, ширина на половине высоты, амплитуда, параметр гаусс-лоренцовской смеси и параметр асимметрии. Контур строится сначала как линейная взвешенная смесь гауссовского и лоренцовского пиков одинаковой ширины, а асимметрия достигается гладким масштабным преобразованием вдоль оси абсцисс. Для описания гаусс-лоренцовской смеси используется параметр p, $0 \le p \le 1$, так что $p=0$ соответствует чисто гауссовской линии, $p=1$ — чисто лоренцовской. Параметр асимметрии ε вводится как отношение $\varepsilon = \Gamma_1/\Gamma_2$, где Γ_1 — часть ширины пика на половине высоты слева от максимума, а Γ_2 — справа от максимума. Форма пика $y(p,\varepsilon,x)$ единичной полной ширины, единичной амплитуды, с положением максимума в точке $x_0 = 0$, и параметрами p и ε задается следующей последовательностью формул:

$$\begin{aligned} \delta &= (\varepsilon^2-1)(\varepsilon^2+1) \\ \beta &= \frac{1}{\sqrt{1-\delta}} + \frac{1}{\sqrt{1+\delta}} \\ \gamma &= \beta x \left(1 + \delta \frac{\beta x}{|\beta x|+1}\right) \end{aligned}$$

$$y(p,\varepsilon,x) \;=\; (1-p)\exp(-\gamma^2\ln 2)+p\frac{1}{1+\gamma^2}.$$

Пик с амплитудой A, шириной Γ и максимумом в точке x_0 запишется как:

$$Y(A,\Gamma,x_0,p,\varepsilon,x)=Ay\left(p,\varepsilon,\frac{x-x_0}{\Gamma}\right).$$

Программа может раскладывать спектры на любое количество линий от 1 до 50, при этом подгоняться могут все 5 параметров $A^j,\Gamma^j,x_0^j,p^j,\varepsilon^j$ для каждой j-й линии. Для оптимизации разложения используется метод χ^2, т. е. по параметрам $A^j,\Gamma^j,x_0^j,p^j,\varepsilon^j$ минимизируется сумма

$$\frac{\chi^2}{N}=\frac{1}{N}\sum_i\frac{1}{\sigma_i^2}\left[y_i-\sum_j Y(A^j,\Gamma^j,x_0^j,p^j,\varepsilon^j,x_i)\right]^2\to\min, \qquad (1)$$

где N — число степеней свободы распределения χ^2, равное числу точек спектра минус число подгоняемых параметров, σ_i — стандартное отклонение экспериментальных амплитуд y_i, а x_i — абсциссы, соответствующие ординатам y_i. Для минимизации суммы (1) использован метод сопряженных градиентов [6, с. 487]. В ходе минимизации любые из параметров $A^j,\Gamma^j,x_0^j,p^j,\varepsilon^j$ могут быть зафиксированы, т. е. могут рассматриваться как априорно известные, что сокращает полное число определяемых величин.

Для вычисления статистических ошибок параметров разложения не подходят линейные методы, основанные на разложении в ряд минимизируемого функционала вблизи минимума, так как статистические ошибки параметров линий часто бывают велики. Для вычисления статистических ошибок использовался метод Монте-Карло. В методе Монте-Карло сначала на основе параметров разложения спектра, полученных в ходе первичной подгонки, строится модельный гладкий контур, аппроксимирующий экспериментальный спектр, а затем с использованием известных значений σ_i зашумляется случайными отклонениями с нормальным законом распределения. Такая процедура повторяется несколько раз (обычно 20–30), и каждый такой "экспериментальный" спектр обрабатывается в точности тем же способом, что и исходный. Получается несколько наборов параметров $\{A^j,\Gamma^j,x_0^j,p^j,\varepsilon^j\}$, статистическая обработка которых позволяет вычислить статистические ошибки разложения. Вычисляются стандартные отклонения для каждого параметра и полная ковариационная матрица площадей линий.

Эта программа представляет собой достаточно сложную систему, поэтому реализована в виде отдельного горизонтального меню.

`Decomp|LeastSquare|Pre` — падающее меню. Начальные установки программы: из какого раздела спектр обрабатывается, в каком разделе записаны величины σ_i^2, рабочий участок спектра.

`Decomp|LeastSquare|init_Appr` — падающее меню. Программа подготовки начального приближения. Позволяет в полуавтоматическом режиме задать начальные параметры линий, начальные диапазоны неопределенности на все параметры, точности определения параметров, возможную фиксацию тех или иных параметров, или редактировать эти величины, если они были заданы раньше.

`Decomp|LeastSquare|Iterate` — падающее меню. В этом пункте можно установить режим трассировки итераций метода сопряженных градиентов и запустить процесс подгонки. Трассировка проводится в графическом виде, с выдачей текущего состояния разложения спектра в графическое окно.

`Decomp|LeastSquare|Picture` — падающее меню. Отрисовка результатов разложения разными способами. Могут быть выданы отдельные пики, суммарный теоретический спектр и сумма любого набора пиков, причем либо в виде просто рисунка, либо с занесением информации в один из разделов памяти, номер которого установит пользователь.

`Decomp|LeastSquare|List` — падающее меню. Программа просмотра различных таблиц с результатами разложения спектра.

`Decomp|LeastSquare|Errors` — падающее меню. Программа вычисления ошибок разложения методом Монте-Карло. Меню содержит установку опций количества модельных спектров и некоторых других, запуск процесса и просмотр результатов вычислений.

3.5 Options

Программа установки различных опций работы пакета SPRO. Представлена падающим меню.

`Options|Representation units`— установка единиц измерения, в которых будут представляться положения (отдельно) и ширины (отдельно) пиков: в каналах, или энергетических единицах (если была проведена калибровка шкалы).

`Options|You can allocate...`— выдает таблицу, в которой показано, сколько разделов для спектров можно разместить в памяти, в зависимости от размера спектров.

`Options|Number of spectra`— установить количество разделов памяти для спектров (от 1 до 10).

`Options|sPectrum length`— установить размер раздела памяти (от 16 до 8192).

`Options|Colors`— установка цветов 14 основных элементов экрана в интерактивном режиме. Оставшиеся примерно 40 цветов можно изменять с помощью SL-программ.

`Options|Save sets`— спасение установок в файл по умолчанию или в специальный пользовательский файл.

`Options|Load sets`— загрузка установок из пользовательского файла.

3.6 Другие пункты головного меню

Оставшиеся три пункта головного меню имеют следующее назначение:

`pRog`— запуск пользовательской SL-программы.

`caLc`— калькулятор с функциями.

`Help`— общая помощь по пакету SPRO, содержит, в основном, описание пользовательского интерфейса.

4 Язык программирования SL

В данном препринте невозможно дать полное описание языка SL и встроенной библиотеки процедур и функций. Будут описаны только наиболее характерные черты языка и приведены некоторые примеры. Полное описание языка содержится в отчете [7] и в дисковом файле `SL_RUS.DOC`, входящем в комплект пакета SPRO.

Язык SL разработан для решения следующего основного круга задач:

1.Реорганизация существующего или создание нового пользовательского интерфейса под конкретные задачи или группы задач, создание пользовательских меню из имеющегося в пакете набора утилит;

2.Математическое программирование: добавление к пакету новых алгоритмов обработки данных, создание недостающих генераторов модельных кривых и т.д.;

3.Связь пакета с нестандартным внешним оборудованием: многоканальными анализаторами, спектрометрами, накопителями информации и т.д., что позволяет сделать пакет полной интегрированной системой сбора и обработки спектроскопической информации.

Программа на языке SL для пакета SPRO представляет из себя текстовый файл (размером до 64 Кбайт), который может быть подготовлен любым текстовым редактором. Для файла рекомендуется расширение `.PRG`. Программа может быть запущена на выполнение либо из головного меню пакета (`Main menu|pRog`), либо, минуя головное меню, сразу после запуска `SPRO.EXE`. В последнем случае имя программы должно быть задано в командной строке с ключом `-P=`, например:

```
spro.exe  -P=myprog.prg
```

Язык SL представляет собой типичный универсальный, структурный, процедурно-ориентированный язык программирования с алгебраической нотацией и включает в себя некоторые элементы языков Си, Паскаль, Фортран-77 и структурных диалектов BASIC. Спецификой языка SL является наличие встроенного интерфейса с пакетом SPRO.

Доступ к данным и программам пакета SPRO из SL-программы осуществляется через набор предопределенных переменных, функций и процедур. Язык SL реализован в виде интерпретатора с предкомпиляцией, поэтому он имеет типичное для таких систем (не очень высокое) быстродействие. Однако из SL-программы могут быть запущены как оверлеи внешние `.EXE`-модули, выполненные с помощью любого компилятора. При этом организован прямой доступ из внешних программ к основным данным пакета. Такие внешние `.EXE`-модули могут выполнить часть работы, связанную либо с большим объемом вычислений, либо со сложными системными операциями, в то время как пользовательский интерфейс может полностью взять на себя SL-программа.

4.1 Основы SL

4.1.1 Имена

Язык SL case-sensitive, т. е. он чувствителен к различию больших и малых букв. Имена могут быть трех типов:

i. Зарезервированные имена. Они представляют различные понятия языка и не могут использоваться ни в каких других смыслах. Зарезервированные имена в точности следующие:

ABORT	ENDIF	IF	PROG	exp
AND	ENDSW	INPUT	RETURN	int
AUTO	EXIT	LOOP	STATIC	isdefine
BEGIN	FGETVAR	MOD	STEP	log
CASE	FINPUT	NEXT	SWITCH	rnd
CONTINUE	FOR	NOP	THEN	sgn
DEFAULT	FPRINT	NOT	VAR	sin
DO	FPUTVAR	NULL	WHILE	sqr
ELSE	FUNC	OR	abs	tan
ELSEIF	GLOBAL	PRINT	atn	
END	GOTO	PROC	cos	

ii. Пользовательские имена. Это имена переменных, функций и подпрограмм, которые пользователь сам вводит в своей программе. Пользовательские имена строятся только из больших и малых букв латинского алфавита и цифр, при этом должны начинаться с буквы и иметь не более 8 символов.

iii. Предопределенные имена. Эти имена представляют различные элементы интерфейса SL-программы и пакета SPRO, а так же библиотечные программы. Все предопределенные имена начинаются с символа подчеркивания "_" и в силу этого не могут быть спутаны с пользовательскими именами. Имеются предопределенные переменные, функции и процедуры.

4.1.2 Операторы и комментарии

Операторы могут быть двух типов: операторы описания переменных и заголовков программных модулей (неисполняемые) и исполняемые. К этому следует добавить метки, а так же операторные скобки BEGIN и END, ограничивающие начало и конец исполняемого кода каждого программного модуля. В каждой строке программы может встретиться ровно один оператор или метка. Длина строки не может превышать 80 символов. Если есть необходимость ввести в программу более длинный оператор, оператор можно разорвать в любом месте, где может встретиться пробел, вставить символ и продолжить оператор на следующей строке. Кроме того, в разрыв оператора до окончания строки может быть вставлен любой комментарий. Кроме комментариев в разрыве оператора, в любом месте программы может встретиться комментарий, начинающийся с двойного символа дроби // и продолжающийся до конца строки. Пример:

```
PROC outStr(x%,y%,string$[])
и
PROC outStr(x%,        \ x-coordinate of output
            y%,        \ y-coordinate of output
            string$[]  \ output string
           ) \\ это заголовок процедуры вывода строки
```

есть две эквивалентные записи одного и того же оператора заголовка процедуры. Пустые строки игнорируются в любом месте программы, но не могут встречаться внутри разорванного оператора, пробелы не играют роли в синтаксисе SL и могут использоваться программистом произвольным образом.

4.1.3 Типы данных

Язык SL поддерживает 5 базовых типов данных:
byte — однобайтовое беззнаковое целое;
int — двухбайтовое целое;
long - четырехбайтовое целое;
float — четырехбайтовое вещественное число;
double— восьмибайтовое вещественное число.

В тексте программы для обозначения типов данных используются специальные символы:

```
byte   - $
int    - %
long   - &
float  - !
double - #
```

Символы типов данных используются в следующих трех случаях:

i. Для явного задания типа числовой константы (если необходимо) Например: 10$, 25%, 0&, 2.5!, 2.5#[1].

ii. При объявлении переменной (в разделе описаний). Например:

```
doubleX#      - переменная doubleX объявлена как double,
text$[100,81] - переменная  text объявлена как массив
                элементов типа byte размером 100*81
                (100 строк текста).
```

iii. Как явный модификатор типа выражения. В этом случае символы типа используются как имена функций: #(i*j) — вычислить i*j и привести результат к типу double.

4.1.4 Простые переменные и массивы. Строки и строчные константы

Переменная может быть либо простой, либо массивом. Массивы могут быть либо одномерными, либо двухмерными (но не более), при обращении к элементам массива используется стандартная запись вида a[i], b[i,j], при этом индексы могут задаваться любыми выражениями. Элементы массива нумеруются начиная с нуля, только целыми числами. Для массивов классов памяти GLOBAL и STATIC индексы могут принимать значения от 0 до 32766, а для массивов класса AUTO — от 0 до 255.

Массивы типа byte ($) обычно используются для представления сток (одномерные) или массивов строк (двухмерные). Конец строки не совпадает с концом массива, но отмечается элементом 0 — т.е. строки представляются в стандартном ASCIIZ - формате. При инициализации строк, а так же при передаче фактических параметров подпрограммам и функциям можно использовать строчные константы. Строчная константа записывается как строка в двойных кавычках. Например, копирование строки в массив-строку string может быть записано так:

```
_copy(string,"Any string of characters.")
```

4.1.5 Общая структура программы

Программа начинается с секции описания глобальных переменных (переменных, видимых из любого места программы). За ней следуют программные модули. Секция описания глобальных переменных может отсутствовать. Программные модули могут быть трех типов: головная программа, процедуры и функции. Головной модуль обязательно должен присутствовать в программе и располагаться текстуально первым среди всех других модулей. Любой программный модуль имеет следующую структуру:

---

[1] Для представления констант типа byte может использоваться форма записи типа 'A', 'c',..., означающая соответствующий ASCII-код

```
заголовок модуля
  секция описания локальных переменных
BEGIN
  исполнимая часть программы
END
```

Заголовки программных модулей имеют различный вид для модулей разных типов:

```
Головной модуль:  PROG
Процедура:        PROC имя_процедуры (param_list)
Функция:          FUNC имя_функции   (param_list)
```

где param_list означает список формальных параметров. Заметим, что в заголовке функции отсутствует информация о типе возвращаемого значения. Последнее связано с тем, что тип значения, возвращаемый функцией, определяется динамически, и одна и та же функция в разных случаях может, вообще говоря, возвращать значения разных типов.

4.1.6 Классы памяти и секции описания переменных

В соответствии с модульной структурой программ переменные могут быть глобальными — видимыми из любой части программы, и локальными — видимыми только из того программного модуля, в котором они были определены. Сами имена программных модулей являются глобальными. В свою очередь локальные переменные могут быть статическими и автоматическими. Значения статических переменных сохраняются между вызовами модуля, в котором они были определены, а значения автоматических — нет, область памяти, которую они занимают, освобождается после выхода из модуля, в котором они были описаны.

Секция описания глобальных переменных. При описании переменной приводится ее имя, в виде суффикса символ типа (обязательно, никаких умолчаний), и размерность массива, если это массив. Например:

```
GLOBAL x%, y%, floatarr![10], dblarr#[5,5]
GLOBAL list$[20,40]
```

При описании глобальных переменных они могут быть явно инициализированы. В качестве инициализаторов могут использоваться числовые и символьные константы, а для $-массивов и строки:

```
a). GLOBAL x% := 1, y% := -1, eps# := 1e-20, c$ := 'A'
b). GLOBAL floatarr![10] := [1,2,3,4,5,6,7,8,9,10]
c). GLOBAL dblarr#[5,5] := [[1,2,3],[4,5,6],[7,8,9,10,11]]
d). DLOBAL string$[81] := "Any string no longer then 80."
e). GLOBAL list$[20,40] := ["string1","string2","string3"]
```


Если явный инициализатор отсутствует, или присутствующих инициализаторов не хватает (как в примере c).), производится неявная инициализация нулями.

Секция описания локальных переменных Секция описания локальных переменных располагается между заголовком программного модуля и словом BEGIN, с которого начинается исполняемая часть модуля. Эта секция состоит из операторов описания статических и автоматических переменных. Они могут чередоваться в произвольном порядке и количество их не ограничено. Оператор описания статических переменных имеет вид:

```
STATIC список описаний
```

и полностью аналогичен оператору GLOBAL. В частности, может быть проведена инициализация статических переменных. Оператор описания автоматических переменных имеет вид:

```
AUTO список описаний
```

и аналогичен операторам GLOBAL и STATIC с той лишь разницей, что инициализировать переменные класса AUTO нельзя, и никакой неявной инициализации не происходит. Переменные класса AUTO важны для организации рекурсивного вызова подпрограмм.

4.1.7 Список формальных параметров

В описании заголовка любого программного модуля кроме головного должен присутствовать список формальных параметров. Синтаксис списка обычный: (параметр,...,параметр). Если модуль не получает параметров, список все равно должен быть, но он будет пустым. Например: PROC mysubr(). При вызове процедуры или функции с пустым списком параметров скобки должны присутствовать: mysubr(). Имя формального параметра является локальным именем внутри программного модуля, в котором оно было определено, и может переопределить любое глобальное имя.

Определены три способа передачи параметров в программный модуль:

i. *Передача параметра по значению.* В списке формальных параметров в этом случае присутствует имя параметра и в качестве суффикса символ типа. Например:

```
PROC mysubr1(x!,y!)
```

Процедура получает два float-параметра. В качестве фактических параметров вместо x и y могут быть подставлены любые выражения. Они будут вычислены, приведены к float-типу и переданы процедуре. Процедура mysubr1 может как угодно менять внутри себя переменные

x и y, и это никак не скажется на значениях переменных вне этого модуля.

ii. *Передача параметров по ссылке.* В списке формальных параметров в этом случае присутствует запись, начинающаяся с зарезервированного слова VAR, затем имя переменной и суффикс типа. Например:

```
PROC mysubr2(VAR x!,VAR y!)
```

Процедура получает по ссылке 2 float-параметра x и y. В качестве фактических параметров в данном случае могут использоваться либо простые переменные типа float, либо элементы массива типа float, но не выражения. Любое изменение VAR-параметра внутри программного модуля вызовет соответствующее изменение значения фактического параметра.

iii. *Передача массивов.* В списке формальных параметров в этом случае присутствует запись, состоящая из имени формального параметра-массива, суффикса типа и квадратных скобок []. Например:

```
PROC mysubr3(buffer%[])
```

Передача массивов всегда происходит по ссылке, поэтому изменение элементов массива buffer внутри mysubr3 вызовет соответствующие изменения в элементах фактического параметра-массива. При описании формального параметра-массива не надо указывать размерность массива даже в том случае, когда массив должен быть двухмерным. Это связано с тем, что вызванному программному модулю передается и начальный адрес и фактическая размерность массива, в связи с чем возможна его корректная обработка внутри программного модуля. Размерности параметра-массива доступны и на пользовательском уровне; они могут быть получены с помощью системной функции isdefine( ).

При передаче фактического параметра-массива возможны следующие случаи.

i. Передача массива целиком. В этом случае в качестве фактического параметра указывается только имя массива.

ii. Если фактический массив двухмерный, то можно передать только одну из его строк. Например, если buffer был определен как buffer%[10,10], то в вызове mysubr3(buffer[2]) передается только вторая строка массива buffer, т.е. mysubr3 получает на обработку массив целых чисел размером 1 × 10, начинающийся с элемента buffer[2,0].

iii. Можно предать НИЧЕГО: NULL. Это часто удобно, когда некоторые параметры могут быть определены по умолчанию. Синтаксис в этом случае такой: mysubr3(NULL). Тот факт, что фактически ничего не было передано в программном модуле может быть установлен с помощью уже упомянутой системной функции isdefine. Аргументом функции isdefine должно быть формальное имя массива, она возвращает 1, если массив действительно определен и 0 в противном случае.

Кроме того, isdefine устанавливает две предопределенные переменные: _n1% и _n2%. Они соответствуют первой и второй размерности тестируемого массива. Если тестируемый массив был на самом деле одномерным, то будет _n1=1, если NULL, то _n1 = _n2 = 0.

4.1.8 Выражения

Выражения состоят из числовых примитивов, связанных знаками арифметических операций. Арифметические операции могут быть одноместными и двухместными, они распределены по приоритетам. Определены следующие операции (в порядке возрастания приоритетов):

```
Приоритет 0: OR  - логическое "или" (низший приоритет)
Приоритет 1: AND - логическое "и".
Приоритет 2: операции сравнения.
      =              равно;
      <> или ><      не равно;
      >= или =>      больше или равно;
      <= или =<      меньше или равно;
      >              больше;
      <              меньше.
Приоритет 3: аддитивные операции.
      +    сложение;
      -    вычитание.
Приоритет 4: мультипликативные операции:
      *      умножение;
      /      деление, как целочисленное, так и плавающее,
                    в зависимости от типов операндов;
      MOD    остаток от деления (только целые числа).
Приоритет 5: ^    возведение в степень.
Приоритет 6: унарные операции (самый высокий приоритет).
      +      унарный плюс;
      -      унарный минус;
      NOT    логическое отрицание.
```

В SL нет специального логического типа данных. В определенных условиях любое число может трактоваться как истинностное значение, при этом используется следующий алгоритм. Сначала число приводится к целому типу, если полученное целое число есть нуль, то число интерпретируется как "ложь", если не нуль - как "истина". Результатом операций сравнения является 0% или 1%, в зависимости от того, истинно данное отношение, или нет. Логические операции трактуются как один из типов арифметических операций.

Числовыми примитивами, которые связываются в выражениях знаками операций, могут быть:

- числовые константы любых типов;
- простые переменные;

– элементы массивов;
– функции преобразования типов $(), %(), &(), !(), #();
– системные функции abs(), ... ,isdefine()
– предопределенные и пользовательские функции;
– круглые скобки (...).

В SL круглые скобки трактуются как системная функция (подобная sin(x) и т. д.), которая просто возвращает значение своего аргумента, который может быть любым выражением, без изменений. Это автоматически приводит к тому, что выражения в скобках вычисляются в первую очередь.

4.2 Основные управляющие операторы и структуры языка SL

4.2.1 Оператор присваивания

Оператор присваивания имеет синтаксис:

```
lvalue1 := ... lvalueN := expression
```

lvalue — это простые переменные или элементы массивов, в левой части оператора присваивания может встретиться от 1 до 20 членов. В правой части оператора присваивания стоит любое выражение. Оператор присваивания выполняется в SL следующим образом. Сначала вычисляется значение выражения, а затем *независимо* (!) это значение присваивается каждой из переменных левой части равенства. При этом в случае необходимости производится преобразование результата выражения к типу переменной.

4.2.2 Условный оператор

Полная форма условного оператора. Полная форма условного оператора имеет вид:

```
IF expr1
  операторы...
ELSEIF expr2
  операторы...
  ........
ELSEIF exprN
  операторы...
ELSE
  операторы...
ENDIF
```

Здесь expr1...exprN — любые выражения, значения которых будут интерпретироваться как истинностные значения в соответствии с алгоритмом, изложенным выше. Соответствующие части условного оператора будут выполнены или нет, в зависимости от значения этих выражений. Составные части ELSEIF и (или) ELSE могут отсутствовать. Никакие операторные скобки ни здесь, ни в других составных операторах не требуются. Сами ключевые слова IF, ELSEIF, ELSE, ENDIF играют роль операторных скобок.

Сокращенная форма условного оператора. Весь сокращенный условный оператор должен поместиться в одной строке (возможно с переносами):

```
IF expr THEN simple_operator
```

Simple_operator — это такой оператор, который не является составной частью некоторого структурного оператора, например полного IF. Не может так же использоваться еще один сокращенный условный оператор.

4.2.3 Оператор выбора

Оператор выбора задает действие по одной из альтернатив, определяемых целым числом. Его действие аналогично действию условного оператора в полной форме, но он выполняется гораздо быстрее, т.к. в операторе выбора не перебираются все возможные альтернативы, пока не встретится нужная. В расплату он имеет ряд ограничений.

В одном из вариантов синтаксис оператора выбора такой:

```
SWITCH expression
CASE a0:
  операторы...
CASE a1:
  операторы...
  ..........
CASE aN:
  операторы...
DEFAULT:
  операторы...
ENDSW
```

Здесь a0,...,aN — целые константы, причем начиная с a0 они должны располагаться в возрастающем порядке и, обязательно, с шагом 1. Оператор вычисляет выражение expression, приводит результат к целому типу, а потом выполняет те действия, которые следуют за соответствующей альтернативой CASE :. Выполнив эти действия он сразу переходит к выполнению операторов, следующих за концом оператора выбора ENDSW. Если подходящей альтернативы на нашлось, выполняются операторы, следующие за DEFAULT:. Компонента DEFAULT может и отсутствовать, тогда сразу произойдет переход к концу оператора.

Пример (возможный цикл обработки горизонтального меню):

```
DO
  SWITCH _topmenu(3,reset,cursor,"Title:test",menu,NULL)
    CASE -1: EXIT
    CASE  0: subr0()
    CASE  1: subr1()
    CASE  2: subr2()
  ENDSW
LOOP
```

4.2.4 Операторы цикла

В языке SL определены 2 типа операторов цикла: безусловный цикл DO...LOOP и цикл со счетчиком FOR...NEXT.

Безусловный цикл. Синтаксис безусловного цикла таков:

```
DO
  оператор
  ...
  оператор
LOOP
```

Если не будет явного выхода из цикла, то он будет продолжаться бесконечно. Для явного выхода из цикла используется оператор EXIT. Циклы могут вкладываться друг в друга неограниченное число раз, при этом по оператору EXIT происходит выход только из самого внутреннего, включающего данный EXIT, цикла.

Для управления выполнением цикла определен также оператор CONTINUE. Если CONTINUE встретился где-то в середине цикла, то пропускаются все операторы, оставшиеся до конца цикла и происходит переход к операциям, связанным с повторением цикла.

Операторы EXIT и CONTINUE могут быть как угодно глубоко спрятаны в конструкциях типа IF..., SWITCH..., и при этом продолжают корректно работать по отношению к содержащему их циклу.

Цикл FOR...NEXT Наиболее общий синтаксис этого цикла такой:

```
FOR  loopVar:=expr1  WHILE  expr2  STEP  expr3
  оператор
  ....
  оператор
NEXT
```

Цикл выполняется следующим образом. Перед первым входом в цикл переменной цикла loopVar будет присвоено значение выражения expr1.

Затем вычисляется значение выражения expr3 и запоминается в некоторой системной области — это шаг приращения переменной цикла. Далее, каждый раз по достижении конца тела цикла значение переменной цикла будет увеличиваться на этот шаг приращения. Часть оператора цикла "STEP expr3" может отсутствовать, тогда шаг принимается равным 1. Перед каждым новым проходом тела цикла вычисляется значение выражения expr2 (которое может, вообще говоря, быть никак не связанным с переменной цикла) и если значение выражение "истина", то очередной проход выполняется, в противном случае происходит выход из цикла. Операторы EXIT и CONTINUE работают с циклом FOR...NEXT точно так же, как с DO...LOOP. Тип переменной цикла может быть любой, однако это должна быть простая переменная, не могут использоваться элементы массива и любые предопределенные переменные.

4.2.5 Оператор GOTO

Язык SL поддерживает оператор GOTO (локальный, только внутри одного программного модуля), хотя этот оператор часто критикуется как нарушающий структурность программы. Синтаксис оператора GOTO такой:

```
GOTO lablName
```

Имя метки должно встретиться где-то в том же самом программном модуле, что и GOTO. Оно располагается в отдельной строке с двоеточием на конце перед той строкой программы, на которую должен произойти переход. Например:

```
START:
a:=1
......
GOTO START
```

Переход возможен и на самый конец программного модуля:

```
  GOTO TheEnd
  операторы...
  TheEnd:
END
```

4.2.6 Оператор возврата RETURN

Оператор возврата RETURN осуществляет выход из программного модуля в вызвавшую его программу и может встретиться в двух формах:

```
Для выхода из процедуры:                RETURN
Для выхода из подпрограммы-функции:     RETURN expression
```

Значение выражения expression будет возвращено в вызвавшую программу, причем тип этого значения никак не будет изменен. В процедурах роль оператора RETURN может играть оператор конца модуля END, в функциях обязательно нужен явный RETURN с выражением.

4.2.7 Операторы NOP и ABORT

Оператор NOP не вызывает никакого действия, его можно использовать при наличии пустых альтернатив в операторе SWITCH и т. д.

Оператор окончания ABORT вызывает немедленное прекращение выполнения SL-программы и возврат в пакет SPRO. Альтернативный способ окончания программы — достижение оператора RETURN или END в головном модуле PROG.

4.2.8 Вызов процедур и функций

Вызов пользовательской или предопределенной процедуры является оператором, который может занимать отдельную строку, встречаться в сочетании с сокращенной формой IF, или после CASE :. Вызовы всех видов функций кроме системных математических функций abs()...tan() так же могут встречаться, подобно подпрограммам, в виде отдельных операторов (вне выражений). Возвращаемые значения при этом, естественно, теряются.

4.3 Запуск внешних программ

Из SL-программы в принципе можно запустить внешние программы любого происхождения в случае, если они не требуют параметров командной строки. Передача параметров командной строки не поддерживается, вместо этого обеспечен другой механизм передачи информации - через область связи BIOS. Для запуска внешних программ используется предопределенная процедура:

```
_exec(name$[],
      paramb$[], parami%[], paraml&[], paramf![], paramd#[],
      VAR exitcode%)
```

Здесь name$[] - имя вызываемой программы, включая путь, если она не расположена в директории, из которой была запущена SL-программа. Через VAR exitcode% передается код завершения вызванной программы. Из вызванной программы возможен прямой доступ к основным данным пакета, а так же к массивам параметров paramb$[]...paramd#[]. Все массивы параметров - необязательные параметры процедуры _exec, вместо любого из них может быть передан NULL. Массивы param* могут быть как одномерные, так и двухмерные, любого размера. Механизм передачи данных во внешние программы состоит в следующем. Начиная с адреса 4F0H в области данных BIOS расположены 16 байт области межпрограммной связи. Эта область


```
// My first program. file: myfirst.prg

PROG
  STATIC menu$[4,18]:=["Load",              \
                       "Save",              \
                       "Command",           \
                       "coRrect/transform"]
  STATIC cursor%:=0  // начальное положение курсора 0 (Load)
  STATIC reset%:=1   // сначала выдать всю картинку меню
BEGIN
  DO
    SWITCH _topmenu(4,reset,cursor,"myfirst.prg",menu,NULL)
      CASE -1: EXIT  // Было нажато Esc, выход и конец.
      CASE  0: _Load()
      CASE  1: _Save()
      CASE  2: _Command()
      CASE  3: _coRrecttransform()
    ENDSW
    reset:=1 // обновить картинку меню
  LOOP
END
```

Рисунок 4: Пример программы пользовательского меню.

никак не используется системой, поэтому любая программа может поместить туда какие-нибудь данные для своих потомков. Фактически, процедура _exec использует лишь первые 4 байта из этой области, помещая туда полный far-адрес массива, определенного в области данных пакета SPRO и состоящего из 17 элементов (в версии SPRO 2.1). Каждый элемент массива сам является far-адресом некоторой переменной или области данных пакета SPRO [7], 5 из этих адресов соответствуют массивам параметров param*. Используя эту информацию любая программа, способная работать с адресами, может получить прямой доступ к данным пакета SPRO для чтения и записи, а так же к массивам параметров paramb$[],... . Если вместо некоторого параметра был передан NULL, то соответствующий элемент массива адресов будет far-нулем 0000:0000. Для облегчения создания внешних программ для пакета SPRO на языке Си, в состав пакета входит специальный файл определений SL.H.

4.4 Интерфейс с пакетом SPRO

Взаимодействие с пакетом SPRO и доступ к данным пакета осуществляется через набор предопределенных переменных, процедур, функций и операторов. Например, предопределенный двухмерный массив _specrtum[,] дает доступ к разделам памяти пакета, где хранятся обрабатываемые спектры: _spectrum[1,10] означает десятый

канал первого спектра, и т. д. Предопределенная функция

```
_topmenu(numbalt%,
         VAR reset%,
         VAR cursor%,
         title$[],
         menu$[],
         help$[]).
```

дает возможность SL-программе организовать свое собственное горизонтальное меню, аналогичные подпрограммы имеются для доступа ко всем другим стандартным элементам интерфейса пакета SPRO [7]. Кроме того, в языке SL имеется набор предопределенных процедур, соответствующих всем единицам функционального наполнения пакета. При этом имя процедуры в большинстве случаев точно повторяет название соответствующего пункта в системе меню SPRO (плюс символ подчеркивание спереди). Например, пункту меню LeastSquare соответствует вызов процедуры _LeastSquare(), и т. д. Используя эти средства легко создавать пользовательские меню из различных программ SPRO. Рассмотрим пример. Пусть требуется создать пользовательское меню, включающее только следующие пункты:

```
Main menu|File|Load
Main menu|File|Save
Main menu|Command
Main menu|Processing|coRrect/transform
```

Это меню может быть оформлено в виде SL-программы, приведенной на рис. 4.

Мы не имеем здесь возможности далее обсуждать библиотеку SL и предопределенный переменные, и приведем только список определений тех и других, по которым можно составить представление о возможностях языка.

4.5 Предопределенные переменные

_spnumb% - количество разделов для хранения спектров;
_spleng% - длина спектров;
_ch0% - начальный канал линейной калибровки;
_ch1% - конечный канал линейной калибровки;
_val0# - начальное значение шкалы линейной калибровки;
_val1# - конечное значение шкалы линейной калибровки;
_active% - номер активного спектра (для _spwind());
_marker% - положение маркера (для _spwind());
_setscale% - режим установки шкалы по оси ординат (0/1) (для _spwind());
_reset% - режим переустановки экрана (-1,0,1,2,3) (для _spwind());
_autoret% - режим автовозврата (для _spwind());
_spcsclon% - определена ли спец. калибровка (0/1);
_hotkeyson% - доступны ли горячие клавиши в меню (0/1);
_escape% - флаг escape, устанавливается при некоторых ошибках;

_unitmode% - режим представления данных (единицы измерения);
_n1% - первая размерность массива (для isdefine());
_n2% - вторая размерность массива (для isdefine());
_colors%[57] - массив цветов;
_marks%[100] - массив помеченных точек (для _spwind());
_visual%[10] - массив номеров видимых спектров (для _spwind()});
_lststrd%[2] - массив номеров последних помеченных спектров (для _spwind());
_fbuf0![_spleng] - вспомогательный пользовательский массив;
_fbuf1![_spleng] - вспомогательный пользовательский массив;
_fbuf2![_spleng] - вспомогательный пользовательский массив;
_spcscale![_spleng] - массив специальной (поканальной) калибровки;
_spectrum![_spnumb,_spleng] - спектры;
_tabitems![800] - буфер значений ячеек таблицы (для _table());
_tabflags![800] - буфер флагов активности ячеек таблицы (для _table()).

4.6 Библиотека функций, процедур, и операторов

Переход от каналов к энергиям и обратно:

```
FUNC _chanen(chan)
FUNC _enchan(en)
FUNC _dchanden(dchan)
FUNC _dendchan(den)
```

Интерактивное графическое окно (ввод графических данных и просмотр спектров).

```
PROC _spwind()
Данные графического окна:
   _active%
   _marker%
   _setscale%
   _reset%
   _autoret%
   _marks%[100]
   _visual%[_spnumb]
   _lststrd%[2]

PROC _virtpict(specrtum![]) - отрисовка массива сплошной линией
```

Контекстная помощь:

```
PROC _sethelp(helpname$[])
PROC _help(item$[])
```

Поддержка меню:

```
FUNC _topmenu(numbalt%,     \
              VAR reset%,   \
              VAR cursor%,  \
              title$[],     \   (NULL)
              menu$[],      \
```

```
                help$□)      //  (NULL)

FUNC _pulldown(level%,       \
               numalt%,      \
               VAR cursor%,  \
               setswdth%,    \
               menu$□,       \
               sets$□,       \   (NULL)
               help$□)       //  (NULL)

FUNC _popup(xl%,yt%,      \  (-1,-1)
            numbalt%,     \
            VAR cursor%,  \
            setswdth%,    \
            title$□,      \   (NULL)
            menu$□,       \
            sets$□,       \   (NULL)
            help$□)       //  (NULL)

FUNC _ways(numalt%,title$□,menu$□)
FUNC _confirm(guestion$□)
```

Редактирование таблиц:

```
PROC _table(xl%,yt%,      \  (-1,-1)
            ncol%,        \
            nline%,       \
            windhght%,    \
            suprcol%,     \
            save%,        \
            itmwdht%□,    \   (NULL)
            comtitle$□,   \   (NULL)
            titles$□,     \
            comments$□,   \   (NULL)
            helpitem$□,   \   (NULL)
            map%□)        //  (NULL)
```

Входные и выходные данные таблицы:

```
_tabitems![800]
_tabflags%[800]
```

Окно для ввода чисел с клавиатуры:

```
оператор  INPUT mesage$□, lvalue
```

Окно для ввода строк с клавиатуры:

```
PROC _getstring(title$□,dest$□)
```

Преобразование символьного формата в числовой и обратно:

```
PROC _valtostr(dest$□,minwidth%,just%,maxdigit%,value#)
PROC _strtoval(source$□,VAR dest#)
```

Обработка строк:

```
FUNC _instr(start%,string1$□,string2$□)
FUNC _len(string$□)
PROC _string(dest$□,lenght%,symbol$)
PROC _left(dest$□,source$□,numb%)
PROC _right(dest$□,source$□,numb%)
PROC _mid(dest$□,source$□,start%,numb%)
PROC _cat(dest$□,source$□)
PROC _copy(dest$□,source$□)
```

Оператор PRINT (вывод строки на экран между горизонтальным меню и графическим окном):

```
PRINT string1$□,...,stringN$□ [,]
```

Текстовый вывод на экран и текстовые окна:

```
PROC _outstring(x%,y%,color%,string$□)
PROC _textbar(xl%,yt%,xr%,yb%,color%)
PROC _crewindow(xl%,yt%,xr%,yb%,colframe%,colfield%)
PROC _savewindow(numb%,xl%,yt%,xr%,yb%)
PROC _clrwindow(numb%)
```

Прием нажатой клавиши:

```
PROC _getkey(VAR ascii$,VAR scancode$)
```

Файлы: открытие, закрытие, существование:

```
PROC _open(filename$□,filenumb%)
FUNC _exist(filename$□)
PROC _close(filenumb%)
```

Файлы: последовательный текстовый вывод:

```
оператор FPRINT filenumb%,string1$□,...,stringN$□ [,]
PROC _rewrite(filenumb%)
```

Файлы: последовательный текстовый ввод:

```
оператор FINPUT filenumb%, lvalue     (ввод чисел)
PROC _fgetstring(filenumb%,dest$□)  (ввод строк)
```

Файлы: двоичный ввод-вывод и прямой доступ:

```
оператор FGETVAR filenumb%, lvalue
оператор FPUTVAR filenumb%, lvalue
PROC _rewind(filenumb%,startend%)
FUNC _getpos(filenumb%,VAR filerest&)
PROC _setpos(filenumb%,pos&)
```

Работа с директориями и дисками:

```
PROC _currdir(path$□)
PROC _sprodir(path$□)
PROC _setdisk(disknumb%)
PROC _chdir(path$□)
```

Меню файлов (окно ввода имени файла):

```
FUNC _getname(inoutnam$□)
FUNC _crename(inoutnam$□)
```

Запуск внешних программ:

```
PROC _exec(filename$□, \
           paramb$□,   \  (NULL)
           parami%□,   \  (NULL)
           paraml&□,   \  (NULL)
           paramf!□,   \  (NULL)
           paramd#□,   \  (NULL)
           VAR exitcode%)
```

Графика:

```
PROC _getpixel(x%,y%,VAR color%)
PROC _putpixel(x%,y%,color%)
PROC _getbar(xl%,yt%,dx%,dy%,buf$□)
PROC _putbar(xl%,yt%,dx%,dy%,buf$□)
PROC _bar(xl%,yt%,dx%,dy%,color%)
PROC _line(xl%,yt%,xr%,yb%,color%)
PROC _graphtext(x%,y%,color%,just%,direct%,string$□)
```

Сообщения:

```
PROC _topmes(mesage$□)
PROC _clrtopmes()
PROC _centrmes(mesage$□)
```

Восстановление help-экрана окна интерактивной графики:

```
PROC _rstrhelp()
```

Время и звук:

```
PROC _sound(freq%,waitms%)
```

Системные математические функции:

```
abs(x)  atn(x)  cos(x)  exp(x)  int(x)  rnd(x)
sgn(x)  sin(x)  sqr(x)  tan(x)  log(x)
```

Другие математические функции:

```
FUNC _nrnd(x)     (нормальное случайное число со ст. откл. x)
FUNC _prnd(x)     (пуассоновское случайное число со средним x)
FUNC _unitpeak(gs0lr1#, asymm#, x#)
FUNC _unitpeksqr(gs0lr1#, asymm#)
```


Математические процедуры:

```
PROC _randomize(start%)

PROC _derivative(mode%,       \ 0 - минимализация, 1 - работа
                 pattern%,    \ 3,5,...,51
                 chan%,       \
                 spectrum!□,  \
                 VAR m0#,     \
                 VAR m1#,     \
                 VAR m2#)
```

Литература

[1] D. A. Shirley. – High-resolution X-ray photoelectron spectrum of the valence band of gold. – Phys. Rev. B, 1972, V. 5, N°5, P. 4702–4714.

[2] А. Д. Панов. – Мажоритарная оценка точности алгоритма выделения упругого сигнала в конверсионной спектроскопии изомера урана-235. – Препринт ИАЭ-5976/2, М.: 1996, 14 стр.

[3] А. П. Кучеров. – Вычисление величин, характеризующих спектральные полосы. – Журн. Прикл. Спектроскопии, 1984, Т. 41, N°1, С. 79–82.

[4] А. Д. Панов. – Автоинтерактивная программа для разложения сложных контуров на отдельные линии. Описание и инструкция по эксплуатации. – Отчет ИАЭ 50.05/100, 1987, 91 стр.

[5] А. Д. Панов. – Автоинтерактивная программа для разложения сложных контуров на отдельные линии. – ВАНТ, сер. Общая и ядерная физика, 1988, вып. 2(42), С. 92.

[6] В. В. Иванов. – Методы вычислений на ЭВМ. – Киев: Наукова Думка, 1986, 583 стр.

[7] А. Д. Панов. – Программирование в среде пакета SPRO. Язык программирования SL. – Отчет ИАЭ 50.05/109, 1993, 59 стр.