3. Задачник и обучающее множество

Эта глава посвящена одному из наиболее важных и обделенных вниманием компонентов нейрокомпьютера – задачнику. Важность этого компонента определяется тем, что при обучении сетей всех видов с использованием любых алгоритмов обучения сети необходимо предъявлять примеры, на которых она обучается решению задачи. Источником данных для сети является задачник. Кроме того, задачник содержит правильные ответы для сетей, обучаемых с учителем.

3.1 Обсуждение Задачника

В этом разделе рассматриваются основные структуры и функции компонента задачник. Отметим, что задачник рассматривается только с точки зрения его использования нейронной сетью. Совершенно очевидно, что невозможно предусмотреть всех вариантов интерфейса между пользователем и задачником. Действительно, было бы странно, если бы в одном и том же интерфейсе обрабатывались задачники, содержащие только числовые поля, задачники, содержащие исключительно графическую информацию и задачники смешанного типа.

3.1.1 Структуры данных задачника

С точки зрения нейрокомпьютера задачник представляет собой прямоугольную таблицу, поля которой содержат информацию о входных данных примеров задачи, правильные ответы и другую информацию. На данный момент существует три основных способа хранения однотипных данных – базы данных, электронные таблицы, текстовые файлы. Какой из них является оптимальным для хранения данных в задачнике? Основными критериями являются удобство в использовании, компактность и универсальность. Поскольку задачник должен хранить однотипные данные и предоставлять их для обработки другим компонентам нейрокомпьютера, а не производить вычисления, то функционально задачник должен являться базой данных. Наиболее подходящим кажется формат табличных (реляционных) баз данных.

В современных операционных системах предусмотрены различные способы обмена данными между приложениями. Наиболее универсальным является обмен в символьном формате. Мы предлагаем использовать именно символьный формат данных для обмена между приложениями. Вопрос конкретной реализации обмена оставим за пределами рассмотрения, поскольку это чисто технический вопрос. Мы полагаем, что вне зависимости от того, каким путем и из какого приложения данные попали в задачник, их представление должно быть одинаковым (принятым в данной реализации задачника).

3.1.2 Поля задачника

Далее будем полагать, что задачник является реляционной базой данных из одной таблицы. Каждому примеру соответствует одна запись базы данных. Каждому данному – одно поле. В данном разделе рассмотрены допустимые типы полей, с точки зрения типа хранящихся в них данных. В разд. «Состав данных задачника» все поля разбиваются по смысловой нагрузке. Все поля базы данных можно разбить на четыре типа – числовые поля, текстовые поля, перечислимые поля и поля типа рисунок.

Числовые поля. Поля числовых типов данных Integer, Long и Real предназначены для хранения различных чисел. Отметим, поля числового типа могут нести любую смысловую нагрузку.

Перечислимые поля. Поля перечислимого типа служат для хранения качественных признаков – полей базы данных, содержащих, как правило, текстовую информацию, но имеющих малое число различных значений. Простейшим примером поля перечислимого типа является поле «пол» – это поле может принимать только два значения – «мужской» или «женский». Поле перечислимого типа не хранит соответствующего текстового значения, вместо него в поле содержится номер значения. Поля перечислимого типа могут быть только входными данными, комментариями или ответами.

Строки (текстовые поля). Поля текстового типа предназначены для хранения тестовой информации. Они могут быть только комментариями.

Рисунок. Поля типа рисунок предназначены для хранения графической информации. В данной работе не устанавливается стандарт хранения полей типа рисунок. Оговаривается только способ хранения полей типа рисунок на диске для файлов задачника, созданного в нейрокомпьютере. При передаче рисунков предобработчику используется формат, согласованный для предобработчика и задачника.

3.1.3 Состав данных задачника

Компонент задачник является необходимой частью нейрокомпьютера вне зависимости от типа применяемых в нем нейронных сетей. Однако в зависимости от решаемой задачи содержимое задачника

может меняться. Так, например, для решения задачи классификации без учителя используют нейросети, основанные на методе динамических ядер [223, 261] (наиболее известным частным случаем таких сетей являются сети Кохонена [98, 99]). Задачник для такой сети должен содержать только векторы входных данных и предобработанных входных данных. При использовании обучаемых сетей, основанных на принципе двойственности, к задачнику необходимо добавить вектор ответов сети. Кроме того, некоторые исследователи хотят иметь возможность просмотреть ответы, выданные сетью, вектор оценок примера, показатели значимости входных сигналов и, возможно, некоторые другие величины. Поэтому, стандартный задачник должен иметь возможность предоставить пользователю всю необходимую информацию. Но если задачник всегда будет отводить память для всех необходимых величин, то для простейших случаев использование памяти будет неэффективным. Таким образом, возникает противоречие, между необходимостью предоставить пользователю всю необходимую ему информацию и эффективностью использования памяти. Рассмотрим более подробно состав данных задачника.

3.1.3.1 Цвет примера и обучающая выборка

Довольно часто при обучении нейронных сетей возникает необходимость использовать в обучении не все примеры задачника, а только часть. Например, такая возможность необходима при использовании метода скользящего контроля для оценки качества обучения сети. Существует несколько способов реализации такой возможности. Кроме того, часто бывает полезно приписать примерам ряд признаков. Так, при просмотре задачника, пользователю полезно видеть степень обученности примера (например, отображать зеленым цветом примеры, которые решаются сетью идеально, желтым – те, которые сеть решает правильно, но не идеально, а красным – те, при решении которых сеть допускает ошибки).

Ту часть задачника, которая в данный момент используется в обучении нейронной сети, будем называть обучающей выборкой. Для выделения из задачника обучающей выборки предлагается использовать механизм «цветов». Если все примеры покрашены в некоторые цвета, то обучающую выборку можно задать, указав цвета примеров, которые необходимо использовать в обучении. В соответствии с предлагаемой схемой, каждый пример покрашен каким–то цветом, а при задании обучающей выборки можно задать комбинацию цветов. Схема работы с цветами детально рассмотрена в разделе «Переменные типа цвет и операции с цветами» главы «Общий стандарт».

Выделенную с помощью механизма цветов часть задачника будем далее называть текущей выборкой. Очевидно, что обучающая выборка является частным случаем текущей выборки.

3.1.3.2 Входные данные

Входные данные – данные, необходимые для решения сетью примера. Входные данные являются вектором. Существует всего несколько видов входных данных. Каждый компонент вектора входных данных может быть:

числом;

полем с ограниченным числом состояний;

рисунком.

3.1.3.3 Комментарии

Пользователю, при работе с задачником, часто бывает необходимо иметь возможность идентифицировать примеры не только по номерам. Например, при работе с медицинскими базами данных полезно иметь поле, содержащее фамилию больного или номер истории болезни. Для этих целей в задачнике может потребоваться хранить вектор комментариев, которые не могут быть использованы в обучении.

3.1.3.4 Предобработанные данные

Предобработанные данные – это вектор входных сигналов сети, полученный из входных данных после предобработки, выполняемой компонентом предобработчик. Хранение задачником этого вектора необязательно. Каждый элемент вектора предобработанных данных является действительным числом. Следует отметить, что любая нетривиальная предобработка, как правило, изменяет размерность вектора.

3.1.3.5 Правильные ответы

Правильные ответы – вектор ответов, которые должна выдать обученная нейронная сеть при решении примера. Этот вектор абсолютно необходим при обучении сетей с учителем. При использовании других видов сетей хранение задачником этого вектора необязательно. Компонентами вектора ответа могут быть как числа, так и поля с ограниченным набором состояний. В первом случае будем говорить о задаче аппроксимации функции, а во втором – о задаче классификации объектов.

3.1.3.6 Полученные ответы

Полученные ответы – вектор ответов, выданных сетью при решении примера. Для задачника хранение этой части примера не обязательно.

3.1.3.7 Оценки

Оценки – вектор оценок, полученный сетью за решение всех подзадач примера (число подзадач равно числу ответов примера). Хранение этого вектора задачником не обязательно.

3.1.3.8 Вес примера

Вес примера – скалярный параметр, позволяющий регулировать интенсивность участия примера в процессе обучения. Для не обучаемых нейронных сетей вес примера может использоваться для учета вклада данных примера в формируемую карту связей. Применение весов примеров зависит от типа используемой сети.

3.1.3.9 Достоверность ответа

При составлении задачника ответы довольно часто получаются как результат измерения или путем логических выводов в условиях нечеткой информации (например, в медицине). В этих случаях одни ответы имеют большую достоверность, чем другие. Некоторые способы построения оценки или формирования карты связей нейронной сети позволяют использовать эти данные. Достоверность ответа является вектором, поскольку ответ каждой подзадачи данного примера может иметь свою достоверность. Каждый элемент вектора достоверности ответа является действительным числом от нуля до единицы.

3.1.3.10 Уверенность в ответе

При использовании некоторых видов оценки (см. главу «Оценка и интерпретатор ответа») интерпретатор ответа способен оценить уверенность сети в полученном ответе. Вектор уверенности сети в ответах (для каждого ответа своя уверенность) может оказаться полезным для пользователя. Каждый элемент вектора уверенности в ответе является действительным числом от нуля до единицы.

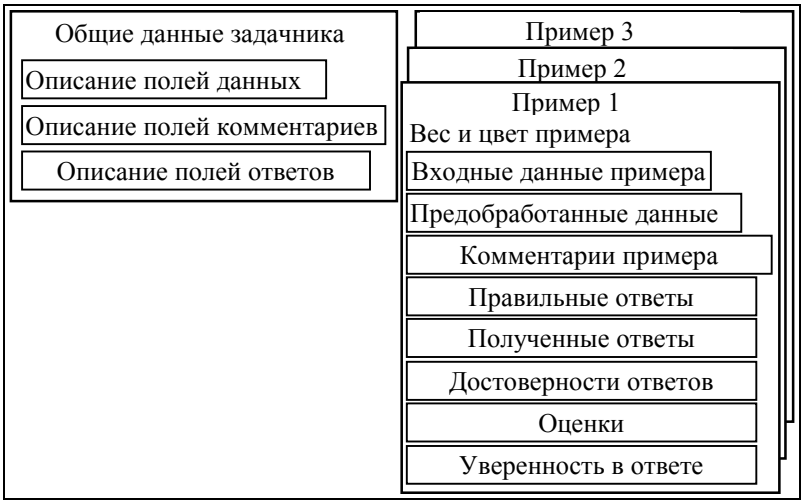

Рис. 1. Схема данных задачника.

Все перечисленные выше векторы можно разбить на четыре типа по структуре:

- Входные данные. Таких векторов обычно два – вектор описания полей данных (содержит описание полей данных: имя поля, его тип и возможно некоторую дополнительную информацию) и собственно вектор данных. Причем каждый пример имеет свой вектор данных, но вектор описания полей данных один для всех примеров задачника. Эти векторы имеют одинаковое число элементов, и их элементы попарно соответствуют друг другу.
- Вектор ответов. При обучении с учителем, в задачнике есть, по крайней мере, два вектора этого вида – вектор описания полей ответов и вектор правильных ответов. Кроме того, возможно хранение в задачнике векторов вычисленных ответов, достоверности ответов и уверенности в ответе. Вектор описания полей ответов – один для всех примеров задачника. Все остальные векторы данного типа хранятся по одному экземпляру каждого вектора на пример.
- Вектор комментариев. Таких векторов обычно только два – вектор описания полей комментариев и вектор комментариев. Вектор описания полей комментариев – один на весь задачник, а вектор комментариев – один на пример.

На рис. 1 приведено схематическое устройство задачника. Такое представление данных позволяет гибко использовать память. Однако следует учесть, что часть полей может переходить из одного вектора в другой. Например, при исключении одного входного данного из использования (см. главу «Контрастер»), соответствующее ему поле переходит из вектора входных данных в вектор комментариев.

3.1.4 Рекомендуемое управление памятью

Наиболее быстрым в работе является хранение всех примеров в памяти. Однако при большом числе примеров хранение всего задачника в памяти является невозможным. Но применение маски используемых данных позволяет задачнику держать в памяти компьютера только те векторы данных примеров, которые будет необходимо возвращать в запросах. Так, например, при обучении сети методом обратного распространения ошибки задачнику требуется хранить в памяти только векторы правильных ответов и предобработанных данных. Все остальные векторы можно хранить на дисках. Такое управление позволяет максимально эффективно использовать оперативную память компьютера. Задачник в ходе работы может выбирать всю свободную память для хранения данных. Однако он должен иметь экономный режим – режим работы при минимальном использовании оперативной памяти.

Запрос «Освободить память» (**FreeMemory**), описанный в главе «Общий стандарт», относится ко всему задачнику в целом. При обработке следующих запросов задачник может снова забирать свободную память. Такое использование памяти позволяет остальным компонентам нейрокомпьютера использовать излишки памяти, потребляемой задачником. Например, при запуске процедуры обучения компоненте нейронная сеть необходимо создать несколько копий синаптической карты. Если обнаружена нехватка памяти, она выдает макрокомпоненту нейрокомпьютер запрос на освобождение памяти, и все компоненты, выполняя этот запрос, оставляют себе только минимально необходимую память. После создания необходимого числа копий синаптической карты остаток памяти могут использовать те компоненты, которым она необходима.

3.2 Стандарт первого уровня компонента задачник

В этом разделе приводится описание хранения задачника на внешнем носителе.

3.2.1 Язык описания задачника

В языке описания задачника используется ряд ключевых слов, специфических для этого языка. Эти ключевые слова приведены в табл. 1.

Таблица 1.

Ключевые слова специфические для языка описания задачника

Ключевое слово	Краткое описание
TaskBook	Заголовок описания задачника
Structure	Заголовок описания структуры задачника
Field	Начало описания поля
Picture	Поле типа рисунок
Source	Описание источника данных
External	Описание внешнего источника данных

Список предопределенных констант языка описания задачника приведен в табл. 2. Эти константы используются при указании типа вектора, к которому принадлежит описываемое поле, при указании используемых векторов в запросе на открытие сеанса и при указании типа вектора в запросах на получение или занесение данных.

Таблица 2

Предопределенные константы

Идентификатор	Значение	Смысл
tbColor	1	Цвет примера
tbInput	2	Входной сигнал
tbPrepared	3	Предобработанные данные
tbAnswers	4	Правильные ответы
tbReliability	5	Достоверность ответа
tbCalcAnswers	6	Полученные ответы
tbCalcReliability	7	Уверенность в ответе
tbWeight	8	Вес примера
tbEstimation	9	Оценки
tbComment	10	Комментарии

3.2.1.1 БНФ языка описания задачника

Обозначения, принятые в данном расширении БНФ и описание ряда конструкций приведены в главе «Общий стандарт» в разделе «Описание языка описания компонентов».

<Описание задачника> ::= <Заголовок задачника> <Описание структуры задачника> <Описание источника данных> <Конец описания задачника>

<Заголовок задачника> ::= **TaskBook** <Имя задачника>

<Имя задачника> ::= <Идентификатор>

<Описание структуры задачника> ::= <Заголовок описания структуры> <Описание полей> <Описание цвета><Описание веса> <Конец описания структуры>

<Заголовок описания структуры> ::= **Structure**
<Описание цвета> ::= **Field** <Имя поля цвет> **tbColor Color End Field**
<Имя поля цвет> ::= <Константа типа ***String***>
<Описание веса> ::= **Field** <Имя поля вес> **tbWeight Real End Field**
<Имя поля вес> ::= <Константа типа ***String***>
<Описание полей> ::= <Описание поля> [<Описание полей>]
<Описание поля> ::= **Field** <Имя поля> <Тип вектора> {<Описание целого поля> | <Описание действительного поля> | <Описание перечислимого поля> | <Описание поля рисунка> | <Описание текстового поля>} **End Field**
<Имя поля> ::= <Константа типа ***String***>
<Тип вектора> ::= {**tbInput** | **tbAnswers** | **tbReliability** | **tbCalcAnswers** | **tbCalcReliability** | **tbEstimation**}
<Описание целого поля> ::= {**Long** | **Integer**}
<Описание действительного поля> ::= **Real**
<Описание перечислимого поля> ::= **Enumerated** <Список имен значений> *;*
<Список имен значений> ::= <Имя значения> [, <Список имен значений>]
<Имя значения> ::= <Константа типа ***String***>
<Описание текстового поля> ::= **String** <Максимальная длина строки>
<Максимальная длина строки> ::= <Константа типа ***Integer***>
<Описание поля рисунка> ::= **Picture** <Размер памяти для рисунка>
<Размер памяти для рисунка> ::= <Константа типа ***Long***>
<Конец описания структуры> ::= **End Structure**
<Описание источника данных> ::= **Source** {<Внешний источник> | <Подготовлено в задачнике>}
<Внешний источник> ::= <Имя приложения, которому нужно передать запрос> <SQL – запрос>
<Имя приложения, которому нужно передать запрос> ::= <Константа типа ***String***>
<SQL – запрос> ::= <Константа типа ***String***>
<Подготовлено в задачнике> – В соответствии с порядком описания полей выводятся символьные представления полей, разделенные символом табуляции (байтом содержащим код 9). Примеры (в терминологии баз данных – записи) разделяются символом конца абзаца (переводом строки – байтом, содержащим код 13). Поля рисунки выводятся в виде последовательности <Размер памяти для рисунка> целых чисел, разделенных пробелами, каждое из которых является десятичным представлением числа (от 0 до 255), содержащегося в соответствующем байте области памяти, отведенной для хранения рисунка.
<Конец описания задачника> ::= **End TaskBook**

3.2.1.2 Описание языка описания задачника

В этом разделе приведено подробное описание дополнительной информации (информации, следующей за типом данных поля) для полей, в блоках описания которых она используется.

Перечислимый тип поля. При использовании перечислимого типа поля в векторах данных хранятся не сами значения, а их номера. Для отображения в редакторе задачника значений полей их необходимо брать из блока описания поля. В списке имен значений блока описания перечислимого поля хранятся символьные константы, первая из которых содержит название состояния, соответствующее неопределенному значению поля; вторая – первому из значений, которые может принимать поле, и т.д.

Строка. Поля типа строка предназначены для хранения символьных строк фиксированной длины. Длина строки задается значением параметра <Максимальная длина строки>.

Рисунок. Поля типа рисунок предназначены для хранения графической информации. Первые семь байт поля имеют смысл, приведенный в табл. 3. В таблице принято обозначение Б1 – величина, хранящаяся в первом байте, Б2 – во втором и т.д. Рисунок разворачивается по строкам, начиная с левого верхнего угла, в непрерывный массив, размером (Б2*256+Б1)(Б4*256+Б3).

Таблица 3

Значение первых семи байт поля типа рисунок

Величина	Значение
Б2*256+Б1	Положительное целое число, задающее размер рисунка по горизонтали в пикселях.
Б4*256+Б3	Положительное целое число, задающее размер рисунка по вертикали в пикселях.
(Б7*256+Б6)*256+Б5	Число цветов, использованных в рисунке

Если число цветов равно единице (черно-белое изображение), то каждый следующий байт содержит восемь пикселей изображения. Самый младший бит восьмого байта соответствует левому верхнему пикселю рисунка. Если число цветов равно трем, то каждый байт, начиная с восьмого, содержит информацию о четырех пикселях. Младшие два бита задают левый верхний пиксель рисунка. Если число цветов от 4 до 15, то каждый байт, начиная с восьмого, содержит информацию о двух пикселях. Младшие четыре бита задают левый верхний пиксель рисунка. Если число цветов от 16 до 255, то каждый байт, начиная с восьмого, содержит информацию об одном пикселе. Значение в восьмом байте соответствует левому верхнему пикселю рисунка. При числе цветов от 256 до 65535 каждые два байта, начиная с восьмого, содержат информацию об одном пикселе (первый пиксель имеет цвет номер Б9*256+Б8). Значение в восьмом и девятом байтах соответствует левому верхнему пикселю рисунка. При числе цветов от 65535 до 16777215 каждые три байта, начиная с восьмого, содержат информацию об одном пикселе (первый пиксель имеет цвет номер (Б10*256+Б9)*256+Б8). Значение в восьмом, девятом и десятом байтах соответствует левому верхнему пикселю рисунка.

Альтернативный способ записи рисунка. Предложенный выше способ хорош своей простотой и плох большим объемом данных. Большинство графических форматов файлов (например GIF) обеспечивают высокую степень компрессии графической информации. Для использования этой возможности при записи поля графической информации на диск предлагается альтернативный формат. В этом формате первые два байта должны быть нулевыми. Поскольку в основном формате записи рисунков эти два байта формировали размер рисунка по горизонтали, нулевая ширина рисунка служит признаком использования альтернативного формата. Следующие пять байт хранят ASCII коды типа графического формата. Используются коды заглавных букв латинского алфавита. Если тип графического стандарта содержит менее пяти символов (PCX, GIF), то тип дополняется символами пробела справа. Следующие четыре байта, с восьмого по одиннадцатый, содержат число байт в графическом файле. Начиная с двенадцатого байта, идет информация, содержавшаяся в графическом файле.

Сопоставление элементов векторов правильных ответов, достоверности, вычисленных ответов, уверенности и оценки производится по следующему правилу. Первому полю типа правильный ответ соответствуют первое поле типа достоверность, первое поле типа вычисленный ответ, первое поле типа уверенность, первое поле типа оценка, второму – вторые и т.д.

3.2.1.3 Неопределенные значения

В практике работы большинство таблиц данных не полны. То есть, часть данных в примерах задачника не известна. Задачник должен однозначно указать предобработчику неизвестные данные. Для этих целей для каждого типа входных данных определено специальное значение - неопределенное. Для передачи неизвестных значений используются следующие величины: 10^{-40} для действительных чисел и 0 для всех типов качественных признаков.

3.2.1.4 Пример описания задачника

В этом разделе приведено описания простого задачника для прогнозирования курса американского доллара к рублю. Задачник содержит три примера. В разделе описания данных вместо символа табуляции использован символ «→», а вместо символа конца абзаца – «↵».

TaskBook CursValuty
Structure
  Field "Цвет" **tbColor Color End Field**
  Field "Вес" **tbWeight Real End Field**
  Field "Дата" **tbComment Real End Field**
  Field "Текущий курс" **tbInput Real End Field**
  Field "Курс на следующий день" **tbAnswers Real End Field**
  Field "Достоверность курса" **tbReliability Real End Field**
  Field "Предсказанный курс" **tbCalcAnswers Real End Field**
  Field "Надежность предсказания" **tbCalcReliability Real End Field**
  Field "Оценка предсказания" **tbEstimation Real End Field**
End Structure
Source
HFFFF→1.0→01.01.97→5773→5774→1.0→5775→0.1→0.07↵
HFFFF→1.0→02.01.97→5774→5776→1.0→5777→0.01→0.7↵
HFFFF→1.0→03.01.97→5776→5778→1.0→5779→0.2→0.007↵
End TaskBook

3.3 Стандарт второго уровня компонента задачник

В этом разделе описаны все запросы компонента задачник в виде процедур и функций. При описании используется синтаксис языков Turbo Pascal и C. В Паскаль варианте приведены заголовки функций и процедур. В C варианте – прототипы функций. Большинство запросов, реализуется в виде функций, сообщающих о корректности завершения операции.

Предполагается возможность одновременной работы нескольких сеансов одного задачника. Например, допускается редактирование задачника и одновременное обучение сети по тому же задачнику.

Все запросы к компоненту задачник можно разбить на следующие группы.

1. Чтение и запись задачника.
2. Начало и конец сеанса.
3. Перемещение по примерам.
4. Определение, получение и изменение данных.
5. Окраска примеров.
6. Установление структуры **Задачника**.
7. Добавление и удаление примеров.
8. Обработка ошибок.

3.3.1 Чтение и запись задачника

К этой группе запросов относятся запросы, работающие со всем задачником в целом. Эти запросы считывают задачник, сохраняют задачник на диске или выгружают ранее считанный или созданный задачник.

3.3.1.1 Прочитать задачник (tbAdd)

Описание запроса:

Pascal:

```
Function tbAdd( CompName : PString ) : Logic;
```

C:

```
Logic tbAdd( PString CompName )
```

Описание аргумента:

CompName – указатель на строку символов, содержащую имя файла задачника.

Назначение – служит для считывания задачника.

Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если в данный момент считан задачник, то генерируется запрос tbDelete. Если запрос tbDelete завершается неуспешно, то генерируется внутренняя ошибка 104 – попытка считывания задачника при открытых сеансах ранее считанного задачника. Управление передается обработчику ошибок. Выполнение запроса прекращается.
3. Первые четыре символа строки CompName составляют слово File. Остальная часть строки содержит имя компонента и после пробела имя файла, содержащего компонент.
4. Если во время выполнения запроса возникает ошибка, то генерируется внутренняя ошибка 102 – ошибка чтения задачника. Управление передается обработчику ошибок. Выполнение запроса прекращается. В противном случае выполнение запроса успешно завершается.

3.3.1.2 Записать задачник (tbWrite)

Описание запроса:

Pascal:

```
Function tbWrite( CompName, FileName : PString) : Logic;
```

C:

```
Logic tbWrite(PString CompName, PString FileName)
```

Описание аргументов:

CompName – указатель на строку символов, содержащую имя задачника.

FileName – имя файла, куда надо записать компонента.

Назначение – сохраняет задачник в файле.

Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.

2. Если в момент получения запроса отсутствует считанный задачник, то возникает ошибка 101 – запрос при отсутствии задачника, управление передается обработчику ошибок, а обработка запроса прекращается.
3. Задачник записывается в файл FileName под именем CompName.
4. Если во время выполнения запроса возникает ошибка, то генерируется внутренняя ошибка 103 – ошибка записи задачника. Управление передается обработчику ошибок. Выполнение запроса прекращается. В противном случае выполнение запроса успешно завершается.

3.3.1.3 Закрыть задачник (tbDelete)

Описание запроса:

Pascal:

Function tbDelete : Logic;

C:

Logic tbDelete()

Назначение – удаляет из памяти ранее считанный задачник.

Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если есть открытые сеансы, то возникает ошибка 105 – закрытие задачника при открытых сеансах. Управление передается обработчику ошибок. Выполнение запроса прекращается.
3. Задачник закрывается. Запрос успешно завершается.

3.3.2 Начало и конец сеанса

К этой группе запросов относятся два запроса, открывающие и закрывающие сеансы работы с задачником.

3.3.2.1 Начало сеанса (InitSession)

Описание запроса:

Pascal:

Function InitSession(NewColor : Color; Oper : Integer; Var Handle: Integer) : Logic;

C:

Logic InitSession(Color NewColor, Integer Oper, Integer* Handle)

Описание аргументов:

NewColor – цвет для отбора примеров задачника в текущую выборку.

Oper – операция для отбора в текущую выборку. Должна быть одной из констант CEqual, CIn, CInclude, Cxclude, CIntersect

Handle – номер сеанса. Начальное значение не важно. В этом аргументе возвращается номер сеанса.

Назначение – начинает сеанс. Отбирает текущую выборку.

Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если аргумент Oper является недопустимым, то возникает ошибка 106 – недопустимый код операции при открытии сеанса, управление передается обработчику ошибок. Сеанс не открывается. Возвращается значение ложь.
3. Создается новый сеанс (в одно-сеансовых задачниках просто инициируется сеанс). Номер сеанса заносится в аргумент Handle.
4. Значения аргументов NewColor и Oper сохраняются во внутренних переменных задачника
5. Указателю текущего примера присваивается состояние «до первого примера»
6. InitSession := Next(Handle) – результат выполнения запроса совпадает с результатом выполнения вызванного запроса «Следующий пример».

3.3.2.2 Конец сеанса (EndSession)

Описание запроса:

Pascal:

Procedure EndSession(Handle : Integer);

C:

void EndSession(Integer Handle)

Назначение – закрывает сеанс.

Описание аргументов:

Handle – номер сеанса.

Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если аргумент Handle не корректен возникает ошибка 107 – неверный номер сеанса. Управление передается обработчику ошибок. Выполнение запроса прекращается.
3. Освобождается вся память, взятая для выполнения сеанса. После этого сеанс завершается.

3.3.3 Перемещение по примерам

В эту группу запросов входят запросы позволяющие управлять положением текущего указателя в текущей выборке.

3.3.3.1 В начало (Home)

Описание запроса:
Pascal:

Function Home(Handle : Integer) : Logic;

C:

Logic Home(Integer Handle)

Описание аргументов:
Handle – номер сеанса.
Назначение – делает текущим первый пример текущей выборки.
Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если аргумент Handle не корректен возникает ошибка 107 – неверный номер сеанса. Управление передается обработчику ошибок. Выполнение запроса прекращается.
3. Указателю на текущий пример присваивается значение «до первого примера»
4. Home := Next(Handle) – результат выполнения запроса совпадает с результатом выполнения вызванного запроса «Следующий»

3.3.3.2 В конец (End)

Описание запроса:
Pascal:

Function End(Handle : Integer) : Logic;

C:

Logic End(Integer Handle)

Описание аргументов:
Handle – номер сеанса.
Назначение – делает текущим последний пример текущей выборки.
Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если аргумент Handle не корректен возникает ошибка 107 – неверный номер сеанса. Управление передается обработчику ошибок. Выполнение запроса прекращается.
3. Указателю на текущий пример присваивается значение «после последнего примера»
4. Home := Prev(Handle) – результат выполнения запроса совпадает с результатом выполнения вызванного запроса «Предыдущий»

3.3.3.3 Следующий (Next)

Описание запроса:
Pascal:

Function Next(Handle : Integer) : Logic;

C:

Logic Next(Integer Handle)

Описание аргументов:
Handle – номер сеанса.
Назначение – делает текущим следующий пример текущей выборки.
Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если аргумент Handle не корректен возникает ошибка 107 – неверный номер сеанса. Управление передается обработчику ошибок. Выполнение запроса прекращается.
3. Если значение указателя равно «после последнего примера», то возникает ошибка 108 – переход за конечную границу текущей выборки, и управление передается обработчику ошибок. В случае

возврата управления в запрос, происходит немедленный выход из запроса с возвращением значения ложь.

4. Если значение указателя текущего примера равно «до первого примера», то присваиваем указателю адрес первого примера задачника. Если адрес в переменной в задачнике нет примеров, то возникает ошибка 108 – переход за конечную границу текущей выборки, и управление передается обработчику ошибок. В случае возврата управления в запрос, происходит немедленный выход из запроса с возвращением значения ложь. В противном случае переходим к шагу 6
5. Указатель перемещается на следующий пример задачника. Если следующего примера задачника нет, то указателю присваивается значение «после последнего примера».
6. Переходим к шагу 5, если не верно условие:

   ((GetColor Oper NewColor) And Last,

   где Oper и NewColor – аргументы запроса InitSession, которым был открыт данный сеанс.
7. Next := Not Last (Переход к следующему примеру завершился удачно, если указатель не установлен в значение «после последнего примера»).

3.3.3.4 Предыдущий (Prev)

Описание запроса:

Pascal:

Function Prev(Handle : Integer): Logic;

C:

Logic Prev(Integer Handle)

Описание аргументов:

Handle – номер сеанса.

Назначение – делает текущим предыдущий пример текущей выборки.

Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если аргумент Handle не корректен возникает ошибка 107 – неверный номер сеанса. Управление передается обработчику ошибок. Выполнение запроса прекращается.
3. Если значение указателя равно «до первого примера», то возникает ошибка 109 – переход за начальную границу текущей выборки, и управление передается обработчику ошибок. В случае возврата управления в запрос, происходит немедленный выход из запроса с возвращением значения ложь.
4. Если значение указателя равно «после последнего примера», то присваиваем указателю адрес последнего примера задачника. Если в задачнике нет примеров, то возникает ошибка 109 – переход за начальную границу текущей выборки, и управление передается обработчику ошибок. В случае возврата управления в запрос, происходит немедленный выход из запроса с возвращением значения ложь.
5. В противном случае шаг 7.
6. Указатель перемещается на предыдущий пример задачника. Если предыдущего примера задачника нет, то указателю присваивается значение «до первого примера».
7. Шаг 6 повторяется до тех пор, пока не выполнится условие:

   ((GetColor Oper NewColor) And First
8. Next := Not Last (Переход к следующему примеру завершился удачно, если указатель не установлен в значение «после последнего примера»).

3.3.3.5 Конец (Last)

Описание запроса:

Pascal:

Function Last(Handle : Integer) : Logic;

C:

Logic Last(Integer Handle)

Описание аргументов:

Handle – номер сеанса.

Назначение – возвращает значение истина, если текущим является состояние «после последнего примера», и ложь – в противном случае.

Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если аргумент Handle не корректен возникает ошибка 107 – неверный номер сеанса. Управление передается обработчику ошибок. Выполнение запроса прекращается.

3. Возвращает значение истина, если текущим является состояние «после последнего примера», и ложь – в противном случае.

3.3.3.6 Начало (First)

Описание запроса:

Pascal:

Function First(Handle : Integer): Logic;

C:

Logic First(Integer Handle)

Описание аргументов:

Handle – номер сеанса.

Назначение – возвращает значение истина, если текущим является состояние «перед первым примером», и ложь в противном случае.

Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если аргумент Handle не корректен возникает ошибка 107 – неверный номер сеанса. Управление передается обработчику ошибок. Выполнение запроса прекращается.
3. Возвращает значение истина, если текущим является состояние «перед первым примером», и ложь в противном случае.

3.3.3.7 Пример номер (Example)

Описание запроса:

Pascal:

Function Example(Number : Long; Handle : Integer) : Logic;

C:

Logic Example(Long Number, Integer Handle)

Описание аргументов:

Number – номер примера, который должен быть сделан текущим. Нумерация примеров ведется с единицы.

Handle – номер сеанса.

Назначение – делает текущим пример текущей выборки с указанным номером.

Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если аргумент Handle не корректен возникает ошибка 107 – неверный номер сеанса. Управление передается обработчику ошибок. Выполнение запроса прекращается.
3. Указатель устанавливается в состояние «до первого примера».
4. Number раз выполняем запрос Next.
5. Example := Not Last (Если не установлено состояние «после последнего примера», то запрос выполнен успешно).

3.3.4 Определение, получение и изменение данных

К данной группе запросов относятся запросы позволяющие получать данные из задачника, заносить данные в задачник и сбросить предобработку (необходимо выполнить данный запрос после изменений в данных или предобработчике, если задачник хранит векторы предобработанных данных)

3.3.4.1 Дать пример (Get)

Описание запроса:

Pascal:

Function Get(Handle : Integer; Var Data : PRealArray; What : Integer) : Logic;

C:

Logic Get(Integer Handle, PRealArray* Data, Integer What)

Описание аргументов:

Handle – номер сеанса;

Data – указатель на массив, в котором должны быть возвращены данныt;

What – одна из предопределенных констант tbColor, tbInput, tbPrepared, tbAnswers, tbReliability, tbCalcAnswers, tbCalcReliability, tbWeight, tbEstimation, tbComment

Назначение – возвращает всю указанную в запросе «Параметры примера» информацию.

Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если аргумент Handle не корректен возникает ошибка 107 – неверный номер сеанса. Управление передается обработчику ошибок. Выполнение запроса прекращается.
3. Если аргумент What имеет недопустимое значение, то возникает ошибка 110 – неверный тип вектора в запросе Get. Управление передается обработчику ошибок. Выполнение запроса прекращается.
4. Если текущий указатель указывает на одно из состояний «до первого примера» или «после последнего примера», то возникает ошибка 111 – попытка чтения до или после текущей выборки. Управление передается обработчику ошибок. Запрос завершается неуспешно.
5. Если в аргументе What указан вектор предобработанных данных, но в текущем примере он отсутствует, то генерируется запрос предобработать данные. Если предобработка завершается успешно, то полученный вектор предобработанных данных включается в пример, в противном случае выполнение запроса прекращается. Возвращается значение ложь.
6. В элементы массива, на который указывает аргумент Data, копируются данные из того вектора данных текущего примера, который указан в аргументе What. Если требуемый вектор в задачнике отсутствует, то возникает ошибка 112 – данные отсутствуют и запрос завершается со значением ложь. В противном случае запрос успешно завершается.

3.3.4.2 Обновить данные (Put)

Описание запроса:

Pascal:

```
Function Put( Handle : Integer; Data : PRealArray; What : Integer ) : Logic;
```

C:

```
Logic Put(Integer Handle, PRealArray Data, Integer What)
```

Описание аргументов:

Handle – номер сеанса

Data – указатель на массив, в котором переданы данные, которые должны быть занесены в задачник.

What – одна из предопределенных констант tbColor, tbInput, tbPrepared, tbAnswers, tbReliability, tbCalcAnswers, tbCalcReliability, tbWeight, tbEstimation, tbComment

Назначение – обновить данные текущего примера

Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если аргумент Handle не корректен возникает ошибка 107 – неверный номер сеанса. Управление передается обработчику ошибок. Выполнение запроса прекращается.
3. Если аргумент What имеет недопустимое значение, то возникает ошибка 113 – неверный тип вектора в запросе Put. Управление передается обработчику ошибок. Выполнение запроса прекращается.
4. Если текущий указатель указывает на одно из состояний «до первого примера» или «после последнего примера», то возникает ошибка 111 – попытка чтения до или после текущей выборки. Управление передается обработчику ошибок. Запрос завершается неуспешно.
5. Если устанавливается вектор входных данных, то для текущего примера должен быть освобожден вектор предобработанных данных.
6. В данные примера копируются значения, указанные в массиве Data. Запрос успешно завершается.

3.3.4.3 Сбросить предобработку (RemovePrepare)

Описание запроса:

Pascal:

```
Procedure RemovePrepare;
```

C:

```
void RemovePrepare()
```

Назначение – отмена предобработки всех ранее предобработанных примеров.

Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. У всех примеров задачника освобождаются вектора предобработанных данных.

3.3.5 Окраска примеров

В данный раздел помещены запросы для работы с цветами. Отметим, что цвет примера, возвращаемый запросом GetColor можно получить также с помощью запроса Get.

3.3.5.1 Дать цвет примера (GetColor)

Описание запроса:

Pascal:

Function GetColor(Handle : Integer) : Color;

C:

Logic GetColor(Integer Handle)

Описание аргументов:

Handle – номер сеанса

Назначение – возвращает цвет текущего примера.

Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если аргумент Handle не корректен возникает ошибка 107 – неверный номер сеанса. Управление передается обработчику ошибок. Выполнение запроса прекращается.
3. Если текущий указатель указывает на одно из состояний «до первого примера» или «после последнего примера», то возникает ошибка 111 – попытка чтения до или после текущей выборки. Управление передается обработчику ошибок. Запрос завершается неуспешно.
4. Возвращается цвет текущего примера.

3.3.5.2 Покрасить пример (PaintCurrent)

Описание запроса:

Pascal:

Function PaintCurrent(Handle : Integer; NewColor, ColorMask : Color; Oper : Integer) : Logic;

C:

Logic PaintCurrent(Integer Handle, Color NewColor, Color ColorMask, Integer Oper)

Описание аргументов:

Handle – номер сеанса.

NewColor – новый цвет для окраски примера.

ColorMask – маска цвета для окраски примера.

Oper – операция, используемая при окраске примера. Должна быть одной из констант COr, CAnd, CXor, CNot.

Назначение – изменяет цвет текущего примера.

Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если аргумент Handle не корректен возникает ошибка 107 – неверный номер сеанса. Управление передается обработчику ошибок. Выполнение запроса прекращается.
3. Если Oper не корректен, то возникает ошибка 114 – неверная операция окраски примера. Управление передается обработчику ошибок. Запрос завершается со значением ложь.
4. Новый цвет примера := (Старый цвет примера And ColorMask) Oper NewColor

3.3.5.3 Ошибки компонента задачника

В табл. 4 приведен полный список ошибок, которые могут возникать при выполнении запросов компонентом задачник, и действия стандартного обработчика ошибок.

Таблица 4.

Ошибки компонента задачник и действия стандартного обработчика ошибок.

№	Название ошибки	Стандартная обработка
101	Запрос при отсутствии задачника	Занесение номера в Error
102	Ошибка чтения задачника	Занесение номера в Error
103	Ошибка записи задачника	Занесение номера в Error
104	Попытка считывания задачника при открытых сеансах ранее считанного задачника	Занесение номера в Error
105	Закрытие задачника при открытых сеансах	Занесение номера в Error
106	Недопустимый код операции при открытии сеанса	Занесение номера в Error
107	Неверный номер сеанса	Занесение номера в Error
108	переход за конечную границу текущей выборки	Игнорируется
109	Переход за начальную границу текущей выборки	Игнорируется
110	Неверный тип вектора в запросе Get	Занесение номера в Error
111	Попытка чтения до или после текущей выборки	Занесение номера в Error
112	Данные отсутствуют	Игнорируется
113	Неверный тип вектора в запросе Put	Занесение номера в Error
114	Неверная операция окраски примера	Занесение номера в Error