Министерство образования и науки Российской Федерации

Федеральное агентство по образованию

Государственное образовательное учреждение
высшего профессионального образования
«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

М. Э. Абрамян

Конструктор учебных заданий для электронного задачника Programming Taskbook

Методическая разработка
для преподавателей программирования

Ростов-на-Дону
2009

Печатается по решению
учебно-методической комиссии
факультета математики, механики и компьютерных наук ЮФУ
от 27 апреля 2009 г. (протокол № 8)

Р е ц е н з е н т ы :
к. ф.-м. н., доцент С. С. Михалкович,
ст. преп. Л. А. Мачулина

Аннотация

В методической разработке описываются средства электронного задачника Programming Taskbook, позволяющие создавать новые группы учебных заданий. Ее основная часть посвящена подробному описанию компонентов конструктора учебных заданий PT4TaskMaker, реализованного в виде модуля языка Паскаль. Приводится ряд примеров использования данного конструктора для разработки заданий по различным темам базового курса программирования.

Дополнительно описываются новые возможности программы «Конструктор вариантов», обеспечивающие быструю перекомпоновку имеющихся заданий, а также тестирование сводных групп, полученных в результате подобной перекомпоновки.

Автор: М. Э. Абрамян.

© М. Э. Абрамян, 2009

Предисловие

Электронный задачник Programming Taskbook в настоящее время активно используется на факультете математики, механики и компьютерных наук и других факультетах ЮФУ при проведении практических занятий по программированию. Опубликован ряд пособий, базирующихся на данном задачнике и ориентированных на различные языки программирования [1–4, 6–8]. Имеется методическая разработка для преподавателей [5], в которой описываются дополнительные средства задачника, входящие в комплекс Teacher Pack for Programming Taskbook.

Настоящая методическая разработка, которую можно рассматривать как продолжение работы [5], посвящена новым средствам электронного задачника Programming Taskbook, появившимся в версии 4.8. Эти средства позволяют преподавателю дополнять задачник новыми наборами учебных заданий и, таким образом, делают задачник «расширяемым».

Основным инструментом, предназначенным для разработки новых групп заданий, является конструктор заданий PT4TaskMaker, реализованный в виде модуля на языке Pascal и доступный для использования в средах программирования Borland Delphi (начиная с версии 4.0) и Free Pascal Lazarus 0.9. Следует подчеркнуть, что разработанные с помощью данного конструктора группы заданий будут доступны для выполнения на любых языках и в любых программных средах, поддерживаемых задачником. В настоящем пособии дается общий обзор возможностей конструктора PT4TaskMaker (п. 1) и подробно описываются его компоненты (п. 2–3). В отдельном разделе рассматриваются вопросы форматирования текстов заданий (см. п. 4); в нем дается описание имеющихся в конструкторе управляющих последовательностей. Наконец, в завершающем разделе, посвященном конструктору PT4TaskMaker, приводятся многочисленные примеры его использования для разработки заданий по различным темам курса программирования (см. п. 5).

Дополнением к конструктору PT4TaskMaker могут служить новые средства, включенные в программу PTVarMaker «Конструктор вариантов» (подробное описание данной программы содержится в [5]). Эти новые средства позволяют разрабатывать *сводные группы*, то есть группы, не содержащие реализации новых заданий, а лишь импортирующие наборы заданий из базовых и дополнительных групп и компонующие их в порядке, который требуется преподавателю. Конструктор вариантов предоставляет упрощенный способ создания сводных групп, не требующий написания программного кода. Этот способ и связанные с ним новые команды, появившиеся в конструкторе вариантов, подробно описываются в заключительном разделе методических указаний (см. п. 6).

Конструктор заданий PT4TaskMaker и конструктор вариантов PTVarMaker входят в комплекс «Teacher Pack for Programming Taskbook 4», являющийся свободно распространяемым дополнением к универсальному варианту электронного задачника Programming Taskbook 4. Данный комплекс включает также гипертекстовую справочную систему «Teacher Pack Info», содержащую описание всех его компонентов.

Подробную информацию о задачнике Programming Taskbook можно получить на его сайте `http://ptaskbook.com/`.

1. Конструктор учебных заданий: общее описание

1.1. Назначение и состав конструктора учебных заданий

Конструктор учебных заданий PT4TaskMaker позволяет разрабатывать новые группы заданий для электронного задачника Programming Taskbook, начиная с версии 4.8. Каждая группа должна оформляться в виде динамической библиотеки (dll-файла). Dll-файлы с новыми группами могут находиться либо в рабочем каталоге учащегося, либо в подкаталоге Lib системного каталога задачника. Подключение новых групп происходит автоматически при инициализации задания, поэтому вид проекта-заготовки при работе с новыми группами не отличается от стандартного вида проекта, ориентированного на базовые группы заданий. Кроме того, новые группы автоматически добавляются в список доступных групп в программных модулях PT4Demo и PT4Load. Созданные в виде dll-файлов новые группы заданий могут использоваться не только в любых средах программирования, поддерживаемых универсальным вариантом задачника, но и в среде PascalABC.NET; для этого достаточно разместить dll-файл в подкаталоге PT4\Lib системного каталога PascalABC.NET или в рабочем каталоге учащегося.

Поскольку конструктор учебных заданий предназначен, прежде всего, для преподавателей программирования, связанные с ним файлы включены не в базовый дистрибутив задачника Programming Taskbook, а в дистрибутив комплекса Teacher Pack for PT4. При установке данного комплекса в его системном каталоге (по умолчанию это каталог PT4TeacherPack, расположенный в каталоге Program Files) создается подкаталог TASKMAKE, содержащий шесть файлов: PT4TaskMaker.pas, PT4MakerDemo.dpr, PT4MakerDemo.dof, PT4MakerDemo.lpr, PT4MakerDemo.lpi и PT4MakerDemo.dll.

Файл **PT4TaskMaker.pas** содержит исходный текст модуля PT4TaskMaker, включающего все процедуры и функции конструктора учебных заданий. Данный модуль можно использовать как в среде Borland Delphi, начиная с версии 4.0, так и в среде Free Pascal Lazarus.

Файлы **PT4MakerDemo.dpr и PT4MakerDemo.dof** представляют собой компоненты Delphi-проекта для создании dll-библиотеки с демонстрационной группой заданий MakerDemo; dpr-файл содержит исходный текст библиотеки, а dof-файл — настройки проекта, определяющие, в частности, главное приложение (host application) для тестирования библиотеки.

Файлы **PT4MakerDemo.lpr и PT4MakerDemo.lpi** представляют собой компоненты проекта для среды Lazarus, предназначенного для создании dll-библиотеки с той же самой демонстрационной группой заданий MakerDemo; lpr-файл содержит исходный текст библиотеки, а lpi-файл — настройки проекта.

Файл **PT4MakerDemo.dll** получен в результате компиляции Delphi-проекта PT4MakerDemo в среде Turbo Delphi 2006.

Тексты dpr- и lpr-файлов практически совпадают; различаются лишь директивы компилятора, указанные в начале этих файлов, кроме того, раздел exports lpr-файла, в отличие от аналогичного раздела dpr-файла, содержит имя функции activate. Процесс разработки данной демонстрационной группы заданий описан в разделе «Примеры» (см. п. 5).

Подробное описание возможностей конструктора учебных заданий включено в гипертекстовую справочную систему Teacher Pack Info. Данную систему можно вызвать непосредственно из меню комплекса Teacher Pack (расположенного в группе «Программы» Главного меню Windows), выполнив команду «Teacher Pack Info».

Еще один вариант конструктора PT4TaskMaker включен в систему PascalABC.NET. Этот вариант, в отличие от варианта из комплекса Teacher Pack, позволяет создавать группы заданий в виде модулей системы PascalABC.NET (pcu-файлов), которые доступны для использования только в среде PascalABC.NET.

1.2. Обзор элементов модуля PT4TaskMaker

Модуль PT4TaskMaker реализован для языка Pascal (среды Borland Delphi и Free Pascal Lazarus; вариант модуля реализован также для языка PascalABC.NET) и содержит следующие элементы:

- процедурный тип TInitTaskProc; в модуле с группой заданий должна быть определена *основная процедура группы* типа TInitTaskProc, позволяющая генерировать задание с требуемым номером (см. п. 2.1);
- процедура CreateGroup, определяющая общие характеристики группы заданий (см. п. 2.1);
- процедуры и константы, используемые для создания нового задания и добавления в него формулировки, а также исходных и контрольных данных базовых типов boolean, integer, real, char, string; обычно каждое задание оформляется в виде отдельной процедуры, которая вызывается из основной процедуры группы (см. п. 2.2);
- процедура UseTask, позволяющая импортировать в создаваемую группу задания из других групп; эта процедура обычно вызывается в основной процедуре группы (см. п. 2.3);
- процедуры, обеспечивающие добавление комментариев, оформляемых в виде *преамбулы* к группе и ее подгруппам (см. п. 2.4);
- процедура RegisterGroup, обеспечивающая регистрацию группы, реализованной в виде pcu-модуля системы PascalABC.NET (см. п. 2.5);
- функции и константы, позволяющие определить текущее состояние задачника: используемый язык программирования и текущую *локаль* — русскую или английскую (см. п. 2.6);
- функции, предоставляющие разработчику заданий образцы слов, предложений и многострочных текстов (см. п. 2.7);
- процедуры, позволяющие включать в задание файловые данные (см. п. 3.1);
- процедуры, позволяющие включать в задание указатели и динамические структуры данных — линейные списки и деревья (см. п. 3.2).

1.3. Библиотечные и сводные группы

При разработке новых заданий целесообразно объединять их в группы с именами, содержащими, кроме названия темы, дополнительные сведения, например, версию созданной группы и данные об авторе. Однако допустимые имена групп могут содержать не более 9 символов, что является недостаточным для указания дополнительных сведений. С другой стороны, группы, содержащие только новые задания (*библиотечные группы*), вряд ли будут использоваться непосредственно в учебном

процессе; более целесообразной будет компоновка этих новых заданий с заданиями из соответствующей базовой группы задачника (и, возможно, с заданиями из других библиотечных групп).

Поэтому при разработке группы, содержащей исключительно новые задания (*библиотечной группы*, или *библиотеки заданий*) разрешено указывать имя длины более 9 символов (но не более 25).

Библиотечные группы не предназначены для непосредственного использования в учебном процессе. Задания из этих групп можно запускать с помощью процедуры Task, однако они будут отображаться только в *демонстрационном режиме* (точнее, в режиме «просмотра библиотеки заданий», о чем будет свидетельствовать соответствующий текст, указываемый в окне задачника вместо сведений об учащемся). Библиотечные группы не включаются в список групп, отображаемый в модуле PT4Load. Если библиотечная группа реализована в виде dll-библиотеки, то она доступна для просмотра с помощью модуля PT4Demo (в этом модуле библиотечные группы указываются в конце списка доступных групп).

Для применения на занятиях удобно использовать *сводные группы*, не содержащие реализации новых заданий, а лишь импортирующие наборы заданий из базовых и дополнительных библиотечных групп и компонующие их в порядке, который требуется преподавателю. При импортировании задания из другой группы у него изменяется название и ключ (которые берутся из характеристик сводной группы), однако сохраняются такие характеристики импортируемой группы, как заголовки подгрупп и сведения об авторе. Имена сводных групп должны содержать не более 9 символов. Для разработки сводных групп можно использовать упрощенный механизм, не требующий написания программного кода (см. п. 6): достаточно подготовить текстовый файл, в котором в специальном формате будут содержаться все данные, необходимые для создания сводной группы, а затем обработать этот файл в программе «Конструктор вариантов», входящей в комплекс Teacher Pack.

1.4. Структура проекта с описанием группы заданий

В данном разделе описываются правила, которым должен удовлетворять проект (dll-библиотека или pcu-модуль), содержащий описание новой группы заданий (см. также п. 5).

Каждый проект, реализуемый в виде dll-библиотеки, должен содержать определение *единственной* группы заданий. Имя библиотеки с группой заданий должно иметь вид `PT4`<*имя группы*><*маркер локали*>, где <*маркер локали*> является либо пустой строкой, либо имеет вид `_ru` или `_en`. Например, группа MakerDemo может быть реализована в виде библиотек PT4MakerDemo.dll, PT4MakerDemo_ru.dll и PT4MakerDemo_en.dll. Библиотеки с явно указанным маркером локали используются только в варианте задачника, соответствующем данной локали (ru — в русском варианте задачника, en — в английском). Библиотеки, в которых не указан маркер локали, используются в любом варианте задачника (если они не перекрываются библиотекой с явно указанной локалью). Порядок поиска библиотек для требуемой группы следующий: вначале просматривается *рабочий каталог учащегося* и в нем ищется библиотека с данным именем и явно указанной локалью; если она не найдена, то ищется библиотека с данным именем без маркера локали; если она не найдена, то поиск библиотек (в этом же порядке) выполняется в *каталоге Lib системного каталога задачника*.

При разработке проекта в виде pcu-модуля системы PascalABC.NET перечисленные выше требования не являются обязательными: pcu-модулю можно присваивать произвольное имя, и в одном pcu-модуле можно определять несколько групп заданий. Однако следование этим требованиям делает созданные модули более удобными для использования, поскольку позволяет по их именам определять имена реализованных в них групп.

Проект с описанием новой группы должен иметь определенную структуру. Вначале опишем структуру проекта, реализованного в виде dll-библиотеки:

```
library PT4MakerDemo;

uses PT4TaskMaker;

//процедуры, реализующие конкретные задания
...

procedure InitTask(num: integer); stdcall;
begin
  // в данной процедуре выполняются вызовы вспомогательных процедур,
  // реализующих все задания группы; номер задания определяется
  // параметром num; для определения процедуры, соответствующей
  // требуемому номеру, обычно используется оператор case
  ...
end;

procedure inittaskgroup;
begin
  // процедура inittaskgroup должна быть оформлена как экспортируемая
  // процедура данной библиотеки; в ней выполняется вызов стартовой
  // процедуры CreateGroup и могут вызываться процедуры, связанные
  // с добавлением комментариев (преамбул) для группы и ее подгрупп
  CreateGroup('MakerDemo', 'Примеры различных задач',
    'М. Э. Абрамян, 2009', 'qwqfsdf13dfttd', 8, InitTask);
  ...
end;

exports inittaskgroup, activate;

begin
end.
```

В проектах, реализуемых в среде Delphi, в разделе exports можно не указывать процедуру activate.

Структура проекта, реализованного в виде pcu-модуля системы PascalABC.NET, имеет некоторые отличия:

```
unit PT4MakerDemo;

uses PT4TaskMaker;

//процедуры, реализующие конкретные задания
...

procedure InitTask(num: integer);
```

```
begin
  ...
end;

procedure inittaskgroup;
begin
  CreateGroup('MakerDemo', 'Примеры различных задач',
    'М. Э. Абрамян, 2009', 'qwqfsdf13dfttd', 8, InitTask);
  ...
  RegisterGroup('PT4MakerDemo');
end;

begin
  inittaskgroup;
end.
```

Обратите внимание на отсутствие модификатора stdcall после заголовка процедуры InitTask, на вызов процедуры RegisterGroup в конце процедуры inittaskgroup, а также на то обстоятельство, что вместо экспортирования процедуры inittaskgroup выполняется ее вызов в секции инициализации модуля.

В группы, реализованные в виде dll-библиотек, нельзя импортировать задания и тексты преамбул из групп, реализованных в виде pcu-модулей. В группы, реализованные в виде pcu-модулей, можно импортировать задания и тексты преамбул как групп, реализованных в виде pcu-модулей, так и групп, реализованных в виде dll-библиотек. Однако при импортировании данных из других pcu-модулей необходимо указать имена этих модулей в списке uses.

При определении новой группы заданий можно учитывать *текущий язык программирования*, установленный для задачника, и в зависимости от этого языка поразному инициализировать некоторые задания группы. Эта возможность является особенно полезной при реализации заданий, связанных с обработкой динамических структур данных, поскольку в языке Visual Basic подобные задания должны быть недоступны, а в языках Pascal и C++ они должны оформляться по-другому, нежели в языках VB.NET и C# (см. также п. 5.7).

Разрабатываемые группы заданий желательно снабжать *дополнительными комментариями* (см. п. 5.3). Эти комментарии не отображаются в окне задачника, однако включаются в html-описание группы в виде *преамбулы группы* (html-описание группы можно создать, либо вызвав в программе процедуру Task с параметром вида '<*имя группы*>#', например, `'MakerDemo#'`, либо воспользовавшись кнопкой  в окне программного модуля PT4Demo). Кроме того, большие группы заданий целесообразно разделять на несколько подгрупп, в каждую из которых также можно добавлять комментарии (*преамбулы подгрупп*). В преамбулах, как и в формулировках заданий, можно использовать специальные *управляющие последовательности* (см. п. 4), которые позволяют отформатировать текст требуемым образом (в частности, обеспечивают выделение переменных, позволяют использовать в тексте верхние и нижние индексы, специальные символы и т. д.). Форматирование используется и при отображении формулировки задания в окне задачника, и при генерации html-страницы с описанием группы заданий.

2. Основные компоненты конструктора заданий

2.1. Определение общих характеристик группы заданий

При создании новой группы заданий требуется определить следующие характеристики этой группы:

- *имя группы* (GroupName) — текстовая строка, содержащая от 1 до 25 символов — цифр и латинских букв, причем последний символ не может быть цифрой (если имя группы содержит более 9 символов, то она считается особой *библиотечной группой,* работа с которой отличается от работы с обычной группой — см. п. 1.3); запрещается использовать имена стандартных групп задачника (Begin, Integer и т. д.); имена, различающиеся только регистром букв, считаются совпадающими;
- *описание группы* (GroupDescription) — непустая текстовая строка с кратким описанием данной группы; при генерации полного описания группы в виде html-страницы данная строка указывается в качестве *заголовка* этого описания;
- *сведения об авторе* (GroupAuthor) — текстовая строка с информацией о разработчике данной группы заданий (фамилия, инициалы, год разработки, e-mail и т. п.; строка может быть пустой);
- *ключ группы* (GroupKey) — непустая текстовая строка с произвольным набором символов, позволяющая в дальнейшем идентифицировать в файле результатов results.dat и results.abc те выполненные задания, которые относятся к данной группе;
- *количество заданий в группе* (TaskCount) — целое число в диапазоне от 1 до 999, определяющее количество заданий в группе;
- *основная процедура группы заданий* (InitTaskProc) — процедура с одним параметром типа integer, обеспечивающая инициализацию всех заданий данной группы (параметр данной процедуры определяет номер задания в пределах группы).

Из перечисленных характеристик в дополнительном комментарии нуждается *ключ группы*. Если не использовать подобную характеристику, то становится невозможной идентификация группы, к которой относятся задания, выполненные учащимся. Действительно, имя задания, сохраненное в файле результатов, не позволяет однозначно его идентифицировать, поскольку ничто не мешает разработать другую группу с тем же именем и совершенно другими заданиями, после чего «подменить» ею исходную группу. Проблему решает использование ключа группы, который сложно подделать, так как он известен только разработчику группы. При успешном выполнении задания в файл результатов дополнительно записывается *идентификатор группы*, вычисляемый на основе ее ключа и позволяющий однозначно определить группу, к которой относится выполненное задание. Поскольку информация, связанная с идентификаторами групп, представляет интерес только для преподавателя, ознакомиться с ней можно только с помощью программы «Контрольный центр преподавателя», описанной в [5] (см. ее команду «Check-файлы | Просмотреть файл check.inf»).

Примечание. При выводе краткого описания группы в программных модулях PT4Demo и PT4Load первый символ этого описания преобразуется к нижнему регистру (поскольку текст описания располагается в этих модулях после двоеточия). Если

понижать регистр первого символа не следует (например, в случае, если этот символ является началом фамилии), то в начале в начале краткого описания группы надо указать дополнительный символ-метку «^» (шапочка).

Для определения характеристик новой группы необходимо вызвать процедуру CreateGroup, указав эти характеристики в качестве параметров:

```
procedure CreateGroup(GroupName, GroupDescription,
  GroupAuthor, GroupKey: string; TaskCount: integer;
  InitTaskProc: TInitTaskProc);
```

Тип TInitTaskProc является *процедурным типом* и определяется следующим образом:

```
type TInitTaskProc = procedure(n: integer); stdcall;
```

Процедуру CreateGroup необходимо вызывать в процедуре inittaskgroup, которая должна экспортироваться библиотекой, содержащей данную группу. При реализации группы в виде pcu-файла системы PascalABC.NET процедура inittaskgroup должна вызываться в секции инициализации модуля, содержащего определение новой группы заданий. Основная процедура группы (типа TInitTaskProc) при ее описании в библиотеке должна иметь модификатор stdcall (в модуле PascalABC.NET этот модификатор не требуется).

Процедура CreateGroup контролирует правильность переданных ей параметров и в случае ошибки выводит на экран информационное окно с ее описанием. В подобной ситуации все последующие действия, связанные с определением данной группы, игнорируются, и группа не включается в список доступных для использования групп заданий. Перечислим некоторые из возможных ошибок:

- в процедуре inittaskgroup определяется более одной группы заданий (в этом случае определения всех групп, кроме первой, игнорируются); при реализации групп в виде pcu-файла данное ограничение отсутствует;
- имя группы не соответствует имени dll-файла, в котором данная группа определяется (напомним, что имя dll-файла должно иметь вид `PT4`*<имя группы>* или `PT4`*<имя группы><маркер локали>* — см. п. 1.4); при реализации группы в виде pcu-файла данное ограничение отсутствует;
- к задачнику Programming Taskbook уже подключена группа с указанным именем;
- имя группы не является допустимым (в частности, совпадает с именем одной из базовых групп задачника);
- не указано краткое описание группы;
- не указан ключ группы;
- количество заданий не принадлежит диапазону 1–999;
- процедурная переменная InitTaskProc равна nil.

2.2. Базовые константы и процедуры для создания новых заданий

```
const
  xCenter = 0;
  xLeft = 100;
  xRight = 200;
```

Эти константы отвечают за выравнивание данных по горизонтали: константа xCenter центрирует текст, связанный с элементом данных, относительно всей экранной строки, константы xLeft и xRight центрируют текст в пределах левой и правой половины экранной строки соответственно. Используются в качестве параметра X в процедурах групп Data и Result, а также в процедуре TaskText..

```
procedure CreateTask(SubgroupName: string);
procedure CreateTask;
```

Данная процедура должна быть вызвана первой при инициализации нового задания; в качестве параметра SubgroupName указывается заголовок *подгруппы*, в которую включается задание (задания целесообразно разбивать на подгруппы, если их количество в группе является достаточно большим; в случае деления группы на подгруппы *каждое* задание рекомендуется связывать с какой-либо подгруппой). Если параметр является пустой строкой или отсутствует, то задание не связывается с какой-либо подгруппой. В окне задачника заголовок подгруппы выводится над именем задания; если подгруппа для данного задания не указана, то выводится краткое описание всей группы (определенное в параметре GroupDescription процедуры CreateGroup). При выводе краткого описания группы или заголовка подгруппы в окне задачника его текст преобразуется к верхнему регистру.

```
procedure TaskText(S: string; X, Y: integer);
```

Данная процедура добавляет к *формулировке задания* строку S, которая располагается в строке Y (от 1 до 5) раздела формулировки задания, начиная с позиции X (при указании параметра X следует учитывать, что ширина раздела формулировок (как и разделов исходных и результирующих данных) равна 78 символам. Кроме явного указания значения позиции X можно использовать специальные константы xCenter, xLeft и xRight; в частности, если параметр X равен 0, то строка центрируется. Рекомендуется всегда центрировать строки в формулировках заданий (как это делается в базовых группах, входящих в задачник); явное указание позиции X следует использовать лишь при выводе многострочных формул и в других случаях специального выравнивания текста. Все строки должны добавляться к формулировке *последовательно*; при этом если формулировка содержит 1 строку, то ее следует располагать на экранной строке с номером 3, если 2 строки — на экранных строках 2 и 4, если 3 строки — на экранных строках 2, 3 и 4, если 4 строки — на экранных строках с номерами от 2 до 5 (именно так оформляются задания в базовых группах задачника). Нарушение порядка добавления строк не проявится при отображении формулировки в окне задачника, однако приведет к неверному выводу формулировки в html-описании группы.

Кроме пяти строк с основным текстом формулировки, который отображается на экране при выводе задания, можно указывать *дополнительные строки*, отображаемые на экране при прокрутке текста задания (связанные с прокруткой кнопки отображаются в окне задачника справа от раздела формулировок, если в формулировке текущего задания имеются дополнительные строки). Все дополнительные строки, как и основные, должны добавляться к формулировке последовательно, причем параметр Y для таких строк надо положить равным 0. Максимальное количество дополнительных строк равно 200.

В строке S можно использовать *управляющие последовательности* (см. п. 4).

Если при выводе строки S часть ее не умещается на экранной строке, то выводится сообщение об ошибке «Ошибочное позиционирование по горизонтали». Если

ошибка произошла при выводе основной строки, то лишняя часть строки S отображается на следующей строке (или на первой строке, если ошибочной является пятая строка в разделе формулировок).

Определенный с помощью процедур TaskText текст формулировки задания используется также при формировании html-описания группы. В этом случае деление на строки, указанное для экранного вывода, игнорируется, однако учитываются дополнительные управляющие последовательности, позволяющие разбивать текст на абзацы с различным способом выравнивания (на отображение текста в окне задачника эти дополнительные последовательности не влияют).

Если при определении задания не указана его формулировка, то выводится сообщение об ошибке.

```
procedure DataB(Cmt: string; B: boolean; X, Y: integer);
procedure DataN(Cmt: string; N: Integer; X, Y, W: integer);
procedure DataN2(Cmt: string; N1, N2: Integer; X, Y, W:
  integer);
procedure DataN3(Cmt: string; N1, N2, N3: Integer; X, Y, W:
  integer);
procedure DataR(Cmt: string; R: real; X, Y, W: integer);
procedure DataR2(Cmt: string; R1, R2: real; X, Y, W: integer);
procedure DataR3(Cmt: string; R1, R2, R3: real; X, Y, W:
  integer);
procedure DataC(Cmt: string; C: char; X, Y: integer);
procedure DataS(Cmt: string; S: string; X, Y: integer);
```

Процедуры группы Data добавляют к заданию *элементы исходных данных*. Добавленные элементы, вместе со строкой-комментарием Cmt, отображаются в разделе исходных данных, начиная с позиции X строки Y (для Y допускаются значения от 1 до 5; ширина экранной строки равна 78 позициям). Как и для процедуры TaskText, параметр X может принимать три особых значения: 0 (центрирование по горизонтали относительно всей экранной строки), 100 (центрирование по горизонтали относительно левой половины экранной строки), 200 (центрирование по горизонтали относительно правой половины экранной строки). Эти значения можно также задавать с помощью констант xCenter, xLeft и xRight.

Параметр W определяет *ширину поля вывода* для числовых данных (выравнивание всегда производится по правому краю поля вывода). Если ширины поля вывода недостаточно, то значение параметра W игнорируется, и для вывода элемента используется минимально необходимое число экранных позиций. При определении ширины поля вывода для вещественного числа следует учитывать размер отображаемой дробной части (который определяется процедурой SetPrecision, описываемой далее).

Для нечисловых данных ширина поля вывода полагается равной фактической ширине данных; в частности, для данных символьного типа отводятся 3 позиции, содержащие начальный апостроф, собственно символ и конечный апостроф, а для логического типа отводятся 5 позиций, достаточных для вывода названий обеих логических констант в любом используемом языке программирования. Для строки отводятся $L + 2$ позиции, где L — длина строки (начальная и конечная позиции используются для вывода апострофов). В зависимости от текущего языка программирования используются либо одинарные, либо двойные апострофы.

Как и в случае процедуры TaskText, при выходе текста за пределы экранной строки выводится сообщение об ошибке.

Используя процедуры группы Data, в задание можно включить до 200 различных скалярных исходных данных (при этом следует учитывать, что некоторые процедуры, например, DataN2 и DataN3, добавляют в набор исходных данных несколько элементов). Наложение различных элементов в разделе исходных данных задачником не контролируется, поэтому при размещении данных следует обращать особое внимание на то, чтобы последующие элементы не скрывали предыдущие. Кроме того, важен порядок определения исходных данных, так как именно в этом порядке данные будут передаваться программе учащегося, выполняющей это задание. Следует придерживаться стандартных правил, принятых в базовых группах задачника: данные должны перебираться по строкам (в направлении сверху вниз), а в пределах каждой строки — слева направо.

В параметре Cmt, содержащем текст комментария к определяемому элементу исходных данных, можно использовать управляющие последовательности (например, для отображения индексов).

В любом задании должен быть задан хотя бы один элемент исходных данных; в противном случае выводится сообщение об ошибке.

```
procedure DataComment(Cmt: string; X, Y: integer);
```

Процедура позволяет добавлять в раздел исходных данных комментарий Cmt, не связанный с каким-либо элементом исходных данных. Этот комментарий использует ресурсы, выделяемые для хранения исходных данных, поэтому общее число подобных комментариев, вместе с общим числом исходных данных, не должно превосходить 200.

```
procedure ResultB(Cmt: string; B: boolean; X, Y: integer);
procedure ResultN(Cmt: string; N: Integer; X, Y, W: integer);
procedure ResultN2(Cmt: string; N1, N2: Integer; X, Y, W:
  integer);
procedure ResultN3(Cmt: string; N1, N2, N3: Integer; X, Y, W:
  integer);
procedure ResultR(Cmt: string; R: real; X, Y, W: integer);
procedure ResultR2(Cmt: string; R1, R2: real; X, Y, W:
  integer);
procedure ResultR3(Cmt: string; R1, R2, R3: real; X, Y, W:
  integer);
procedure ResultC(Cmt: string; C: char; X, Y: integer);
procedure ResultS(Cmt: string; S: string; X, Y: integer);
```

Процедуры данной группы добавляют к заданию *элементы результирующих данных* вместе с их контрольными значениями. Комментарии Cmt к результирующим данным сразу отображаются в разделе результатов. Контрольные значения отображаются в разделе «Пример верного решения». Смысл параметров X, Y и W — тот же, что и для процедур группы Data. Задание может содержать до 200 скалярных элементов результирующих данных.

Как и в случае исходных данных, если элемент контрольных данных не умещается в поле, выделенном для его отображения (шириной W позиций), то параметр W игнорируется, и для вывода используется минимально необходимое число экранных позиций. Однако если элемент *результирующих* данных, переданный в задачник программой, решающей задание, не «уложится» в размер, выделенный для соответствующего элемента контрольных данных, то в правой позиции поля вывода для этого элемента отобразится символ «*» (звездочка) красного цвета. Подобная ситуация

возможна как для чисел, так и для строк (если программа учащегося выведет число или строку, размер которых больше требуемого). Для того чтобы в этой ситуации увидеть полный текст всех подобных элементов результирующих данных, следует переместить курсор мыши в раздел результатов в окне задачника; через 1–2 секунды полный текст всех данных, размер которых превышает допустимый, появится во всплывающей подсказке.

Порядок вызова процедур группы Result важен, так как он соответствует порядку, в котором результирующие данные, полученные программой учащегося, должны передаваться задачнику для проверки их правильности. Поэтому, как и для исходных данных, для набора результирующих данных должен соблюдаться стандартный порядок их размещения: сверху вниз по строкам и слева направо в каждой строке.

В параметре Cmt, содержащем текст комментария к определяемому элементу результирующих данных, можно использовать управляющие последовательности.

Проверка правильности результатов, полученных программой учащегося, выполняется путем сравнения текста, изображающего эти результаты в окне задачника, с текстом, изображающим соответствующие контрольные данные. Это означает, в частности, что вычислительная погрешность, возникающая при обработке вещественных чисел, не будет влиять на проверку правильности, если при отображении этих чисел не используется слишком большое количество дробных знаков (напомним, что число дробных знаков можно задать с помощью процедуры SetPrecision).

В любом задании должен быть задан хотя бы один элемент результирующих данных; в противном случае выводится сообщение об ошибке.

```
procedure ResultComment(Cmt: string; X, Y: integer);
```

Процедура позволяет добавлять в раздел результатов комментарий Cmt, не связанный с каким-либо элементом результирующих данных. Этот комментарий использует ресурсы, выделяемые для хранения результирующих данных, поэтому общее число подобных комментариев, вместе с общим числом результирующих данных, не должно превосходить 200.

```
procedure SetPrecision(N: integer);
```

Процедура устанавливает количество N дробных знаков, используемое в дальнейшем при выводе всех элементов данных вещественного типа. По умолчанию количество дробных знаков равно 2. Если оно равно 0, то вещественные данные отображаются в экспоненциальном формате, а количество дробных знаков определяется шириной поля вывода, указанной для данного числа. Действие текущей настройки, определенной процедурой SetPrecision, продолжается до очередного вызова этой процедуры, однако не распространяется на другие учебные задания текущей группы. Заметим, что при отображении вещественных чисел в качестве десятичного разделителя всегда используется *точка*.

```
procedure SetRequiredDataCount(N: integer);
```

Процедура определяет минимально необходимое количество N элементов исходных данных, требуемое для правильного решения задания при текущем наборе исходных данных. По умолчанию это количество равно общему числу всех указанных в задании исходных данных. Если параметр N имеет нулевое или отрицательное значение, то выводится сообщение об ошибке; если значение параметра превышает общее число элементов исходных данных, то сообщение об ошибке не выводится, а требуемое количество исходных данных полагается равным их общему количеству.

Примером задания, в котором необходимо использовать данную процедуру, может служить задание Series10. В этом задании дается набор из N целых чисел и требуется вывести True, если данный набор содержит положительные числа, и False в противном случае. Ясно, что если при считывании элементов набора будет обнаружено положительное число, то можно сразу выводить значение True и завершать выполнение задания. Однако если при подготовке задания не указать минимально необходимое число исходных данных с помощью процедуры SetRequiredDataCount, то по умолчанию будет считаться, что для решения необходимо прочесть *все* исходные данные, и приведенный выше правильный вариант решения будет расценен как ошибочный (при этом будет выведено сообщение «Введены не все требуемые исходные данные»).

Если заданное с помощью процедуры SetRequiredDataCount количество требуемых исходных данных меньше их общего количества, то программа учащегося не обязана считывать все исходные данные: достаточно прочесть только требуемые. Однако если программа прочтет все данные и выведет правильный ответ, это также будет считаться верным вариантом решения.

```
procedure SetTestCount(N: integer);
```

Процедура определяет количество N успешных тестовых испытаний программы учащегося, необходимое для того, чтобы задание было зачтено как выполненное. По умолчанию количество тестовых испытаний полагается равным 5. Значение N должно находиться в пределах от 2 до 9; при указании других вариантов параметра N выводится сообщение об ошибке.

После каждого успешного тестового испытания в окне задачника выводится сообщение (на зеленом фоне), в котором указывается номер испытания и общее число тестов, необходимых для выполнения данного задания, например: «Верное решение. Тест номер 2 (из 5)». Если при очередном тестовом испытании программы ею будет получено ошибочное решение, то счетчик успешных тестов будет сброшен в 0, и тестирование (после исправления обнаруженной ошибки) придется начинать заново.

```
function Center(I, N, W, B: integer): integer;
```

Вспомогательная функция, которая позволяет размещать по центру экранной строки набор из N элементов данных одинаковой ширины. Эта функция возвращает горизонтальную координату, начиная с которой следует выводить I-й элемент набора (I меняется от 1 до N) при условии, что ширина каждого элемента равна W позициям, а между элементами надо указывать B пробелов. Функция Center обычно используется в качестве параметра X в процедурах групп Data и Result при выводе однотипных наборов данных (в частности, элементов массива).

В качестве примера приведем фрагмент, обеспечивающий формирование и вывод в разделе исходных данных массива вещественных чисел:

```
n := 2 + Random(9);
DataN('N = ', n, 0, 2, 1);
for i := 1 to n do
begin
  a[i] := 9.99 * Random;
  DataR('', a[i], Center(i, n, 4, 2), 4, 4);
end;
```

Вначале (во второй строке области исходных данных) выводится размер N массива, определяемый с помощью датчика случайных чисел и принимающий значения в

диапазоне от 2 до 10 (он снабжается комментарием «N =»). Затем (в четвертой строке) выводятся сами элементы массива, причем благодаря использованию функции Center весь список выравнивается относительно центра экранной строки независимо от количества элементов. Целые части всех элементов лежат в диапазоне от 0 до 9, то есть представляются одной цифрой, одна позиция отводится под отображение десятичного разделителя-точки и по умолчанию указываются два дробных знака, поэтому для каждого элемента следует выделить 4 экранных позиции; это число указывается дважды: как второй параметр функции Center и как последний параметр процедуры DataR. Промежуток между элементами полагается равным 2 экранным позициям (это последний, четвертый параметр функции Center).

Обратите внимание на то, что при использовании функции Center строку комментария следует оставлять пустой.

2.3. Импортирование существующих заданий в новую группу

```
procedure UseTask(GroupName: string; TaskNumber: integer);
```

Данная процедура позволяет *импортировать* в создаваемую группу задание с номером TaskNumber из группы GroupName. Она обычно вызывается непосредственно в основной процедуре группы. Если импортируемое задание не найдено, то при попытке его запуска в окне задачника выводится сообщение «Задание не реализовано для текущего языка программирования», и этот же текст, выделенный курсивом, указывается в html-описании группы после имени, которое должно быть связано с импортированным заданием.

При использовании мини-варианта задачника импортированные задания будут доступны для выполнения только в том случае, если они доступны для выполнения в базовых группах.

В параметре GroupName после имени группы можно дополнительно указывать *поправку для вычисления ссылки на другое задание* (поправка является целым числом и отделяется от имени группы символом #). Например, если в группу Demo в качестве задания Demo10 импортируется задание Proc46, а в качестве Demo11 — задание Proc49, ссылающееся на Proc46, то при импортировании задания Proc49 необходимо указать поправку, равную 2. Если этого не сделать, то в формулировке задания Demo11 будет указана ссылка не на задание Demo10, а на задание Demo8 (поскольку оно находится «на том же расстоянии» от задания Demo11, что и задание Proc46 относительно задания Proc49). Добавление поправки 2 должно быть оформлено следующим образом: `UseTask('Proc#2',49)`.

Примечание. Если новая группа заданий разрабатывается в среде PascalABC.NET, а группа, из которой импортируются задания, также была разработана в PascalABC.NET и, следовательно, содержится в некотором pcu-модуле, то в список uses модуля с новой группой необходимо добавить имя модуля, содержащего группу, из которой импортируются задания.

2.4. Документирование группы заданий

Группы заданий можно снабжать комментариями, делая их «самодокументируемыми». Комментарии можно добавлять не только к группе, но и к ее *подгруппам*, то есть наборам подряд идущих заданий в пределах группы (для включения задания в

определенную подгруппу необходимо указать заголовок этой подгруппы в качестве параметра процедуры CreateTask — см. п. 2.2).

Комментарии не отображаются в окне задачника, но включаются в html-описание группы. Они располагаются между заголовком группы (подгруппы) и формулировками заданий. Таким образом, эти комментарии представляют собой *преамбулы* к группе или ее подгруппам.

Определять преамбулу к подгруппе имеет смысл только в случае, если с этой подгруппой связаны некоторые задания, входящие в определяемую группу. Если группа не содержит заданий, связанных с некоторой подгруппой, то преамбула этой подгруппы в html-описании не выводится.

Для определения преамбул предназначены следующие процедуры.

```
procedure CommentText(S: string);
```

Данная процедура добавляет содержимое строки S к текущей преамбуле, отделяя это содержимое от предыдущего текста преамбулы пробелом. В строке S можно использовать *управляющие последовательности*, обеспечивающие ее форматирование (см. п. 4). Например, для перехода к новому абзацу преамбулы следует использовать последовательность `\P` (управляющие последовательности чувствительны к регистру букв).

```
procedure UseComment(GroupName, SubgroupName: string);
procedure UseComment(GroupName: string);
```

Процедура UseComment добавляет к текущей преамбуле текст преамбулы подгруппы SubgroupName группы GroupName или, если параметр SubgroupName является пустой строкой или отсутствует, текст преамбулы самой группы GroupName. Этот текст отделяется от предыдущего текста преамбулы пробелом. Регистр символов в параметрах GroupName и SubgroupName может быть произвольным.

Если группа с именем GroupName не найдена или в ней отсутствует подгруппа SubgroupName, то процедура не выполняет никаких действий; сообщение об ошибке в этом случае не выводится.

Примечание. Если новая группа заданий разрабатывается в среде PascalABC.NET, а группа, из которой импортируется преамбула, также была разработана в PascalABC.NET и, следовательно, содержится в некотором pcu-модуле, то в список uses модуля с новой группой необходимо добавить имя модуля, содержащего группу, из которой импортируется преамбула.

Процедуры CommentText и UseComment должны вызваться после функции CreateGroup; при этом они определяют преамбулу данной группы. Для того чтобы они определяли преамбулу какой-либо подгруппы данной группы, перед их вызовом необходимо вызвать процедуру Subgroup, описываемую далее.

```
procedure Subgroup(SubgroupName: string);
```

Данная процедура устанавливает режим добавления текста к преамбуле *подгруппы* SubgroupName текущей группы. Этот режим сохраняется до следующего вызова данной процедуры или до завершения определения текущей группы заданий (определение группы, создаваемой в виде dll-файла, завершается при выходе из процедуры inittaskgroup, а группы, создаваемой в виде pcu-файла, — при вызове процедуры RegisterGroup — см. п. 2.5).

Процедуру Subgroup можно вызывать несколько раз для одной и той же подгруппы, при этом ранее определенный текст преамбулы будет дополняться новыми

данными. При вызове процедуры Subgroup с параметром — пустой строкой устанавливается режим дополнения преамбулы *группы* (напомним, что этот режим устанавливается также сразу после вызова процедуры CreateGroup).

2.5. Регистрация новой группы заданий в модуле PT4Load для системы PascalABC.NET

```
procedure RegisterGroup(UnitName: string);
```

Данная процедура предназначена, прежде всего, для использования при создании групп в среде PascalABC.NET (в виде pcu-файлов), однако она доступна и в варианте конструктора, ориентированном на разработку групп в виде dll-файлов. Эта процедура обеспечивает завершение формирования текущей группы; в частности, это означает, что после ее вызова все последующие вызовы процедур, связанных с определением преамбул для группы и ее подгрупп (см. п. 2.4), будут игнорироваться. Впрочем, для группы, создаваемой в виде dll-файла, необходимые завершающие действия выполняются автоматически при выходе из процедуры inittaskgroup, поэтому необходимости в специальном вызове процедуры RegisterGroup не возникает. В то же время, при создании группы в виде pcu-файла системы PascalABC.NET рекомендуется *всегда* вызывать процедуру RegisterGroup после формирования текущей группы, поскольку это предотвращает опасность случайного добавления постороннего текста к последней определяемой преамбуле.

В варианте конструктора для создания pcu-файлов процедура RegisterGroup обеспечивает еще одно важное действие, а именно, она добавляет информацию о созданной группе во вспомогательный программный модуль PT4Load, интегрированный в систему PascalABC.NET. Это позволяет использовать модуль PT4Load для создания программ-заготовок, связанных с данной группой. Следует заметить, что необходимость в специальных действиях по регистрации новой группы в модуле PT4Load возникает только для групп, созданных в виде pcu-файлов, так как группы, определенные в виде dll-файлов и размещенные в текущем каталоге или подкаталоге Lib системного каталога задачника, регистрируются в модуле PT4Load (а также в модуле PT4Demo) автоматически. Напомним, что при использовании варианта задачника, интегрированного в PascalABC.NET, системным каталогом задачника считается подкаталог PT4 системного каталога PascalABC.NET.

Параметр UnitName процедуры должен содержать имя pcu-модуля (при вызове данной процедуры для группы, определяемой в виде dll-файла, параметр может содержать произвольный текст, например, пустую строку). Информация о регистрируемой группе сохраняется в файле PTGroups.dat, который размещается в подкаталоге Lib системного каталога PascalABC.NET, а также в каталоге, из которого запускается программа с заданием. При загрузке модуля PT4Load вначале считывается информация из файла PTGroups.dat, расположенного в текущем каталоге, а затем — из подкаталога Lib системного каталога PascalABC.NET.

При использовании английской версии задачника вместо файла PTGroups.dat используется файл PTGroups_en.dat.

Вызов процедуры RegisterGroup, как и процедуры CreateGroup, производится в секции инициализации pcu-модуля, поэтому добавление (или восстановление) информации о данной группе заданий будет происходить при каждом запуске программы, использующей этот модуль. При этом сразу после выполнения этой программы

данная информация будет доступна для использования в модуле PT4Load (в частности, при отображении на экране окна PT4Load новая группа появится в списке имеющихся групп, и можно будет создать заготовку для любого задания из этой группы). Следует, однако, заметить, что если некоторая группа заданий реализована как в виде dll-файла, так и в виде pcu-файла, то предпочтение будет отдано группе из dll-файла как более универсальной, и именно с ней будет связана программа-заготовка, созданная модулем PT4Load.

Примечание. В модуле PT4Load не отображаются библиотечные группы, то есть группы, имена которых содержат более 9 символов (задания из подобных групп не предназначены для непосредственного выполнения; их можно лишь включать в состав других групп). Для библиотечных групп процедура RegisterGroup не выполняет действий, связанных с их регистрацией.

2.6. Константы и функции для определения текущего состояния задачника

```
const
  lgPascal = $0001;
  lgVB = $0002;
  lgCPP = $0004;
  lgCS = $0100;
  lgVBNET = $0200;
  lgPascalNET = $0400;
  lgAll = $FFFF;
  lgNET = $FF00;
  lgWithPointers = $003D;
  lgPascalABCNET = $0401;
```

Данные константы, совместно с описываемой далее функцией CurrentLanguage, позволяют определить язык программирования, на который в данный момент (то есть в момент инициализации текущей группы заданий) настроен задачник. Константы lgPascal, lgVB, lgCPP, lgCS, lgVBNET соответствуют конкретному языку из числа тех, которые доступны в текущей версии задачника (Pascal, Visual Basic, C++, C#, Visual Basic .NET). Эти константы являются *битовыми флагами*. Некоторые константы являются комбинациями битовых флагов (то есть *битовыми масками*) и позволяют определить, к какой категории относится текущий язык:

- lgAll — любой язык,
- lgWithPointers — язык, поддерживающий работу с указателями (в версии 4.8 задачника это языки Pascal и C++),
- lgNET — язык платформы .NET (в версии 4.8 задачника это языки C# и Visual Basic .NET).

Особое место занимает язык, реализованный в системе PascalABC.NET, поскольку в нем объединяются свойства обычного языка Pascal и языка платформы .NET. Поэтому данному языку соответствует комбинация флагов lgPascal + lgPascalNET; это, в частности, означает, что он принадлежит одновременно к категориям lgWithPointers и lgNET. Для языка PascalABC.NET предусмотрена также именованная константа lgPascalABCNET.

```
function CurrentLanguage: integer;
```

Функция возвращает значение, соответствующее языку программирования, на

который в данный момент настроен задачник. Помимо сравнения возвращаемого значения функции с константами, соответствующими конкретному языку (например, `CurrentLanguage = lgCPP`), можно также использовать данную функцию для определения *категории*, к которой относится текущий язык программирования; в этом случае необходимо применять побитовые операции. Например, для проверки того, что текущий язык программирования относится к категории языков платформы .NET, достаточно проверить истинность условия `CurrentLanguage and lgNET <> 0`.

При использовании задачника совместно с системой PascalABC.NET функция CurrentLanguage всегда возвращает значение `lgPascalABCNET`.

```
function CurrentLocale: string;
```

Функция возвращает строку, соответствующую текущей *локали*, то есть текущему языку интерфейса, используемому в задачнике. В версии 4.8 задачника возможными возвращаемыми значениями функции CurrentLocale являются `'ru'` (русский вариант задачника) и `'en'` (английский вариант).

2.7. Образцы слов и предложений

Приведенные ниже элементы конструктора PT4TaskMaker позволяют получить доступ к встроенным в него образцам текстовых исходных данных: словам (Word), предложениям (Sentence) и многострочным текстам (Text).

```
const
  SampleError = '#ERROR?';
  MaxLineCount = 50;

function WordCount: integer;
function SentenceCount: integer;
function TextCount: integer;
function WordSample(N: integer): string;
function SentenceSample(N: integer): string;
function TextSample(N: integer): string;

function EnWordCount: integer;
function EnSentenceCount: integer;
function EnTextCount: integer;
function EnWordSample(N: integer): string;
function EnSentenceSample(N: integer): string;
function EnTextSample(N: integer): string;
```

Функции WordSample, SentenceSample и TextSample возвращают текстовые данные, соответствующие текущей *локали*, то есть текущему языку интерфейса, используемому в задачнике (см. описание функции CurrentLocale в п. 2.6): для русского варианта задачника возвращаются русские данные, для английского — английские. Варианты этих функций, снабженные префиксом En, возвращают английские текстовые данные в *любом* варианте задачника.

Функции, оканчивающиеся словом Count, возвращают *количество* соответствующих элементов данных. В реализации конструктора для версии задачника 4.8 доступно 116 слов, 61 предложение и 85 текстов как на русском, так и на английском языке.

Функции WordSample/EnWordSample и SentenceSample/EnSentenceSample возвращают соответственно слово или предложение с индексом N (индексирование проводится от 0).

Функция TextSample/EnTextSample возвращает строку, связанную с многострочным текстом, имеющим индекс N (индексирование также проводится от 0). При этом между соседними строками этого текста располагаются символы #13#10 (маркеры конца строки). В конце текста маркер конца строки отсутствует, число строк в тексте не превышает значения константы MaxLineCount. Любой текст состоит из нескольких абзацев; между абзацами текста помещается одна пустая строка, отступы в начале абзацев («красная строка») не используются. В тексте не используются также переносы слов.

Если параметр N является недопустимым, то все функции возвращают особую строку, равную константе SampleError.

Буква «ё» в русских текстовых данных не используется.

Все слова-образцы состоят из заглавных (прописных) букв. Помимо слов «общего вида» в набор слов включены слова, обладающие следующими особенностями (наличие подобных особых слов может оказаться полезным при составлении заданий):

- слова, начинающиеся и оканчивающиеся одной и той же буквой;
- слова, содержащие три одинаковые буквы (в русском наборе — три буквы «А», в английском наборе — три буквы «E»).

Длина предложений-образцов не превосходит 76 символов; таким образом, любое предложение умещается на одной экранной строке (напомним, что строки при выводе в окне задачника обрамляются апострофами).

Многострочные тексты предназначены для использования, прежде всего, в заданиях на обработку текстовых файлов (см. реализацию подобного задания в п. 5.6). Длина строк в многострочных текстах не превосходит 50 символов.

3. Дополнительные компоненты конструктора заданий

3.1. Процедуры для включения в задание файлов

В конструкторе учебных заданий PT4TaskMaker предусмотрена возможность включения в каждое учебное задание (в качестве исходных или результирующих данных) до 10 файлов. Кроме текстовых файлов (описатель text) в заданиях можно использовать типизированные файлы, содержащие элементы типа integer, real, char, ShortString. Каждый файл должен содержать не более 999 элементов (для текстовых файлов элементами считаются файловые строки); в случае, если файл содержит более 999 элементов, элементы с номерами, превышающими 999, в окне задачника не отображаются. Для корректного отображения на экране, а также для правильной проверки результирующих файлов необходимо, чтобы строки в файлах типа file of ShortString и text состояли из не более чем 70 символов.

Все процедуры, связанные с определением файловых данных, следует вызывать после вызова процедуры CreateTask (см. п. 2.2).

Файлы должны создаваться в текущем каталоге, поэтому при задании их имен не следует указывать имя диска и путь. Рекомендуется снабжать имена всех файлов, используемых в заданиях, расширением .tst.

Все данные из файла, как правило, нельзя одновременно отобразить в окне задачника, поэтому для файловых элементов предусмотрена возможность *прокрутки*. Для типизированных файлов прокрутка выполняется в горизонтальном направлении, а для текстовых файлов — в вертикальном.

```
procedure DataFileN(FileName: string; Y, W: integer);
procedure DataFileR(FileName: string; Y, W: integer);
procedure DataFileC(FileName: string; Y, W: integer);
procedure DataFileS(FileName: string; Y, W: integer);
procedure DataFileT(FileName: string; Y1, Y2: integer);
```

Процедуры группы DataFile с именами, завершающимися символами N, R, C, S, T, позволяют включить в задание в качестве исходного файла один файл типа file of integer, file of real, file of char, file of ShortString, text соответственно. К моменту вызова процедуры файл, включаемый в задание, должен быть создан, заполнен исходными данными и закрыт. Имя этого файла передается параметром FileName.

Два последних параметра имеют разный смысл для процедур, обрабатывающих типизированные файлы, и для процедуры DataFileT, обрабатывающей текстовые файлы. Для процедур, связанных с типизированными файлами, параметр Y указывает номер экранной строки в области исходных данных, в которой будут отображаться элементы данного файла, а параметр W указывает количество позиций, отводимых под отображение *одного* элемента файла. Если фактическая длина элемента файла оказывается меньше параметра W, то изображение элемента дополняется пробелами (пробелы добавляются слева для числовых данных и справа для символьных); если длина элемента файла окажется больше значения W, то в конце поля, выделенного для его вывода, будет указан символ «*» (звездочка) красного цвета. При определении параметра W необходимо предусматривать дополнительные позиции для пробелов, служащих разделителями элементов, а в случае строковых и символьных файлов — для апострофов, автоматически добавляемых к каждому элементу при его отображении на экране. Способ отображения вещественных чисел устанавливается, как и для обычных исходных данных, процедурой SetPrecision (см. п. 2.2); по умолчанию вещественные числа отображаются в формате с фиксированной точкой и двумя знаками в дробной части. Количество элементов, отображаемых на экране, определяется автоматически так, чтобы заполнить по возможности всю экранную строку. Никакие другие исходные данные на этой строке размещать нельзя.

Для процедуры DataFileT параметры Y1 и Y2 определяют соответственно номер первой и последней экранной строки той части области исходных данных, которая отводится под отображение текстового файла. На каждой экранной строке размещается одна строка из текстового файла.

Параметры Y, Y1, Y2 должны принимать значения от 1 до 5 (Y и Y1 могут также принимать значение 0; этот случай описан в конце данного раздела); значение Y1 не должно превышать значение Y2. Параметр W должен лежать в диапазоне 1–72.

При наличии нескольких исходных файлов вызов соответствующих процедур группы DataFile может проводиться в любом порядке, независимо от порядка расположения этих файлов на экране. При попытке размещения двух файлов на одной экранной строке выводится сообщение об ошибке. Вызовы процедур группы DataFile могут проводиться как до, так и после вызовов процедур группы Data, определяющих «обычные», не файловые исходные данные (см. п. 2.2).

Вызов процедур группы DataFile не влияет на содержимое включаемых в задание файлов. Он лишь приводит к копированию этого содержимого в специальный буфер в оперативной памяти. Созданная копия используется для отображения содержимого файла на экране; это позволяет просматривать начальное содержимое исходного файла и после его преобразования (или даже удаления) в ходе решения задания.

Поскольку при различных тестовых испытаниях учебного задания желательно не только изменять содержимое исходных файлов, но также и предлагать для обработки файлы с различными именами, возникает опасность «засорения» диска файлами, созданными при предыдущих тестовых испытаниях. Для того чтобы этого не произошло, в задачнике предусмотрено автоматическое удаление при завершении тестового испытания всех файлов, включенных в задание с помощью процедур группы DataFile. Заметим, что это удаление производится и в случае аварийного завершения программы, выполняющей учебное задание. Если же исходный файл был удален самой программой, выполняющей задание, то задачник не будет пытаться удалить этот файл еще раз.

Иногда (хотя и весьма редко — см., например, задание File4) при решении задания не требуется отображать содержимое исходного файла на экране. В этом случае в соответствующей процедуре группы DataFile параметр Y или Y1 надо положить равным 0. Если исходный файл не требуется ни отображать на экране, ни удалять после завершения тестового испытания, то в вызове процедуры группы DataFile нет необходимости.

Следует заметить, что формат *двоичных строковых файлов* является различным для разных языков программирования. Однако «обязанности» по преобразованию исходного строкового файла, имеющего формат языка Pascal, то есть описанного как file of ShortString, в формат текущего языка, установленного для задачника, берет на себя сам задачник, поэтому учитывать эти особенности при разработке заданий на строковые файлы не требуется.

```
procedure ResultFileN(FileName: string; Y, W: integer);
procedure ResultFileR(FileName: string; Y, W: integer);
procedure ResultFileC(FileName: string; Y, W: integer);
procedure ResultFileS(FileName: string; Y, W: integer);
procedure ResultFileT(FileName: string; Y1, Y2: integer);
```

Процедуры группы ResultFile с именами, завершающимися символами N, R, C, S, T, позволяют включить в задание в качестве результирующего файла один файл типа file of integer, file of real, file of char, file of ShortString, text соответственно. К моменту вызова процедуры файл, включаемый в задание, должен быть создан, заполнен контрольными данными и закрыт. Под контрольными данными понимаются, как обычно, данные, которые должны содержаться в результирующем файле в случае правильного решения задания.

Смысл параметров процедур группы ResultFile совпадает со смыслом соответствующих параметров процедур группы DataFile, за исключением того, что теперь номера экранных строк Y, Y1, Y2 относятся к области результирующих данных. Ограничения на параметры для процедур группы ResultFile накладываются те же, что и для процедур группы DataFile.

В результате выполнения процедуры из группы ResultFile содержимое указанного контрольного файла будет скопировано в специальный буфер в оперативной памяти, после чего контрольный файл будет *автоматически удален с диска*. В дальней-

шем, при выполнении задания, файл с таким же именем должен быть создан и заполнен требуемыми данными программой самого учащегося. Контрольные данные, записанные в оперативную память процедурой из группы ResultFile, используются при проверке правильности содержимого результирующего файла (созданного в ходе выполнения задания). Кроме того, эти контрольные данные могут выводиться на экран в качестве примера правильного решения. Имя файла и другие параметры, указанные в процедуре ResultFile, будут также использоваться для поиска и отображения на экране результирующего файла, созданного при выполнении задания. Все результирующие файлы, созданные в ходе решения задания, автоматически удаляются с диска при завершении программы. Подобное удаление производится и при аварийном завершении программы; если же результирующий файл не создан, то попытка его удалить не производится.

Как и в случае процедур группы DataFile, отображение некоторых результирующих файлов можно отключить, положив в соответствующих процедурах ResultFile значения параметров Y или Y1 равными 0 (см., например, задание File1). Это, естественно, не отменит сравнения содержимого результирующих файлов с контрольными данными и удаления результирующих файлов при завершении программы.

3.2. Процедуры для включения в задание указателей и динамических структур данных

В учебные задания можно включать указатели только одного определенного в задачнике типа:

```
type
  PNode = ^TNode;
  TNode = record
    Data : integer;
    Next : PNode;
    Prev : PNode;
    Left: PNode;
    Right: PNode;
    Parent: PNode;
  end;
```

Этот тип позволяет формировать одно- и двусвязные линейные динамические структуры (при этом используются поля связи Next и Prev), бинарные деревья и деревья общего вида (при этом используются поля связи Left и Right), а также бинарные деревья с обратной связью (при этом используются поля связи Left, Right и Parent). Поскольку значение адреса, хранящегося в указателе, не представляет интереса (и, кроме того, может изменяться при каждом тестовом запуске программы), на экране отображается не оно, а условное обозначение указателя «ptr», снабженное обязательным комментарием, например, P_1 = ptr, P_X = ptr и т. д. В задании можно использовать до 36 различных указателей, которым присваиваются номера от 0 до 35. Указатели с номерами от 0 до 9 имеют комментарии с цифровым индексом (P_0–P_9), а указатели с номерами от 10 до 35 — с буквенным (P_A–P_Z). Если некоторый указатель имеет значение nil, то вместо текста «ptr» отображается текст «nil» (или аналогичный текст с обозначением нулевого указателя, соответствующего текущему языку программирования), например, P_1 = nil для языка Pascal, P_1 = NULL для языка C++. Коммен-

тарии к указателям также используются при отображении на экране динамических структур, если они содержат элементы, с которыми связаны данные указатели.

```
procedure SetPointer(NP: integer; P: PNode);
```

Эта процедура позволяет определить в учебном задании указатель с номером NP (значение этого указателя при инициализации задания будет равно P, однако при *выполнении* задания оно может измениться). Все прочие процедуры, связанные с этим указателем и описываемые далее, используют не его конкретное значение, а номер NP.

Номер NP должен лежать в диапазоне от 0 до 35; он указывается в обязательном комментарии к данному указателю (см. выше). Если процедура SetPointer для указателя с номером NP не вызвана, то указатель с этим номером будет иметь значение nil.

```
procedure DataP(Cmt: string; NP: integer; X, Y: integer);
procedure ResultP(Cmt: string; NP: integer; X, Y: integer);
```

Процедуры DataP и ResultP помещают указатель с номером NP в список исходных или, соответственно, результирующих данных учебного задания и отображают его на экране. При отображении на экране указатель снабжается обязательным комментарием вида $P_{\#}$ =, где в качестве символа # указывается символ, связываемый с указателем (для NP от 0 до 9 — соответствующая цифра, для NP от 10 до 35 — заглавная латинская буква от A до Z). Само значение указателя на экран не выводится; вместо него указывается одно из двух условных обозначений: ptr для ненулевого указателя и nil для нулевого указателя. Как и прочие элементы данных, указатель может снабжаться дополнительным комментарием Cmt, который приписывается слева к обязательному (например, вызов процедуры с параметрами Cmt = 'Адрес начала стека: ' и NP = 6 приведет к выводу на экран следующей строки: `Адрес начала стека: P6 = ptr`). В дополнительном комментарии Cmt можно использовать *управляющие последовательности* (см. п. 4). В процедурах DataP и ResultP задается также экранная позиция, начиная с которой элемент данных выводится в соответствующую экранную область. Параметр Y определяет номер строки (от 1 до 5), параметр X — позицию в строке (от 1 до 78; как обычно, требуется, чтобы элемент данных вместе с комментарием полностью умещался на строке).

```
procedure DataList(NP: integer; X, Y: integer);
procedure ResultList(NP: integer; X, Y: integer);
```

Процедуры DataList и ResultList предназначены для помещения структуры типа «одно- или двусвязный линейный динамический список» в набор исходных или, соответственно, результирующих данных, а также для вывода этой структуры на экран. Параметр NP задает номер указателя (предварительно определенный процедурой SetPointer), который указывает на *начало* данного списка, то есть на его первый элемент. Если соответствующий указатель равен nil, то список считается пустым. Пустой список на экране не отображается.

Непустой динамический список отображается на двух экранных строках; первая строка содержит имена указателей, входящих в задание и связанных с данным списком (они задаются процедурой ShowPointer), во второй строке — значения элементов списка (точнее, их полей Data целого типа) и виды связи между элементами.

Если память для элемента результирующего списка должна быть выделена программой учащегося, то значение его поля Data на экране обрамляется точками (например, `.23.`). Если в исходной динамической структуре требуется разрушить один

или несколько элементов, то эти элементы выделяются более бледным цветом, а в случае, если программа учащегося не освободит память, занимаемую этими элементами, они будут выделены красным цветом. Наконец, если в списке, преобразованном программой учащегося, элементы располагаются не на требуемых местах, то они заключаются в скобки (например, `(23)`), а если элемент списка содержит ошибочную ссылку Next, то она помечается двумя красными звездочками (например `46 - **`). Красные звездочки указываются в конце списка также в случае, если его длина превышает максимально допустимую. Специальные обозначения используются также для *циклических списков* (см. пример 3, приведенный далее в этом пункте).

Элемент данных типа «линейный список» должен содержать не более 14 элементов типа TNode, причем значения их полей Data должны лежать в диапазоне от –9 до 99, поскольку для каждого поля Data отводится по *две* экранные позиции. Для большей наглядности рекомендуется использовать числа из диапазона 10–99, резервируя однозначные и отрицательные числа для особых элементов (например, барьерного элемента циклического списка — см. задание Dynamic70). Если значение элемента списка не умещается в поле вывода, то в его последней экранной позиции выводится красная звездочка — признак ошибки.

Для отображения списка как двусвязного необходимо, чтобы в его элементах были определены поля связи Next и Prev; в этом случае связи между соседними элементами списка обозначаются двойными линиями: «=». Если в задании требуется использовать односвязный список, то для его элементов надо определить только поле связи Next, а для полей Prev следует указать значение, *не связанное* с элементами этого списка (например, адрес какой-либо глобальной переменной типа TNode). Связи между элементами односвязных списков обозначаются одинарными линиями: «-».

Вызов процедуры DataList или ResultList приводит к тому, что соответствующий список становится *текущей динамической структурой* для данного задания. Все последующие вызовы процедур ShowPointer, SetNewNode и SetDisposedNode будут влиять на эту текущую структуру.

```
procedure DataBinTree(NP, X, Y1, Y2: integer);
procedure ResultBinTree(NP, X, Y1, Y2: integer);
procedure DataTree(NP, X, Y1, Y2: integer);
procedure ResultTree(NP, X, Y1, Y2: integer);
```

Эти процедуры предназначены для включения в задание *бинарных деревьев* и *деревьев общего вида* (называемых также *деревьями с произвольным ветвлением*) в качестве исходных (DataBinTree и DataTree) или результирующих (ResultBinTree и ResultTree) данных. Для деревьев общего вида используется представление «левая дочерняя вершина — правая сестра» («left child — right sibling»). Как и для процедур DataList и ResultList, описанных выше, первым параметром этих процедур является *номер* указателя типа PNode, ранее определенного с помощью процедуры SetPointer. В данном случае этот указатель должен указывать на *корень* добавляемого в задание дерева; если он равен nil, то дерево считается пустым и не отображается на экране. В отличие от процедур, связанных с линейными списками, для отображения дерева можно (и рекомендуется) выделять на экране *более двух строк*; номера начальной и конечной экранной строки задаются параметрами Y1 и Y2 соответственно. В отличие от других «прокручиваемых» данных (а именно, типизированных и текстовых файлов), дерево может не занимать выделенные для него строки по всей ширине: параметр X показывает, начиная с какой позиции экранных строк будет отображаться де-

рево и связанная с ним информация. Следует отметить, что использовать для других целей можно только *левую часть* строки, связанной с деревом; все позиции строки, начиная с позиции X, будут использоваться для отображения дерева.

Вызов любой из описываемых процедур приводит к тому, что соответствующее дерево становится *текущей динамической структурой* для данного задания. Все последующие вызовы процедур ShowPointer, SetNewNode и SetDisposedNode будут влиять на эту текущую структуру.

При отображении дерева используются обозначения, аналогичные тем, которые применяются при отображении линейных динамических структур. В частности, в качестве вершины изображается значение ее поля Data, причем для вывода этого значения выделяются две экранные позиции (если двух позиций недостаточно, например, в случае значения 234, то на второй из выделенных позиций изображается красная звездочка: `2*`). В качестве обозначения связей между вершинами используются одинарные и двойные линии («-» и «=»); двойные линии, как и для линейных списков, означают, что связь между вершинами является двусторонней (так называемые *деревья с обратной связью* — см. пример 6). Обратная связь обеспечивается полем Parent; ее можно использовать только для бинарных деревьев.

Для деревьев предусмотрены два варианта отображения. Первый вариант предназначен для отображения бинарного дерева; он применяется для деревьев, включенных в задание процедурами DataBinTree и ResultBinTree. В этом варианте обе дочерние вершины располагаются ниже родительской вершины (на следующем уровне — см. примеры 5 и 6). Второй вариант предназначен для отображения *дерева общего вида* (вершины которого могут содержать более двух дочерних вершин); он применяется для деревьев, включенных в задание процедурами DataTree и ResultTree. В этом варианте вершина, определяемая полем Left вершины P, как обычно, располагается ниже и левее вершины P и задает ее первую (левую) дочернюю вершину, а вершина, определяемая полем Right, моделирует следующую вершину-«*сестру*» вершины P и поэтому располагается *на том же уровне*, что и вершина P. Такой способ отображения деревьев позволяет, в частности, легко определить глубину дерева общего вида и номер уровня для любой его вершины (см. пример 7).

Перечислим другие обозначения, имеющие тот же смысл, что и для линейных структур:

- если вершина дерева должна быть создана в программе учащегося, то данная вершина выделяется слева и справа точками, например, `.23.` (для этого используется процедура SetNewNode);
- если в дереве, преобразованном программой учащегося, существующие вершины располагаются не на своих местах, то они заключаются в скобки: `(23)`;
- если в исходном дереве требуется разрушить одну или несколько вершин, то эти вершины выделяются более бледным цветом (а в случае, если программа учащегося не освободит память, занимаемую этими вершинами, они будут выделены красным цветом);
- если переход по ссылке Left или Right для данной вершины дерева невозможен, то, как и в случае линейных структур, это отмечается красными звездочками, которые, однако, изображаются не рядом с данной вершиной, а ниже вершины (что подчеркивает тот факт, что ошибка возникла при попытке перехода на следующий уровень дерева).

Для деревьев, в отличие от линейных структур, нулевые поля связи не отображаются. Если поле Left или Right равно nil, то на изображении дерева у соответствующей вершины просто отсутствует левая или правая связь.

Максимальное число вершин в дереве равно 18; это объясняется тем, что каждая вершина занимает 4 позиции экранной строки и, кроме того, 4 начальных позиции отводятся под дополнительную информацию (номера уровней). Поэтому, с учетом того, что ширина экранной строки в окне задачника равна 78, вписать в нее можно только дерево с не более чем 18 вершинами. Впрочем, в заданиях рекомендуется использовать не более 16 вершин, начиная вывод дерева с 11 экранной позиции; это дает возможность использовать левую часть строк для отображения других данных, например, указателей, связанных с данным деревом.

Количество уровней дерева ограничивается только количеством его вершин и, таким образом, может достигать 18. В соответствии с общепринятой практикой уровни дерева нумеруются от 0. Номер уровня отображается в левой части области, отведенной под изображение дерева; он выделяется цветом и отделяется от изображения дерева двоеточием.

Если количество уровней превышает число экранных строк, выделенных для отображения дерева, то для дерева становится возможной *прокрутка*, подобная прокрутке файловых данных (точнее, данных из текстовых файлов, поскольку для деревьев, как и для текстовых файлов, прокрутка выполняется в вертикальном направлении). На возможность прокрутки указывают дополнительные символы, которые изображаются слева от номера уровня. Символ «стрелка вверх», расположенный на первой экранной строке, отведенной для отображения дерева, означает, что изображение дерева можно пролистать вверх, а символ «стрелка вниз», расположенный на последней экранной строке, отведенной для отображения дерева, означает, что изображение дерева можно пролистать вниз (см. пример 8).

Обычно первая строка, отводимая под изображение дерева, содержит его корень и помечается слева числом 0 (нулевой уровень дерева). Единственная ситуация, когда это правило нарушается, связана с ошибочным формированием бинарного дерева с обратной связью в случае, если поле Parent корня не содержит значение nil. В этой ситуации перед строкой с изображением корня дерева помещается еще одна строка, в которой над корнем изображается красная звездочка — признак ошибки.

Изображение дерева может также содержать строки, расположенные ниже последнего уровня; эти строки могут потребоваться для вывода имен указателей, связанных с вершинами-листьями, расположенными на последнем уровне, а также для вывода звездочек, отмечающих ошибочные ссылки Left или Right для вершин, расположенных на последнем уровне дерева. Заметим, что ссылка считается ошибочной в двух случаях:

- если она содержит неверный адрес,
- если она ссылается на вершину, которую нельзя отобразить, поскольку для этой вершины не предусмотрено экранного места.

В частности, звездочки обязательно будут выведены при попытке отобразить на экране дерево с количеством вершин, превышающим 18.

Приведем примеры изображений линейных списков и деревьев. В этих примерах предполагается, что текущим языком программирования является Паскаль; для других языков вместо nil используются обозначения нулевых указателей (или *нулевых*

объектов — см. далее описание процедуры SetObjectStyle), соответствующие этим языкам.

Пример 1

```
P1
24 - 23 >nil
```

Первый элемент данного списка связан с указателем P_1; список содержит два элемента и является *односвязным*: на это указывает символ «–» между элементами, означающий, что поле Next первого элемента (со значением 24) указывает на второй элемент (со значением 23). Поле Next второго элемента равно nil.

Пример 2

```
      P1          P2
nil< 14 = 23 = 34 >nil
```

Данный список является двусвязным (двойная связь, использующая оба поля связи — Next и Prev, — обозначается знаком «=»), причем в задании с этим списком связаны два указателя: P_1 указывает на его первый, а P_2 — на его последний элемент.

Пример 3

```
           P0
<< = 15 - 23 = 34 = >>
```

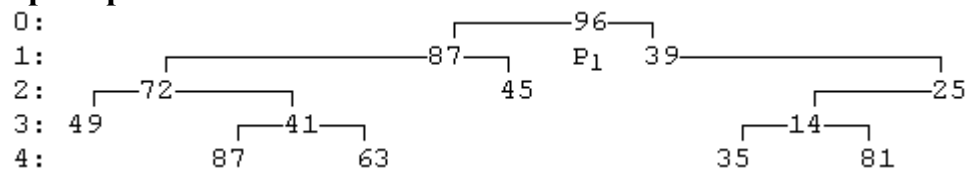
Данный список является двусвязным *циклическим* списком (на его цикличность указывают символы «<<» и «>>»), однако одна из его связей отсутствует. А именно, элемент 23 (на который указывает указатель P_0) не связан с предыдущим элементом 15, то есть поле Prev элемента 23 содержит ошибочное значение (например, равно nil). При правильно разработанном задании подобная ситуация может возникнуть только для ошибочных списков, созданных в программе учащегося. Заметим, что связь в другом направлении (от 15 к 23) имеется, то есть поле Next элемента 15 указывает на элемент 23.

Пример 4

```
PXPY
95 - 63 -.34.- >>
```

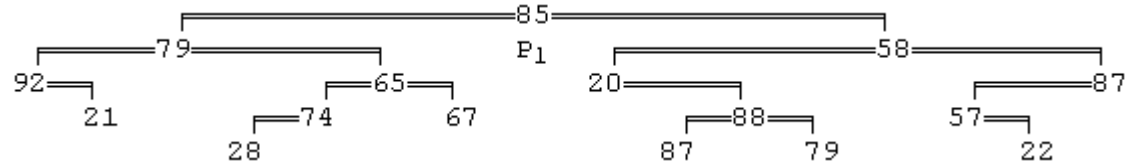
Данный список является односвязным циклическим списком. Он имеет две особенности. Во-первых, на элемент 95 указывают сразу два указателя (P_X и P_Y), и, во-вторых, элемент 34 должен быть размещен в памяти процедурой New при *выполнении* задания (на это указывают обрамляющие его точки). Подобные элементы, естественно, могут содержаться только в *результирующих* списках. Они определяются с помощью процедуры SetNewNode.

Пример 5



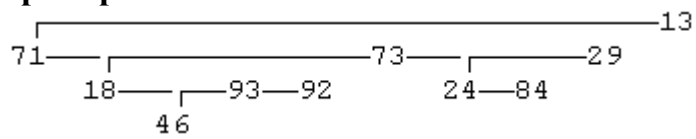
Так выглядит на экране бинарное дерево глубины 4. С корнем этого дерева (поле Data которого равно 96) связан указатель P_1.

Пример 6



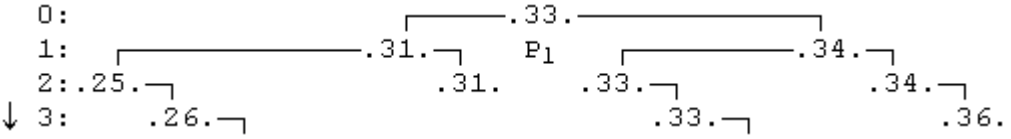
Так выглядит на экране бинарное дерево с обратной связью (номера уровней на данной иллюстрации не указаны).

Пример 7



Так выглядит на экране дерево общего вида (номера уровней и имена связанных с деревом указателей на данной иллюстрации не указаны). В данном случае корень дерева 13 имеет три непосредственных потомка: вершины 71, 73 и 29. Напомним, что в дереве общего вида поле Left определяет первую (левую) дочернюю вершину, а поле Right — очередную (правую) вершину-*сестру*.

Пример 8



Так выглядит на экране бинарное дерево с включенным режимом прокрутки. Дополнительной особенностью этого дерева является наличие точек около каждой его вершины. Это означает, что данное дерево является результирующим, причем память для всех его вершин должна быть выделена в программе учащегося.

```
procedure ShowPointer(NP: integer);
```

Процедура обеспечивает отображение указателя с номером NP при выводе текущего линейного списка или дерева. Например, ее вызов вида ShowPointer(1) обеспечил отображение указателя P_1 в примерах 1, 2 и 5. Если указатель номер NP равен nil, то вызов процедуры ShowPointer игнорируется без вывода сообщения об ошибке. Если указатель с номером NP не является нулевым и не связан ни с одним из элементов списка, то выводится сообщение об ошибке.

С одним элементом списка или дерева можно связать не более двух указателей (исключение составляет последний элемент списка, с которым можно связать не более трех указателей). Порядок вызова процедур ShowPointer для одного и того же элемента списка является произвольным; при отображении указателей, связанных с одним и тем же элементом, они выводятся в отсортированном порядке (например, P_3P_6). В случае списков имена указателей отображаются *над* элементом, и при наличии нескольких указателей на один элемент их имена располагаются слева направо. В случае деревьев имена указателей располагаются *под* элементом, и при наличии нескольких указателей на один элемент их имена располагаются одно под другим. Если количество указателей, связываемых процедурами ShowPointer с данным элементом списка, превосходит максимально допустимое (например, с последним элементом связывается четыре различных указателя), то список связанных указателей дополняется символом ошибки — звездочкой (например, $P_1P_2P_3$*). Если с элементом бинарного дерева связывается более двух указателей, то под вторым указателем изображается еще один указатель вида P*. Символ ошибки * выделяется красным цветом.

Если указатель надо связать с элементом списка или дерева, помеченным точками (см. пример 8), то вызов процедуры ShowPointer для данного указателя надо выполнить *до того*, как для соответствующего элемента списка или дерева будет вызвана процедура SetNewNode (в противном случае при вызове процедуры ShowPointer будет выведено сообщение об ошибке «Не найден элемент с адресом P1»).

```
procedure SetNewNode(NNode: integer);
```

Процедура определяет для текущего списка элемент с номером NNode (нумерация ведется от 1) как элемент, который требуется разместить в памяти с помощью процедуры New *в ходе выполнения задания* (подобные элементы выделяются в списке с помощью обрамляющих точек — см. пример 4). Она также позволяет аналогичным образом выделить элемент текущего дерева (см. пример 8); при этом предполагается, что элементы дерева нумеруются в *префиксном* порядке (в частности, корень дерева всегда имеет номер 1; по поводу префиксного порядка см. задание Tree13).

Данная процедура может применяться только к *результирующим* спискам и деревьям (для определения которых используются процедуры группы Result: ResultList, ResultBinTree, ResultTree). Если результирующий список или дерево не содержит элемента с номером NNode, то выводится сообщение об ошибке.

Следует заметить также, что если указатель на элемент номер NNode требуется отобразить на экране (с помощью процедуры ShowPointer), то это необходимо сделать *до вызова* процедуры SetNewNode.

Если при выполнении задания учащийся будет выделять память (процедурой New для Паскаля или аналогичными средствами для других языков) для тех элементов результирующего списка или дерева, для которых это не предусмотрено заданием, то соответствующие элементы в результирующем списке (дереве) будут обрамлены точками, что приведет к сообщению «Ошибочное решение».

```
procedure SetDisposedNode(NNode: integer);
```

Процедура определяет для текущего списка или дерева элемент с номером NNode (нумерация ведется от 1, элементы дерева нумеруются в префиксном порядке), который требуется удалить из динамической памяти в ходе выполнения задания. Данная процедура может применятся только к исходным спискам и деревьям (для определения которых используются процедуры группы Data: DataList, DataBinTree, DataTree). Если исходный список или дерево не содержит элемента с номером NNode, то выводится сообщение об ошибке.

Элементы, помечаемые с помощью процедуры SetDisposedNode, выделяются на экране цветом меньшей яркости. Если они не удаляются из памяти в ходе выполнения задания, то их цвет изменяется на красный и выводится соответствующее сообщение об ошибке.

```
procedure SetObjectStyle;
```

Данная процедура устанавливает «объектный стиль» для динамических структур и связанных с ними ссылок при выполнении задания в среде PascalABC.NET. Она должна вызываться при формировании заданий, ориентированных на использование не записей TNode и связанных с ними указателей PNode, а *объектов* класса Node (данный класс определен в вариантах задачника для языков платформы .NET и, в частности, для языка PascalABC.NET).

При разработке заданий класс Node *не используется*. Даже если разрабатываемая группа заданий ориентирована на его применение, сами задания надо создавать с помощью записей TNode, указателей PNode и описанных выше процедур. Однако для *решения* подобных заданий на языке платформы .NET вместо записей и связанных с ними указателей надо применять объекты Node. Среди языков платформы .NET, поддерживаемых версией 4.8 задачника, имеется единственное исключение: это язык PascalABC.NET, в котором можно использовать как указатели, так и объекты. Именно

поэтому для данного языка предусмотрена функция SetObjectStyle (в прочих языках платформы .NET *настройка объектного стиля в заданиях на обработку динамических структур выполняется автоматически*).

Создавая задания на обработку динамических структур для языка PascalABC.NET, разработчик должен указать учащемуся требуемый способ решения, используя соответствующие термины в формулировке задания («запись» или «объект», «указатель» или «ссылка» и т. п.), а также настроив, при необходимости, вывод динамических структур на «объектный стиль», вызвав процедуру SetObjectStyle (по умолчанию применяется «стиль указателей», подробно описанный выше). Процедура SetObjectStyle должна быть вызвана после процедуры CreateTask; необходимо также, чтобы ее вызов располагался *перед* вызовами любых процедур, обеспечивающих добавление к заданию динамических структур и связанных с ними указателей. В результате ее вызова изменяется отображение этих элементов данных, а именно:

- вместо текста ptr для непустого указателя указывается текст Node (то есть имя непустого *объекта* типа Node);
- в стандартном комментарии к указателю вместо буквы P указывается буква A, например, A_1 = Node (эту особенность следует учитывать в формулировке задания, используя в ней вместо имен указателей P_1, P_2 и т. д. имена *объектов* A_1, A_2 и т. д.).

Аналогичные изменения (символа P на символ A) выполняются и при отображении указателей, связанных с динамическими структурами. Например, односвязный список, приведенный в примере 1, при установке объектного стиля будет иметь следующий вид:

```
A1
24 - 23 >nil
```

Слово nil осталось неизменным, так как в PascalABC.NET оно применяется для обозначения как нулевых указателей, так и «пустых» объектов. При использовании других языков платформы .NET обозначения «пустых» объектов соответствующим образом корректируются; так, для языка C# применяется обозначение null, а для языка VB.NET — обозначение Noth (от слова Nothing).

Заметим, что объектный стиль используется в базовых группах ObjDyn и ObjTree, имеющихся в варианте задачника для системы PascalABC.NET. Эти группы с содержательной точки зрения полностью аналогичны группам Dynamic и Tree, ориентированным на применение указателей.

4. Форматирование текста заданий

4.1. Общие сведения

В конструкторе учебных заданий PT4TaskMaker предусмотрена возможность *форматирования* текста заданий, а также преамбул для группы и ее подгрупп. Форматирование выполняется с помощью набора *управляющих последовательностей* (команд), большинство из которых имеет вид *символ*.

Используя управляющие последовательности, можно выполнять следующие действия по форматированию текста в окне задачника:

- добавлять в текст специальные символы, в том числе символы шрифта Symbol и буквы западноевропейских языков;

- выделять фрагмент текста полужирным шрифтом;
- использовать в тексте нижние и верхние индексы;
- добавлять в текст задания ссылки на другие задания этой же группы, не указывая при этом название группы (что позволяет корректно изменять эти ссылки при включении задания в другие группы);
- добавлять в текст задания элементы, зависящие от текущего языка программирования (в частности, обозначения логических констант).

Все описанные выше действия обеспечивают требуемое форматирование текста задания как в окне задачника, так и в html-описании данного задания. Аналогичное форматирование (в частности, использование нижних и верхних индексов) можно применять и в текстах комментариев, которые выводятся в окне задачника в разделах исходных и результирующих данных.

Кроме того, имеются управляющие последовательности, не влияющие на текст задания в окне задачника, однако обеспечивающие дополнительное форматирование этого текста (и текста комментариев для группы и ее подгрупп) в html-описании задания или группы заданий. Данные управляющие последовательности позволяют:

- выделять имена переменных курсивом;
- использовать более разнообразное выделение фрагментов текста (помимо полужирного начертания можно установить курсивное начертание, выделение моноширинным шрифтом и специальное выделение);
- разбивать текст задания и преамбулы на отдельные абзацы;
- устанавливать для требуемых фрагментов текста режим вывода с центрированием или с отступом;
- обеспечивать вывод фрагментов текста в несколько столбцов, с возможностью установки способа выравнивания для каждого столбца.

Напомним, что для вывода на экран html-страницы с описанием задания или группы заданий достаточно вызвать процедуру Task, указав в конце ее параметра (имени задания или группы заданий) суффикс #. Кроме того, html-страницы с описанием групп заданий можно генерировать с помощью модуля PT4Demo, используя кнопку  в окне этого модуля.

4.2. Таблица управляющих последовательностей

Управляющие последовательности, приведенные в таблице, можно использовать в формулировках заданий (параметр S процедуры TaskText), комментариях к исходным и результирующим данным (параметр Cmt в процедурах групп Data и Result), а также в дополнительных описаниях (преамбулах) групп и подгрупп учебных заданий (параметр S в процедуре CommentText).

Управляющие последовательности, использованные в параметрах процедур групп Data и Result, влияют только на представление соответствующих комментариев в окне задачника (см. столбец «Окно задачника»). Управляющие последовательности, использованные в параметре S процедуры CommentText, обеспечивают соответствующее форматирование преамбулы к группе заданий и ее подгруппам в тексте html-страницы с описанием группы заданий (см. столбец «Html-страница»). Управляющие последовательности, использованные в параметре S процедуры TaskText, влияют на вид формулировок заданий как в окне задачника, так и в html-описаниях.

Все последовательности вида *символ*, не указанные в приведенной ниже таблице, игнорируются как при выводе текста в окне задачника, так и при его отображения в виде html-страницы.

В *заголовках подгрупп*, указываемых в процедуре CreateTask (параметр SubgroupName), а также в *тексте краткого описания группы*, указываемого в процедуре CreateGroup (параметр GroupDescription), управляющие последовательности не обрабатываются. При указании в этих строках управляющих последовательностей они дословно воспроизводятся и в окне задачника, и в html-описании.

Таблица 1. Управляющие последовательности

Команда	Описание	Окно задачника	Html-страница
Ссылки на другие задания данной группы			
число	Имя задания из текущей группы с номером, меньшим текущего на величину указанного десятичного числа (или `*********`, если результирующий номер оказывается меньшим 1)		
\0*число*	Имя задания из текущей группы с номером, большим текущего на величину указанного десятичного числа (или `*********`, если результирующий номер оказывается большим 999)		
Пробелы			
`\,`	Малый неразрывный пробел	*Игнорируется*	` `
`\;`	Средний неразрывный пробел	*Пробел*	` `
`~`	Обычный неразрывный пробел	*Пробел*	` `
`\q`	Большой пробел	*Игнорируется*	*Пробел*` `*Пробел*` `
`\Q`	Двойной большой пробел	*Игнорируется*	*2 копии большого пробела*
Символы			
`\\`	Обратная косая черта (\)	`\`	`\`
`\&`	Амперсанд (&)	`&`	`&`
`\{`	Открывающая фигурная скобка ({)	`{`	`{`
`\}`	Закрывающая фигурная скобка (})	`}`	`}`
`\~`	Символ «волна» (~)	`~`	`~`

Команда	Описание	Окно задачника	Html-страница
\^	Символ «шапочка» (^)	^	^
_	Символ подчеркивания (_)	_	_
\.	Многоточие (…)	. . . *(три точки)*	…
\=	Длинное тире (—)	—	—
\:	Символ диапазона (короткое тире)(–)	–	–
\-	Символ «минус» (−)	−	−
\<	Открывающие угловые кавычки («)	«	«
\>	Закрывающие угловые кавычки (»)	»	»
*	Символ умножения (·)	·	·
\+	Символ «плюс-минус» (±)	±	±
\o	Символ градуса (°)	°	°
\x	Символ «косой крест» (×)	×	×
\a	Греческая буква «альфа» (α)	α	α
\p	Греческая буква «пи» (π)	π	π
\e	Греческая буква «эпсилон» (ε)	ε	ε
\l	Символ «меньше или равно» (≤)	≤	≤
\g	Символ «больше или равно» (≥)	≥	≥
\n	Символ «не равно» (≠)	≠	≠
\X	Символ «bullet» (•)	•	•

Команда	Описание	Окно задачника	Html-страница
\h*XX*, где *XX* — двузначное 16-ричное число	Символ с кодом *XX* из второй половины кодовой таблицы для западноевропейских языков ANSI Latin-1 (CP1252)	*Соответствующий символ*	&#*DDD*; *(DDD — десятичный код соответствующего символа в кодировке Unicode)*
\H*XX*, где *XX* — двузначное 16-ричное число	Символ с кодом *XX* Windows-шрифта Symbol	*Соответствующий символ*	&#*DDD*; *(DDD — десятичный код соответствующего символа в кодировке Unicode) или* &#*DDD*; *(DDD — десятичное число, равное 16-ричному числу XX)*
Элементы текста, зависящие от текущего языка программирования			
\R	Метка начала квадратного корня	Sqrt(*(Pascal, VB.NET, C#)* sqrt(*(C++)* Sqr(*(Visual Basic версий 5 и 6)*	(
\r	Метка конца квадратного корня	)	)^{1/2}
\t	Логическая константа «True»	True *(Pascal, Visual Basic, VB.NET)* true *(C++, C#)*	
\f	Логическая константа «False»	False *(Pascal, Visual Basic, VB.NET)* false *(C++, C#)*	
\N	Нулевой указатель	nil *(Pascal)* NULL *(C++)*	
\O	Нулевая ссылка для объекта .NET	null *(C#)* Nothing *(VB.NET)* nil *(PascalABC.NET)*	
Индексы			
_*символ*	Односимвольный нижний индекс	*Символ выводится как нижний индекс*	_{*символ*}
^*символ*	Односимвольный верхний индекс	*Символ выводится как верхний индекс*	^{*символ*}

Команда	Описание	Окно задачника	Html-страница
_{	Метка начала многосимвольного нижнего индекса	*Переход в режим нижнего индекса*	<sub>
^{	Метка начала многосимвольного верхнего индекса	*Переход в режим верхнего индекса*	<sup>
}	Метка конца текущего (верхнего или нижнего) многосимвольного индекса	*Выход из режима индекса*	</sub> *или* </sup>
Выделение **(в окне задачника любой режим выделения приводит к полужирному выделению текста)**			
\B	Метка начала полужирного выделения	*Переход в режим выделения*	
\b	Метка конца полужирного выделения	*Выход из режима выделения*	
\I	Метка начала курсивного выделения	*Переход в режим выделения*	
\i	Метка конца курсивного выделения	*Выход из режима выделения*	
\S	Метка начала специального выделения	*Переход в режим выделения*	
\s	Метка конца специального выделения	*Выход из режима выделения*	
Дополнительное форматирование html-страниц **(в окне задачника данные управляющие последовательности игнорируются)**			
{	Метка начала выделения переменной		
}	Метка конца выделения переменной		

Команда	Описание	Окно задачника	Html-страница
\M	Метка начала моноширинного текста		<tt>
\m	Метка конца моноширинного текста		</tt>
|	Разрыв строки		
\P	Начало нового абзаца		*Завершается предыдущий абзац и создается абзац стиля* ptTaskContinue *(при использовании в формулировке задания) или* ptComment *(при использовании в тексте преамбулы)*
\[	Метка начала абзаца с центрированием		*Завершается предыдущий абзац и создается абзац стиля* ptTaskCenter *(при использовании в формулировке задания) или* ptCommentCenter *(при использовании в тексте преамбулы)*
\(	Метка начала абзаца с отступом		*Завершается предыдущий абзац и создается абзац стиля* ptTaskQuote *(при использовании в формулировке задания) или* ptCommentQuote *(при использовании в тексте преамбулы)*
\]	Метка конца абзаца с центрированием		*Завершается предыдущий абзац и создается абзац стиля* ptTaskContinue *(при использовании в формулировке задания) или* ptCommentContinue *(при использовании в тексте преамбулы)*
\)	Метка конца абзаца с отступом		
\J	Метка начала режима выравнивания по столбцам (после нее указывается набор символов r, l, c, оканчивающийся символом &)		<table>
\j	Метка конца режима выравнивания по столбцам		</table>

Команда	Описание	Окно задачника	Html-страница
`&`	Переход к новому столбцу в режиме выравнивания по столбцам		*Добавляется тег* `<td>` *с соответствующим выравниванием; для первого столбца предварительно указывается тег* `<tr>`

4.3. Дополнительные сведения об использовании управляющих последовательностей

Необходимость в специальных командах для генерации ссылок на другие задания группы объясняется тем, что любое имеющееся задание может быть импортировано в группу с другим именем (с помощью процедуры UseTask), и поэтому все ссылки на другие задания этой группы также потребуется откорректировать, указав в них новое имя группы. Разумеется, в подобной ситуации необходимо переносить в новую группу *все* задания, содержащие ссылки друг на друга. Следует заметить, что *разность* между номерами ссылающихся друг на друга заданий не обязана быть такой же, как в исходной группе заданий. Если в новой группе задания находятся на другом «расстоянии» друг от друга, то для указания правильной ссылки достаточно внести соответствующую *поправку* в параметр процедуры UseTask (см. описание данной процедуры в п. 2.3).

Наличие нескольких видов *неразрывных пробелов*, не различающихся в тексте заданий и html-страниц, связано с планируемой в дальнейшем возможностью генерации текста заданий в других форматах (в частности, в формате системы TeX, в котором данные виды пробелов различаются). Приведем рекомендации по использованию неразрывных пробелов:

- вокруг символов =, <, > указывается обычный неразрывный пробел ~; исключением являются фрагменты текста в скобках вида `(> 0)`, в которых рекомендуется использовать малый пробел: `(>\,0)`;
- неразрывный пробел ~ указывается также между текстом и переменной: `стороны~{a} и~{b}`;
- вокруг символов + и − ставится средний пробел **`\;`**;
- символы умножения `\*` и деления `/` пробелами не обрамляются; исключением служит ситуация, когда слева и справа от символа деления указываются прописные буквы; в этом случае желательно использовать обрамление малыми пробелами.

Приведем пример оформления формул (данный пример взят из задания Begin39; обратите внимание на выделение переменных с помощью фигурных скобок, а также на команды, обеспечивающие вывод индексов, выделение квадратного корня и центрирование формулы):

```
TaskText('Найти корни \Iквадратного уравнения\i ' +
  '{A}\*{x}^2\;+\;{B}\*{x}\;+\;{C}~=~0, заданного', 0, 1);
TaskText('своими коэффициентами~{A}, {B}, {C} ' +
  '(коэффициент~{A} не равен~0), если известно,', 0, 2);
TaskText('что дискриминант уравнения положителен. ' +
  'Вывести вначале меньший, а затем',0,3);
TaskText('больший из найденных корней. Корни квадратного ' +
```

```
    'уравнения находятся по формуле', 0, 4);
  TaskText('\[{x}_{1,\,2}~=~(\-{B}\;\+\;\R{D}\r)/(2\*{A}),\] ' +
    'где {D}~\= \Iдискриминант\i, ' +
    'равный {B}^2\;\-\;4\*{A}\*{C}.', 0, 5);
```

В результате обработки данной формулировки задания в окне задачника будет выведен текст:

Найти корни **квадратного уравнения** $A \cdot x^2 + B \cdot x + C = 0$, заданного своими коэффициентами A, B, C (коэффициент A не равен 0), если известно, что дискриминант уравнения положителен. Вывести вначале меньший, а затем больший из найденных корней. Корни квадратного уравнения находятся по формуле $x_{1,2} = (-B \pm Sqrt(D))/(2 \cdot A)$, где D — **дискриминант**, равный $B^2 - 4 \cdot A \cdot C$.

В html-описании этот же текст будет отформатирован следующим образом:

Begin39. Найти корни *квадратного уравнения* $A \cdot x^2 + B \cdot x + C = 0$, заданного своими коэффициентами *A*, *B*, *C* (коэффициент *A* не равен 0), если известно, что дискриминант уравнения положителен. Вывести вначале меньший, а затем больший из найденных корней. Корни квадратного уравнения находятся по формуле

$$x_{1,2} = (-B \pm (D)^{1/2})/(2 \cdot A),$$

где *D* — *дискриминант*, равный $B^2 - 4 \cdot A \cdot C$.

Для указания кавычек в тексте задания следует использовать управляющие последовательности \< и \>.

Управляющие последовательности \t, \f, \N, \O для логических констант, нулевых указателей и объектов генерируют текст, зависящий от выбранного в данный момент языка программирования.

Обычные пробелы, указанные после управляющих последовательностей \q, \Q, \P, \[, \(, \], \), \| и &, учитываются только в тексте задачника (и пропускаются в тексте html-страниц).

Режим специального выделения, устанавливаемый парными командами \S и \s, в окне задачника приводит к выделению полужирным шрифтом, а в html-описании обеспечивает выделение фрагмента текста, аналогичное выделению, используемому для *имени задания* в начале его формулировки (в приведенном выше фрагменте html-описания так выделено имя задания «Begin39»). Данный режим рекомендуется использовать для выделения *заголовков*, размещаемых в начале абзаца (например, если формулировка задания завершается абзацем, содержащим указание, то с помощью специального выделения целесообразно выделить текст «Указание» в начале этого абзаца).

Команды выделения переменной { и } не влияют на ее вид в окне задачника, но обеспечивают ее выделение курсивом в тексте html-страницы. В индексах команды выделения переменной не учитываются, а любые *латинские* буквы в них автоматически выделяются курсивом.

В односимвольных индексах нельзя указывать управляющие последовательности для вывода специальных символов, поэтому при необходимости применения в индексах специальных символов следует использовать режим многосимвольных индексов. Метки индексов не могут быть вложенными. В индексах не допускается использование меток выделения \I, \B, \S и наоборот, внутри выделенного текста не допускается указывать индексы.

Выделенные фрагменты не могут содержать меток выделения другого вида. Режим индексов и выделения, заданный в одной процедуре TaskText или CommentText, не переносится на текст, определяемый при последующих вызовах этих процедур. Если в тексте отсутствуют команды завершения текущего режима (индексов или вы-

деления), то режим автоматически завершается при достижении конца текста, определяемого в текущей процедуре TaskText или CommentText.

В команде начала режима выравнивания по столбцам символы `r`, `l`, `c` определяют *способ выравнивания* в каждом столбце текста (`r` — выравнивание по правому краю, `l` — выравнивание по левому краю, `c` — выравнивание по центру). Их количество должно быть равно числу столбцов. В каждом столбце должен быть хотя бы один непробельный символ (для пустого столбца достаточно указать малый неразрывный пробел `\,`).

Ввиду сложности управляющих последовательностей, связанных с выравниванием по столбцам, приведем пример их использования (пример взят из задания If26):

```
TaskText('Для данного вещественного~{x} найти значение ' +
  'следующей функции~{f},', 0, 1);
TaskText('принимающей вещественные значения:', 0, 2);
TaskText('\[\Jrcrl&\,&\,&       \-{x},& если {x}~\l~0,',
  26, 3);
TaskText('&{f}({x})&~=~&{x}^2,& если 0~<~{x}~<~2,', 26, 4);
TaskText('&\,&\,&        4,& если {x}~\g~2.\j\]', 26, 5);
```

В результате обработки данной формулировки задания в окне задачника будет выведен текст:

Для данного вещественного x найти значение следующей функции f,
принимающей вещественные значения:

$$f(x) = \begin{cases} -x, & \text{если } x \le 0, \\ x^2, & \text{если } 0 < x < 2, \\ 4, & \text{если } x \ge 2. \end{cases}$$

В html-описании этот же текст будет отформатирован следующим образом:

If26. Для данного вещественного x найти значение следующей функции f, принимающей вещественные значения:

$$f(x) = \begin{cases} -x, & \text{если } x \le 0, \\ x^2, & \text{если } 0 < x < 2, \\ 4, & \text{если } x \ge 2. \end{cases}$$

Команду разрыва строки `\|` можно использовать как в обычном тексте, так и в режиме выделения с отступом или центрированием (в режиме выравнивания по столбцам переход на новую строку выполняется автоматически). Для корректного отображения текста, выровненного по столбцам, данный текст рекомендуется дополнительно центрировать (как в приведенном выше примере) или использовать для него режим выравнивания с отступом.

Управляющая последовательность `\P` предназначена для разделения абзацев. В тексте, отображаемом в окне задачника, данная команда игнорируется (подобно прочим командам, связанным с разделением на абзацы). Для нее не предусмотрено парной завершающей команды, поскольку необходимые теги при переходе к новому абзацу добавляются в текст html-страницы автоматически. Пробелы после команды `\P` при генерации html-страницы игнорируются, однако они учитываются при отображении текста в окне задачника.

4.4. Генерация специальных символов

Используя две «универсальные» управляющие последовательности `\h` и `\H`, можно включать в текст задания или преамбулы специальные символы, входящие во

вторую половину кодовой таблицы для западноевропейских языков ANSI Latin-1 (команда `\h`) или содержащиеся в Windows-шрифте Symbol (команда `\H`). После имени каждой из этих команд следует указать двузначное шестнадцатеричное число, определяющее код требуемого символа; при этом шестнадцатеричные цифры A, B, C, D, E, F можно указывать в любом регистре. Если двухсимвольный текст после команд нельзя преобразовать в шестнадцатеричное число или число не является допустимым, то команды возвращают символ «?» (знак вопроса).

В случае команды `\h` (символы таблицы Latin-1) допустимыми считаются числа из диапазона 128–255, за исключением кодов неотображаемых символов, например, кода неразрывного пробела 160 (A0) или «мягкого» переноса 173 (AD). Символы таблицы Ansi Latin-1 с кодами 128–159 имеют в кодировке Unicode другие значения кодов; при генерации html-описаний для этих символов используются их коды в таблице Unicode.

С помощью команды `\H` можно получить только *часть* символов, определенных в Windows-шрифте Symbol. Исключены символы, уже присутствующие в таблицах ASCII и ANSI Latin-1 (например, цифры и знаки препинания) или имеющие идентичное начертание с символами из этих таблиц (например, заглавные греческие буквы, совпадающие по начертанию с латинскими: A, B, E, H, X и т. д.). Кроме того, исключены символы с кодами 230–239 и 243–254, представляющие собой фрагменты больших скобок.

Следует заметить, что для части математических символов нельзя обеспечить их правильное отображение в каждом из трех наиболее популярных веб-браузеров (Microsoft Internet Explorer, Mozilla Firefox и Opera) без использования средств веб-программирования. В браузерах Internet Explorer и Firefox можно подключать шрифты Windows, в том числе шрифт Symbol, однако в Opera это сделать нельзя. С другой стороны, в Opera и Firefox для отображения всех стандартных математических символов достаточно указать их код в Unicode-кодировке, однако в стандартных Windows-шрифтах, используемых браузером Internet Explorer, часть символов с требуемыми кодами отсутствует. При реализации команды `\H` для вывода подобных символов в html-документе был выбран вариант, обеспечивающий их правильное отображение в браузере Internet Explorer (и Mozilla Firefox): для этого используется Windows-шрифт Symbol. Однако в браузере Opera (и других браузерах, не поддерживающих шрифты Windows) *данные символы будут отображаться неправильно*.

Примечание. Для возможности использования Windows-шрифтов в браузере Mozilla Firefox следует установить режим «Разрешить веб-сайтам использовать свои шрифты вместо установленных». Соответствующий флажок находится в окне «Шрифты», которое можно отобразить с помощью следующей последовательности действий: выполнить команду меню «Инструменты | Настройки...», в появившемся окне «Настройки» перейти на вкладку «Содержимое» и в разделе «Шрифты и цвета» нажать кнопку «Дополнительно...».

С некоторыми часто используемыми специальными символами связаны особые управляющие последовательности (см. таблицу управляющих последовательностей, раздел «Символы»). Все подобные символы правильно отображаются во всех перечисленных выше браузерах.

Хотя символ пересечения (∩, код 8745) имеется в стандартных Windows-шрифтах, прочие символы, связанные с множествами (объединение, вложение, принадлежность и т. д.), в этих шрифтах отсутствуют. Для того чтобы все обозначения,

связанные с множествами, выглядели в html-документе единообразно, для отображения символа пересечения (команда \Hc7) используется соответствующий символ из шрифта Symbol.

Ниже приводятся таблицы всех символов, которые можно получить с помощью универсальных команд \h и \H. Таблица 2 содержит символы, генерируемые командой \h, а таблица 3 — символы, генерируемые командой \H. Команды из таблицы 3, связанные с теми символами, которые будут неверно отображаться в браузере Opera, выделены полужирным шрифтом.

Таблица 2. Символы, генерируемые командой \h

\h80 €		\h82 ‚	\h83 *f*	\h84 „	\h85 …	\h86 †	\h87 ‡	\h88 ˆ	\h89 ‰
\h8a Š	\h8b ‹	\h8c Œ		\h8e Ž			\h91 ‘	\h92 ’	\h93 “
\h94 ”	\h95 •	\h96 –	\h97 —	\h98 ˜	\h99 ™	\h9a š	\h9b ›	\h9c œ	
\h9e ž	\h9f Ÿ		\ha1 ¡	\ha2 ¢	\ha3 £	\ha4 ¤	\ha5 ¥	\ha6 ¦	\ha7 §
\ha8 ¨	\ha9 ©	\haa ª	\hab «	\hac ¬		\hae ®	\haf ¯	\hb0 °	\hb1 ±
\hb2 ²	\hb3 ³	\hb4 ´	\hb5 µ	\hb6 ¶	\hb7 ·	\hb8 ¸	\hb9 ¹	\hba º	\hbb »
\hbc ¼	\hbd ½	\hbe ¾	\hbf ¿	\hc0 À	\hc1 Á	\hc2 Â	\hc3 Ã	\hc4 Ä	\hc5 Å
\hc6 Æ	\hc7 Ç	\hc8 È	\hc9 É	\hca Ê	\hcb Ë	\hcc Ì	\hcd Í	\hce Î	\hcf Ï
\hd0 Đ	\hd1 Ñ	\hd2 Ò	\hd3 Ó	\hd4 Ô	\hd5 Õ	\hd6 Ö	\hd7 ×	\hd8 Ø	\hd9 Ù
\hda Ú	\hdb Û	\hdc Ü	\hdd Ý	\hde Þ	\hdf ß	\he0 à	\he1 á	\he2 â	\he3 ã
\he4 ä	\he5 å	\he6 æ	\he7 ç	\he8 è	\he9 é	\hea ê	\heb ë	\hec ì	\hed í
\hee î	\hef ï	\hf0 ð	\hf1 ñ	\hf2 ò	\hf3 ó	\hf4 ô	\hf5 õ	\hf6 ö	\hf7 ÷
\hf8 ø	\hf9 ù	\hfa ú	\hfb û	\hfc ü	\hfd ý	\hfe þ	\hff ÿ		

Таблица 3. Символы, генерируемые командой `\H`

\H22 ∀	**\H24** ∃	**\H27** э	\H2d −	**\H40** ≅	\H44 Δ	\H46 Φ	\H47 Γ	\H4c Λ	\H50 Θ
\H51 Θ	\H53 Σ	\H56 ς	\H57 Ω	\H58 Ξ	\H59 Ψ	**\H5c** ∴	**\H5e** ⊥	\H61 α	\H62 β
\H63 χ	\H64 δ	\H65 ε	**\H66** ϕ	\H67 γ	\H68 η	\H69 ι	\H6a φ	\H6b κ	\H6c λ
\H6d μ	\H6e ν	\H70 π	\H71 θ	\H72 ρ	\H73 σ	\H74 τ	\H75 υ	**\H76** ϖ	\H77 ω
\H78 ξ	\H79 ψ	\H7a ζ	**\Ha1** ϒ	\Ha2 ′	\Ha3 ≤	\Ha5 ∞	\Ha7 ♣	\Ha8 ♦	\Ha9 ♥
\Haa ♠	\Hab ↔	\Hac ←	\Had ↑	\Hae →	\Haf ↓	\Hb2 ″	\Hb3 ≥	**\Hb5** ∝	\Hb6 ∂
\Hb9 ≠	\Hba ≡	\Hbb ≈	\Hbd |	\Hbe —	**\Hbf** ↵	\Hc0 ℵ	**\Hc1** ℑ	**\Hc2** ℜ	**\Hc3** ℘
\Hc4 ⊗	**\Hc5** ⊕	**\Hc6** ∅	**\Hc7** ∩	**\Hc8** ∪	**\Hc9** ⊃	**\Hca** ⊇	**\Hcb** ⊄	**\Hcc** ⊂	**\Hcd** ⊆
\Hce ∈	**\Hcf** ∉	**\Hd0** ∠	**\Hd1** ∇	\Hd5 ∏	\Hd6 √	**\Hd9** ∧	**\Hda** ∨	**\Hdb** ⇔	**\Hdc** ⇐
\Hdd ⇑	**\Hde** ⇒	**\Hdf** ⇓	\He0 ◊	\He1 ‹	\He5 ∑	\Hf1 ›	\Hf2 ∫		

5. Примеры

5.1. Создание простейшей сводной группы

Вначале приведем пример создания наиболее простого варианта группы заданий — *сводной группы*, в которой не разрабатываются новые задания, а лишь производится перекомпоновка заданий из имеющихся групп. Создадим группу заданий MakerDemo, в которую импортируем два первых задания из базовой группы Begin. Следуя правилам об именовании dll-файлов с группами заданий, дадим нашему проекту имя PT4MakerDemo.

Будем предполагать, что проект создается в среде Borland Delphi. В этом случае файл должен иметь расширение .dpr. В среде Free Pascal Lazarus текст проекта будет таким же; потребуется лишь сохранить его в файле с расширением .lpr и настроить свойства создаваемого проекта, установив режим совместимости с Delphi. Для этого достаточно поместить в начало lpr-файла следующую директиву компилятора:

```
{$mode delphi}
```

Кроме того, при разработке проекта и в среде Delphi, и в среде Lazarus целесообразно установить режимы проверки диапазона и поддержки Ansi-строк («длинных строк»). Для этого в начале файла достаточно указать следующую строку:

```
{$R+,H+}
```

Режим проверки диапазона полезен, если при разработке заданий используются массивы, поскольку он позволяет сразу выявить ошибки, связанные с выходом индексов элементов массива за допустимый диапазон. Режим длинных строк необходим при использовании в программе функций TextSample и EnTextSample (см. п. 2.7), поскольку они возвращают строки, длина которых превосходит 255 символов.

Файл PT4MakerDemo.dpr, содержащий сводную группу заданий, является кратким и имеет стандартную структуру:

```
library PT4MakerDemo;

uses PT4TaskMaker;

procedure InitTask(num: integer); stdcall;
begin
  case num of
  1..2: UseTask('Begin', num);
  end;
end;

procedure inittaskgroup;
begin
  CreateGroup('MakerDemo', 'Примеры различных задач',
    'М. Э. Абрамян, 2009', 'qwqfsdf13dfttd', 2, InitTask);
end;

exports inittaskgroup, activate;

begin
end.
```

К библиотеке подключается модуль PT4TaskMaker, после чего в ней описывается основная процедура группы заданий InitTask, определяющая задание по его номеру. Поскольку мы не создавали своих заданий, в данной процедуре используется только стандартная процедура UseTask, позволяющая импортировать задания из имеющихся групп. В нашем случае импортируются задания с номерами 1 и 2 из группы Begin.

Затем описывается процедура инициализации данной группы заданий. Она должна иметь стандартное имя inittaskgroup (***набранное строчными, то есть маленькими буквами***) и включаться в список exports процедур, экспортируемых данной библиотекой. В этой процедуре вызывается процедура CreateGroup, в которой задаются настройки создаваемой группы: имя ('MakerDemo'), описание ('Примеры различных задач'), сведения об авторе, строковый ключ, число заданий (2) и основная процедура группы (InitTask).

Примечание. В приведенном примере список exports содержит также имя процедуры activate. Эта процедура определена в модуле PT4MakerDemo; указывать ее в списке exports необходимо лишь для проектов, разрабатываемых в среде Free Pascal Lazarus (в проектах Delphi данная процедура включается в список exports автоматически). Если при разработке новой группы заданий в среде Lazarus процедура activate не будет включена в список exports, то при попытке вызвать задание из этой группы будет выведено соответствующее сообщение об ошибке.

5.2. Тестирование созданной группы

Для успешной компиляции проекта с созданной группой необходимо, чтобы в его каталоге находился файл PT4TaskMaker.pas (этот файл можно скопировать из подкаталога TASKMAKE системного каталога комплекса Teacher Pack for PT4; по умолчанию системный каталог размещается в каталоге Program Files и имеет имя PT4TeacherPack). Однако даже при успешной компиляции проекта просмотреть задания группы не удастся, так как созданную библиотеку (dll-файл) нельзя запускать на выполнение. Для тестирования полученной библиотеки удобно использовать программу PT4Demo.exe, расположенную в системном каталоге задачника Programming Taskbook (по умолчанию системный каталог задачника размещается в каталоге Program Files и имеет имя PT4).

Для того чтобы упростить и ускорить тестирование новых групп с применением программы PT4Demo, в версии 4.8 задачника для этой программы предусмотрены два *параметра командной строки*: –g<*название группы*>[#] и –n[–]<*номер задания*>. Перед описанием действий, связанных с тестированием группы заданий, опишем возможности, предоставляемые данными параметрами.

Если указан параметр –g, содержащий имя существующей группы и оканчивающийся символом #, то при запуске программы PT4Demo сразу создается и отображается на экране html-файл с описанием указанной группы, причем этот файл получает имя PT4<*название группы*>.html и сохраняется в рабочем каталоге задачника (если указанная группа заданий не найдена, то об этом выводится соответствующее сообщение). После отображения html-файла (или вывода сообщения о том, что группа не найдена) программа PT4Demo немедленно завершается.

Если в конце параметра –g не указан символ #, то при отсутствии указанной группы оба параметра игнорируются, а в случае, если группа существует, действия программы PT4Demo зависят от значения параметра –n. Если параметр –n отсутствует или равен 0, то в окне PT4Demo в списке групп выбирается указанная группа, а номер задания полагается равным 1. Если параметр –n больше нуля, то дополнительно выполняются следующие действия:

- номер задания полагается равным данному параметру (если параметр превышает количество заданий в группе, то номер задания полагается равным *максимальному* номеру задания в данной группе) и на экране *сразу отображается окно задачника с указанным заданием*,
- после закрытия окна задачника программа PT4Demo *немедленно завершает работу*.

Если перед номером в параметре –n указан знак «минус», то первое из перечисленных выше дополнительных действий выполняется, а второе — нет (то есть автоматического завершения работы программы PT4Demo при закрытии окна задачника не происходит).

Вернемся к нашему проекту — сводной группе. Для отображения ее заданий на экране необходимо определить главное приложение (host application), которое будет запускаться при запуске проекта, загружать dll-файл со сводной группой и отображать задания этой группы на экране. В среде Delphi для определения главного приложения следует выполнить команду меню «Run | Parameters...», перейти в появившемся окне на вкладку «Locals» и указать путь к главному приложению в поле «Host Application», а требуемые параметры — в поле «Parameters». Поле «Working Directory» следует оставить пустым; в этом случае рабочим каталогом будет считать-

ся тот каталог, который содержит откомпилированную библиотеку с группой заданий. Аналогичное окно настроек предусмотрено и в среде Lazarus; для его отображения надо выполнить команду меню «Run | Run Parameters...».

В качестве главного приложения укажем программу PT4Demo, введя в соответствующее поле ее полное имя вместе с путем (например, `C:\Program Files\PT4\PT4Demo.exe`). Как было отмечено выше при описании параметров командной строки, эта программа предусматривает два варианта просмотра содержимого группы: либо с использованием окна задачника, либо в виде html-страницы. На начальном этапе разработки группы следует использовать окно задачника, так как в нем отображается не только формулировка задания, но и образцы исходных и контрольных данных и связанные с ними комментарии. Поэтому в качестве параметров главного приложения укажем следующую строку: `-gMakerDemo -n999`. Благодаря этим параметрам при запуске программы PT4Demo в ней будет выбрана группа MakerDemo, причем сразу отобразится окно задачника с *последним* заданием данной группы (см. рис. 1). Напомним, что если значение параметра –n превышает число заданий в группе, то параметр полагается равным максимальному допустимому номеру задания.

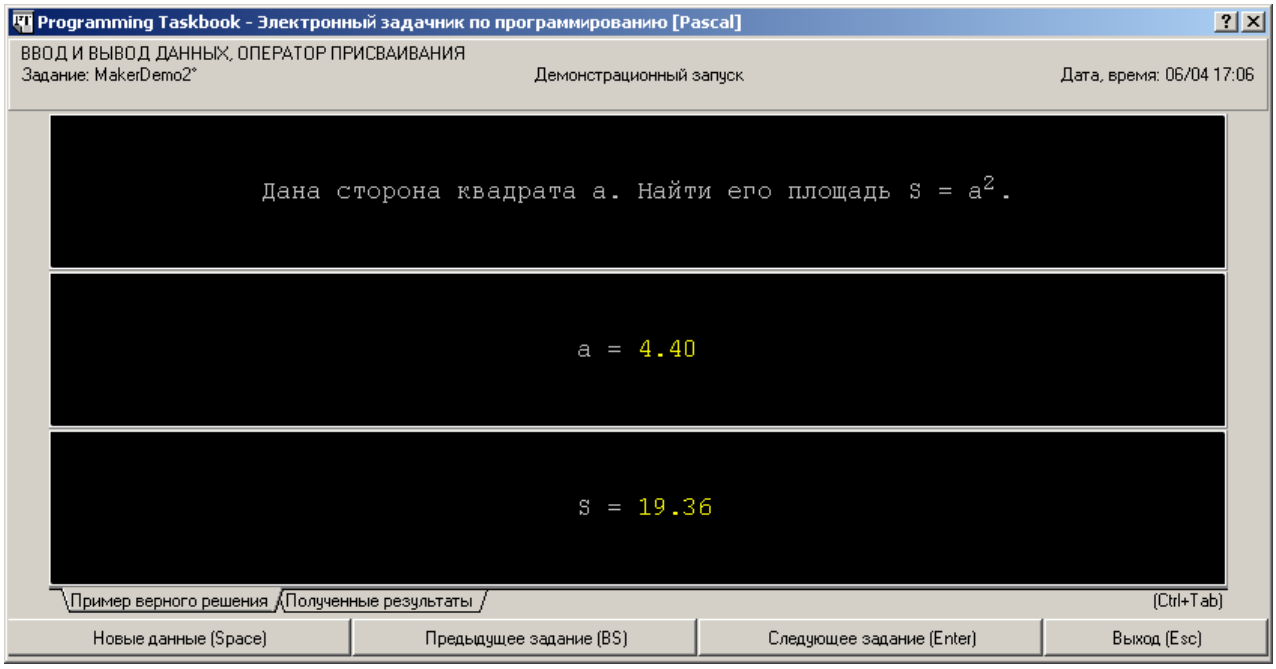


Рис. 1. Окно задачника в демо-режиме

В появившемся окне задачника можно просматривать все имеющиеся задания данной группы, а при его закрытии программа PT4Demo немедленно завершит работу, и мы вернемся в редактор кода среды Delphi.

Примечание. В качестве приложения, тестирующего созданный dll-проект с группой заданий, можно использовать программу, выполняющую одно из заданий этой группы, создав ее с помощью модуля PT4Load, однако перед этим необходимо сделать каталог с dll-проектом рабочим каталогом задачника (используя для этого команду «PT4 Setup» из меню «Programming Taskbook 4», расположенного в группе «Программы» Главного меню Windows) и хотя бы один раз выполнить компиляцию dll-проекта (чтобы в рабочем каталоге появился dll-файл с новой группой заданий). В нашем случае можно создать заготовку для выполнения задания MakerDemo2. Для возможности просмотра всех заданий группы следует вызвать задание MakerDemo2 в

демонстрационном режиме, указав после имени задания в процедуре Task символ ?: `Task('MakerDemo2?')`. При использовании демонстрационного режима можно не указывать номер задания; в этом случае при запуске программы на экране будет отображаться *последнее* задание данной группы.

5.3. Добавление описания группы и ее подгрупп

По тексту, расположенному выше названия задания MakerDemo2 (см. рис. 1), мы видим, что импортированные из группы Begin задания входят в подгруппу с заголовком «Ввод и вывод данных, оператор присваивания». В сводной группе MakerDemo мы можем добавить комментарий (*преамбулу*) как к самой группе, так и к любой имеющейся в ней подгруппе. Кроме того, мы можем *импортировать* преамбулу любой имеющейся группы или подгруппы. Для иллюстрации этих возможностей добавим в процедуру inittaskgroup новые операторы (их надо указать после вызова процедуры CreateGroup):

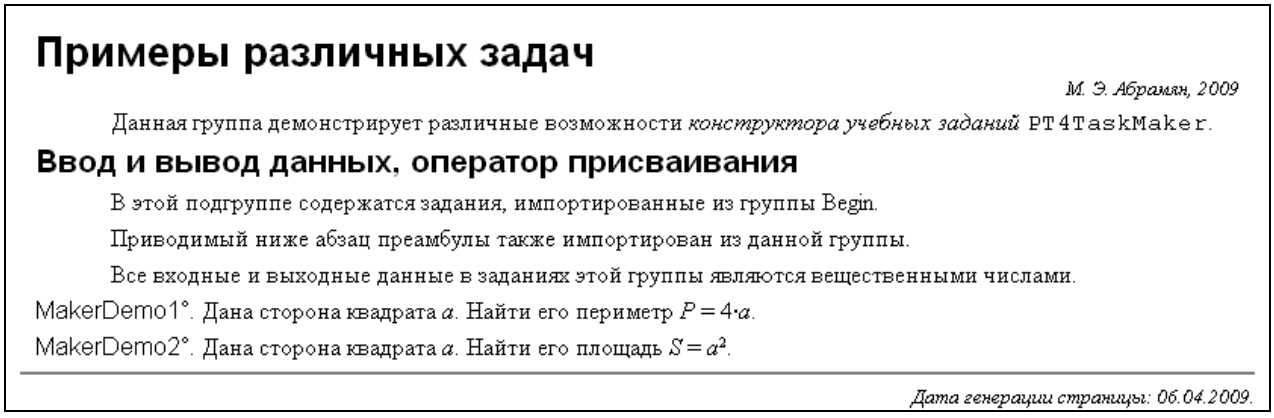
```
CommentText('Данная группа демонстрирует различные возможности');
CommentText('\Iконструктора учебных заданий\i \MPT4TaskMaker\m.');

Subgroup('Ввод и вывод данных, оператор присваивания');
CommentText('В этой подгруппе содержатся задания, импортированные');
CommentText('из группы Begin.\PПриводимый ниже абзац преамбулы');
CommentText('также импортирован из данной группы.\P');
UseComment('Begin');
```

Два первых вызова процедуры CommentText определяют текст преамбулы для группы MakerDemo. Обратите внимание на *управляющие последовательности*: пара последовательностей `\I` и `\i` выделяет *курсивный* фрагмент, а пара `\M` и `\m` выделяет фрагмент, в которым используется моноширинный шрифт. Последующий вызов процедуры Subgroup устанавливает режим определения преамбулы для подгруппы с указанным именем. В тексте этой преамбулы, который, как и текст преамбулы группы, определяется с помощью процедуры CommentText, используется управляющая последовательность `\P`, обеспечивающая переход к новому абзацу.

Наконец, последняя процедура (UseComment) импортирует преамбулу группы Begin в преамбулу нашей подгруппы «Ввод и вывод данных, оператор присваивания». Имеется также вариант процедуры UseComment, позволяющий импортировать преамбулу подгруппы; в этом варианте следует указать два параметра: имя группы и заголовок требуемой подгруппы, входящей в эту группу. Импортировать преамбулы подгрупп можно только для тех групп заданий, в которых имеется разделение на подгруппы (обычно это группы, содержащие большое количество заданий). В группе Begin деления на подгруппы нет, поэтому из нее можно импортировать только преамбулу самой группы.

Для того чтобы ознакомиться с результатом сделанных изменений, следует сгенерировать html-страницу с текстом группы MakerDemo. Для этого достаточно внести небольшое изменение в параметры командной строки главного приложения, а именно, следует дополнить параметр –g символом #, получив в результате строку `-gMakerDemo# -n999`. Теперь при запуске проекта PT4MakerDemo на экране вместо окна задачника с заданием MakerDemo2 появится html-браузер с описанием созданной группы (обратите внимание на последний абзац в описании подгруппы, который был импортирован из группы Begin):



Примеры различных задач

М. Э. Абрамян, 2009

Данная группа демонстрирует различные возможности *конструктора учебных заданий* PT4TaskMaker.

Ввод и вывод данных, оператор присваивания

В этой подгруппе содержатся задания, импортированные из группы Begin.

Приводимый ниже абзац преамбулы также импортирован из данной группы.

Все входные и выходные данные в заданиях этой группы являются вещественными числами.

MakerDemo1°. Дана сторона квадрата a. Найти его периметр $P = 4 \cdot a$.

MakerDemo2°. Дана сторона квадрата a. Найти его площадь $S = a^2$.

Дата генерации страницы: 06.04.2009.

Примечание. Вывести html-описание группы можно также, используя программу-заготовку, созданную для выполнения задания MakerDemo2 (см. примечание в предыдущем пункте). Для этого достаточно изменить параметр в процедуре Task, удалив в нем номер задания и добавив символ #: `Task('MakerDemo#')`. Заметим, что если указать в параметре символ #, не удаляя номер задания (например, `Task('MakerDemo2#')`), то в html-описание будет включено только задание с указанным номером. При этом будут также выведены комментарии ко всей группе и к той подгруппе, к которой относится выбранное задание. Для включения в html-страницу нескольких заданий (или групп заданий) достаточно для каждого из них вызвать процедуру Task с параметром, оканчивающимся символом #.

Завершая данный пункт, заметим, что *сводные группы*, то есть новые наборы уже имеющихся заданий, можно создавать более простым способои, не разрабатывая проекты в Borland Delphi или Free Pascal Lazarus. Достаточно подготовить текстовый файл с определением сводной группы и затем обработать его с помощью программы PTVarMaker «Конструктор вариантов», входящей в комплекс Teacher Pack for PT4 (см. п. 6). Файл с определением сводной группы в нашем случае должен содержать следующий текст:

```
=MakerDemo
=Примеры различных задач
=qwqfsdf13dfttd

-
Данная группа демонстрирует различные возможности
\Iконструктора учебных заданий\i \MPT4TaskMaker\m.
- Ввод и вывод данных, оператор присваивания
В этой подгруппе содержатся задания,
импортированные из группы Begin.
\PПриводимый ниже абзац преамбулы
также импортирован из данной группы.\P
* Begin

Begin 1-2
```

Обработав данный текст, мы получим файл PT4MakerDemo.dll, содержащий сводную группу заданий, а также файл PT4MakerDemo.html с ее описанием. Заметим, что в конструкторе вариантов предусмотрены средства, позволяющие немедленно просмотреть созданную группу заданий в окне задачника, а также вывести на экран ее html-описание (см. п. 6.3).

5.4. Добавление нового задания

Добавим к нашей группе новое задание. Фактически это задание будет дублировать задание Begin3, однако вместо импортирования этого задания мы разработаем его самостоятельно. Все действия по созданию нового задания удобно реализовать во вспомогательной процедуре, которую можно назвать MakerDemo3 (таким образом, название процедуры будет соответствовать имени создаваемого задания, хотя это и не является обязательным):

```
procedure MakerDemo3;
var
  a, b: real;
begin
  CreateTask('Ввод и вывод данных, оператор присваивания');
  TaskText('Даны стороны прямоугольника~{a} и~{b}.', 0, 2);
  TaskText('Найти его площадь {S}~=~{a}\*{b} и периметр ' +
    '{P}~=~2\*({a}\;+\;{b}).', 0, 4);
  a := (1 + Random(100)) / 10;
  b := (1 + Random(100)) / 10;
  DataR('a = ', a, xLeft, 3, 4);
  DataR('b = ', b, xRight, 3, 4);
  ResultR('S = ', a * b, 0, 2, 4);
  ResultR('P = ', 2 * (a + b), 0, 4, 4);
  SetTestCount(3);
end;
```

Процедура MakerDemo3 включает все основные действия, используемые при формировании нового задания:

- *инициализацию* нового задания (процедура CreateTask; мы указали в этой процедуре, что данное задание должно входить в подгруппу «Ввод и вывод данных, оператор присваивания», то есть в ту же подгруппу, что и два предыдущих задания);
- определение его *формулировки* (процедуры TaskText; обратите внимание на используемые в этих процедурах управляющие последовательности);
- определение исходных (процедуры DataR) и результирующих данных (процедуры ResultR);
- указание количества успешных тестовых запусков программы учащегося, достаточных для регистрации задания как выполненного (процедура SetTestCount; для нашего простого задания достаточно трех *проведенных подряд* успешных тестовых запусков).

Необходимо также включить вызов созданной процедуры в основную процедуру группы MakerDemo, связав его с номером 3:

```
procedure InitTask(num: integer);
begin
  case num of
  1..2: UseTask('Begin', num);
  3: MakerDemo3;
  end;
end;
```

Наконец, следует откорректировать число заданий в вызове процедуры CreateGroup, изменив его на 3.

Запустив проект на выполнение, мы увидим в html-описании группы MakerDemo формулировки трех заданий, а удалив из списка параметров главного приложения

символ # (в результате список параметров примет вид `-gMakerDemo -n999`) и повторно запустив проект на выполнение, мы увидим окно задачника с загруженным заданием MakerDemo3. Заметим, что при последующих запусках проекта мы будем получать в окне задачника различные исходные данные; это связано с тем, что в процедуре CreateTask автоматически вызывается процедура Randomize.

5.5. Добавление заданий на обработку двумерных массивов и символьных строк

Добавим к группе MakerDemo еще два задания: первое из них дублирует задание Matrix7 (подгруппа «Двумерные массивы (матрицы): вывод элементов»), а второе не имеет полного аналога в группе String, однако может быть отнесено к ее первой подгруппе: «Символы и строки: основные операции». Реализуем эти задания в процедурах MakerDemo4 и MakerDemo5:

```
procedure MakerDemo4;
var
  m, n, i, j, k: integer;
  a: array [1..5, 1..8] of real;
begin
  CreateTask('Двумерные массивы (матрицы): вывод элементов');
  TaskText('Дана матрица размера~{M}\;\x\;{N} ' +
    'и\ целое число~{K} (1~\l~{K}~\l~{M}).', 0, 2);
  TaskText('Вывести элементы {K}-й строки данной матрицы.',
    0, 4);
  m := 2 + Random(4);
  n := 4 + Random(5);
  k := 1;
  if m = 5 then k := 0;
  DataN('M = ', m, 3, 1, 1);
  DataN('N = ', n, 10, 1, 1);
  for i := 1 to M do
    for j := 1 to N do
    begin
      a[i, j] := 9.98 * Random;
      DataR('', a[i,j], Center(j, n, 4, 2), i + k, 4);
    end;
  k := 1 + Random(m);
  dataN('K = ', k, 68, 5, 1);
  for j := 1 to n do
    ResultR('', a[k, j], Center(j, n, 4, 2), 3, 4);
end;

procedure MakerDemo5;
var
  s: string;
begin
  CreateTask('Символы и строки: основные операции');
  TaskText('Дана строка~{S}.', 0, 2);
  TaskText('Вывести ее первый и последний символ.', 0, 4);
  s := WordSample(Random(WordCount));
  DataS('S = ', s, 0, 3);
  ResultC('Первый символ: ', s[1], xLeft, 3);
  ResultC('Последний символ: ', s[length(s)], xRight, 3);
  SetTestCount(4);
end;
```

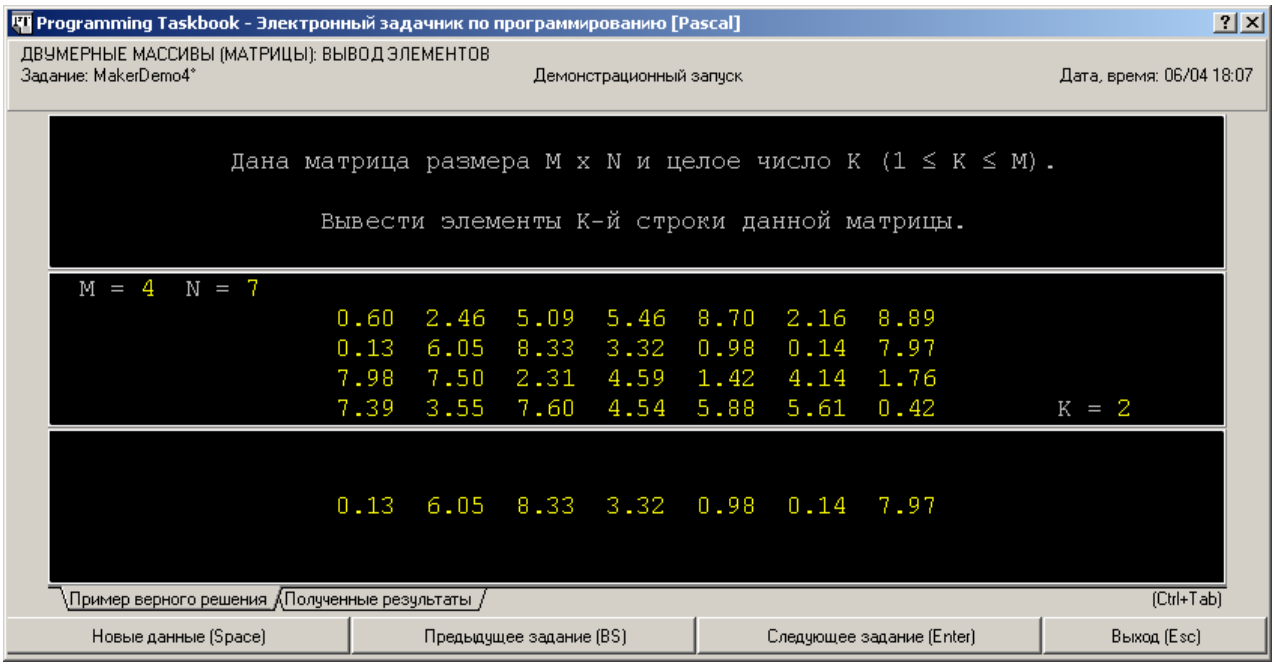


Рис. 2. Задание на обработку двумерного массива (матрицы)

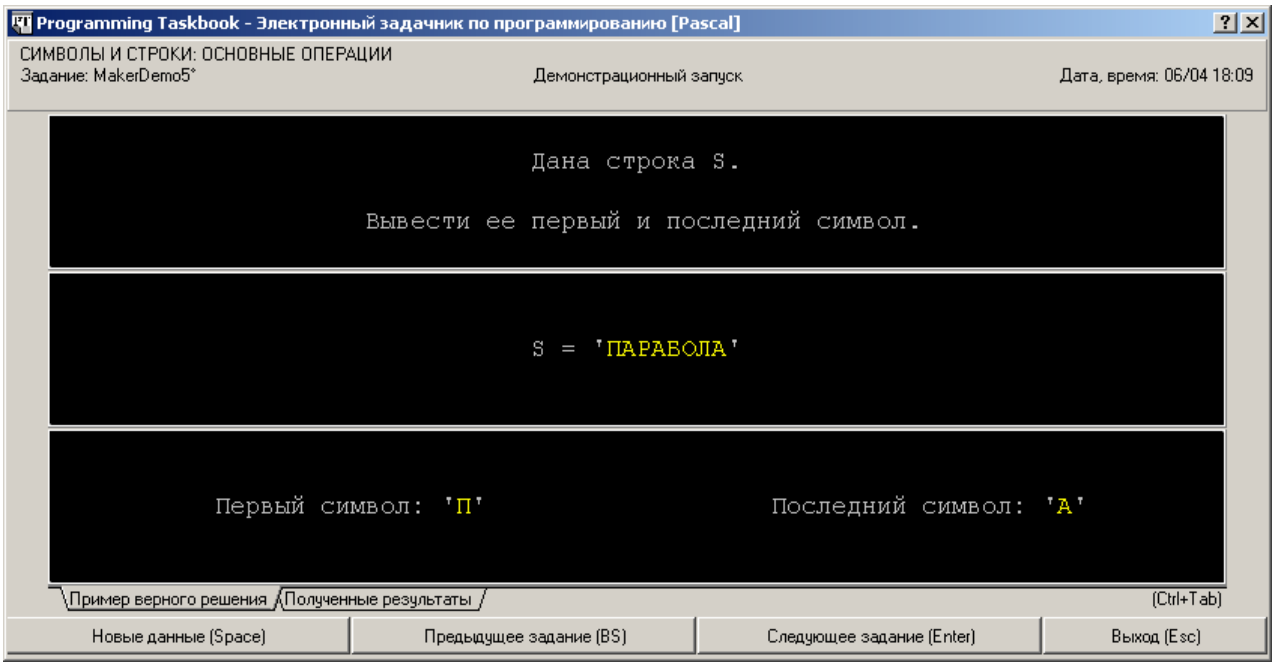


Рис. 3. Задание на обработку текстовых строк

Обратите внимание на использование вспомогательной функции Center для центрирования строк матрицы в области исходных и результирующих данных: каждый элемент матрицы занимает 4 экранные позиции, а между элементами размещается по два пробела. В процедуре MakerDemo4 не вызывается процедура SetTestCount; в этом случае число успешных тестов, необходимых для регистрации задания как выполненного, по умолчанию полагается равным 5.

В процедуре MakerDemo5 для получения исходных символьных строк используются функции WordCount и WordSample (см. п. 2.7). С помощью этих функций можно получать различные варианты русских слов. Заметим, что в конструкторе PT4TaskMaker имеются также функции EnWordCount и EnWordSample, с помощью которых можно получать варианты английских слов.

Осталось изменить количество заданий в вызове процедуры CreateGroup на 5 и включить вызовы новых процедур в основную процедуру группы InitTask:

```
procedure InitTask(num: integer);
begin
  case num of
  1..2: UseTask('Begin', num);
  3: MakerDemo3;
  4: MakerDemo4;
  5: MakerDemo5;
  end;
end;
```

Приведем вид окна задачника для новых заданий (см. рис. 2 и 3).

5.6. Добавление заданий на обработку файлов

Добавим к группе MakerDemo еще два задания: первое из них дублирует задание File63 (подгруппа «Символьные и строковые файлы»), а второе — задание Text16 (подгруппа «Текстовые файлы: основные операции»). Реализуем эти задания в процедурах MakerDemo6 и MakerDemo7:

```
function FileName(Len: integer): string;
const
  c = '0123456789abcdefghijklmnopqrstuvwxyz';
var
  i: integer;
begin
  result := '';
  for i := 1 to Len do
    result := result + c[Random(Length(c))+1];
end;

procedure MakerDemo6;
var
  k, i, j, jmax: integer;
  s1, s2, s3: string;
  fs1: file of ShortString;
  fs2: file of ShortString;
  fc3: file of char;
  s: ShortString;
  c: char;
begin
  CreateTask('Символьные и строковые файлы');
  TaskText('Дано целое число~{K} (>\,0) и строковый файл.', 0, 1);
  TaskText('Создать два новых файла: строковый, содержащий ' +
    'первые {K}~символов', 0, 2);
  TaskText('каждой строки исходного файла, и символьный, ' +
    'содержащий {K}-й символ', 0, 3);
  TaskText('каждой строки (если длина строки меньше~{K}, ' +
    'то в строковый файл', 0, 4);
  TaskText('записывается вся строка, а в символьный файл ' +
    'записывается пробел).', 0, 5);
  s1 := '1' + FileName(5) + '.tst';
  s2 := '2' + FileName(5) + '.tst';
  s3 := '3' + FileName(5) + '.tst';
  Assign(fs1, s1);
  Rewrite(fs1);
  Assign(fs2, s2);
```

```
  Rewrite(fs2);
  Assign(fc3, s3);
  Rewrite(fc3);
  k := 2 + Random(10);
  jmax := 0;
  for i := 1 to 10 + Random(20) do
  begin
    j := 2 + Random(15);
    if jmax < j then
      jmax := j;
    s := FileName(j);
    write(fs1, s);
    if j >= k then
      c := s[k]
    else
      c := ' ';
    write(fc3, c);
    s := copy(s, 1, k);
    write(fs2,s);
  end;
  Close(fs1);
  Close(fs2);
  Close(fc3);
  DataN('K =', k, 0, 1, 2);
  DataS('Имя исходного файла: ', s1, 3, 2);
  DataS('Имя результирующего строкового файла:  ', s2, 3, 4);
  DataS('Имя результирующего символьного файла: ', s3, 3, 5);
  DataComment('Содержимое исходного файла:', xRight, 2);
  DataFileS(s1, 3, jmax + 3);
  ResultComment('Содержимое результирующего строкового файла:',
    0, 2);
  ResultComment('Содержимое результирующего символьного файла:',
    0, 4);
  ResultFileS(s2, 3, k + 3);
  ResultFileC(s3, 5, 4);
end;

procedure MakerDemo7;
var
  p: integer;
  s, s1, s2: string;
  t1, t2: text;
begin
  CreateTask('Текстовые файлы: основные операции');
  TaskText('Дан текстовый файл.', 0, 2);
  TaskText('Удалить из него все пустые строки.', 0, 4);
  s1 := FileName(6) + '.tst';
  s2 := '#' + FileName(6) + '.tst';
  s := TextSample(Random(TextCount));
  Assign(t2, s2);
  Rewrite(t2);
  Assign(t1, s1);
  Rewrite(t1);
  writeln(t2, s);
  Close(t2);
  p := Pos(#13#10#13#10, s);
  while p <> 0 do
```

```
    begin
      Delete(s, p, 2);
      p := Pos(#13#10#13#10, s);
    end;
    writeln(t1, s);
    Close(t1);
    ResultFileT(s1, 1, 5);
    Rename(t2, s1);
    DataFileT(s1, 2, 5);
    DataS('Имя файла: ', s1, 0, 1);
    SetTestCount(3);
end;
```

При реализации этих заданий используется вспомогательная функция FileName(Len), позволяющая создать случайное имя файла длины Len (без расширения). Имя файла при этом будет содержать только цифры и строчные (маленькие) латинские буквы.

Имена файлов, полученные с помощью функции FileName, дополняются расширением .tst (заметим, что в базовых группах File, Text и Param это расширение используется в именах всех исходных и результирующих файлов).

Функция FileName используется также для генерации элементов строкового файла в процедуре MakerDemo6.

Для того чтобы предотвратить возможность случайного совпадения имен файлов, в процедуре MakerDemo6 к созданным именам добавляются префиксы: `1` для первого файла, `2` для второго, `3` для третьего. В процедуре MakerDemo7 имя временного файла дополняется префиксом `#`, что также гарантирует его отличие от имени основного файла задания.

При реализации задания на обработку текстовых файлов для генерации содержимого файла используются функции TextCount и TextSample (см. п. 2.7). Строка, возвращаемая функцией TextSample, представляет собой текст, содержащий *маркеры конца строки* — символы #13#10. Указанные символы *разделяют соседние строки текста* (в конце текста маркер конца строки не указывается). Благодаря наличию маркеров конца строки полученный текст можно записать в текстовый файл с помощью единственной процедуры writeln, которая, кроме записи текста, обеспечивает добавление маркера конца строки в конец файла.

После разработки новых заданий необходимо изменить количество заданий в вызове процедуры CreateGroup на 7 и включить вызовы новых процедур в основную процедуру группы InitTask:

```
procedure InitTask(num: integer);
begin
  case num of
  1..2: UseTask('Begin', num);
  3: MakerDemo3;
  4: MakerDemo4;
  5: MakerDemo5;
  6: MakerDemo6;
  7: MakerDemo7;
  end;
end;
```

Приведем вид окна задачника для новых заданий (см. рис. 4 и 5).

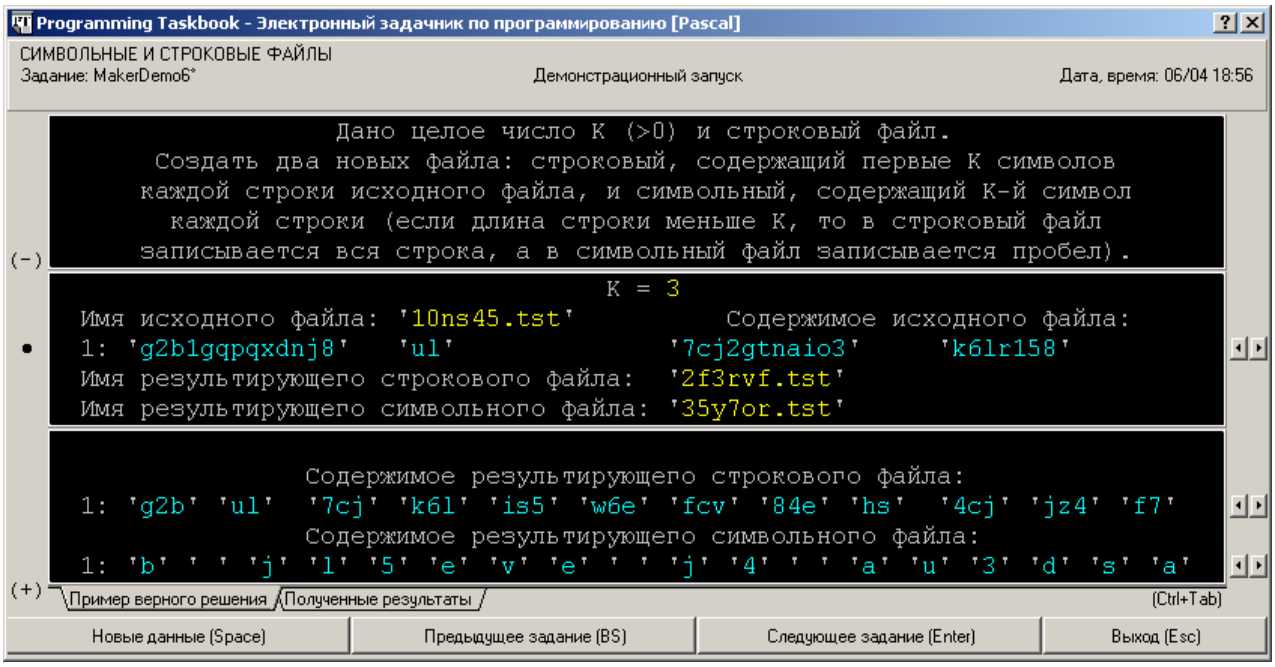


Рис. 4. Задание на обработку двоичных файлов

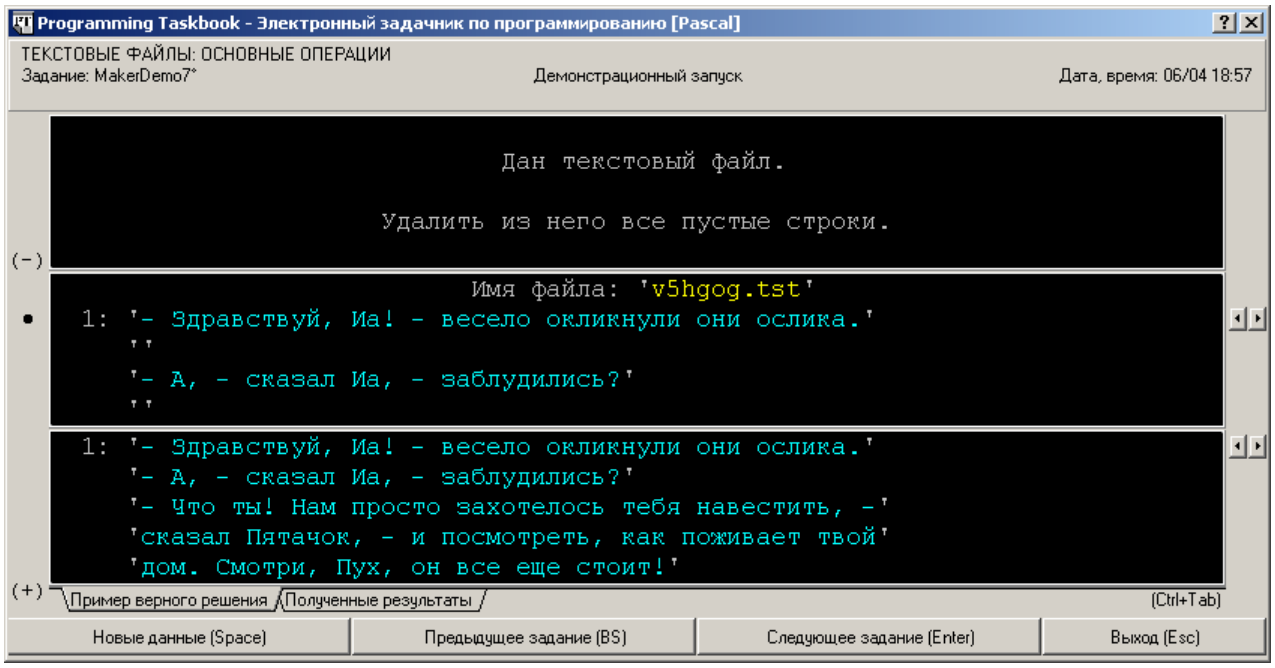


Рис. 5. Задание на обработку текстовых файлов

5.7. Добавление заданий на обработку динамических структур данных

Наконец, добавим в нашу группу задание, посвященное обработке динамических структур данных, причем представим его в двух вариантах: традиционном, основанном на использовании записей типа TNode и связанных с ними указателей типа PNode, и «объектном», характерном для .NET-языков. Следует подчеркнуть, что при *разработке* как традиционного, так и объектного варианта заданий на динамические структуры надо использовать типы TNode и PNode и связанные с ними процедуры конструктора учебных заданий. В то же время, при *выполнении* объектного варианта

задания на одном из .NET-языков требуется использовать объекты типа Node (которые при разработке задания не применяются).

Задание, которое мы реализуем, дублирует задание Dynamic30, посвященное преобразованию односвязного списка в двусвязный (подгруппа «Динамические структуры данных: двусвязный список»). Оформим два варианта этого задания в виде процедур MakerDemo8 и MakerDemo8Net:

```
var WrongNode: TNode;

procedure MakerDemo8Data;
var
  i, n: integer;
  p, p1, p2: PNode;
begin
  if Random(4) = 0 then
    n := 1
  else
    n := 2 + Random(8);
  new(p1);
  p1^.Data := Random(90) + 10;
  p1^.Prev := nil;
  p2 := p1;
  for i := 2 to n do
  begin
    new(p);
    p^.Data := Random(90) + 10;
    p^.Prev := p2;
    p2^.Next := p;
    p2 := p;
  end;
  p2^.Next := nil;
  SetPointer(1, p1);
  SetPointer(2, p2);
  ResultP('Последний элемент: ', 2, 0, 2);
  ResultList(1, 0, 3);
  ShowPointer(2);
  DataP('', 1, 0, 2);
  p := p1;
  for i := 1 to n do
  begin
    p^.prev := @WrongNode;
    p := p^.Next;
  end;
  DataList(1, 0, 3);
  ShowPointer(1);
end;

procedure MakerDemo8;
begin
  CreateTask('Динамические структуры данных: двусвязный список');
  TaskText('Дан указатель~{P}_1 на начало непустой цепочки ' +
    'элементов-записей типа TNode,', 0, 1);
  TaskText('связанных между собой с помощью поля Next. Используя ' +
    'поле Prev записи TNode,', 0, 2);
  TaskText('преобразовать исходную (\Iодносвязную\i) цепочку ' +
    'в  \Iдвусвязную\i, в которой каждый', 0, 3);
  TaskText('элемент связан не только с последующим элементом ' +
```

```
    '(с помощью поля Next),', 0, 4);
  TaskText('но и с предыдущим (с помощью поля Prev). Поле Prev ' +
    'первого элемента положить', 0, 5);
  TaskText('равным~\N. Вывести указатель на последний элемент ' +
    'преобразованной цепочки.', 0, 0);
  MakerDemo8Data;
end;

procedure MakerDemo8Net;
begin
  CreateTask('Динамические структуры данных: двусвязный список');
  TaskText('Дана ссылка~{A}_1 на начало непустой цепочки ' +
    'элементов-объектов типа Node,', 0, 1);
  TaskText('связанных между собой с помощью своих свойств Next. ' +
    'Используя свойства Prev', 0, 2);
  TaskText('данных объектов, преобразовать исходную ' +
    '(\Iодносвязную\i) цепочку в \Iдвусвязную\i,', 0, 3);
  TaskText('в которой каждый элемент связан не только ' +
    'с последующим элементом (с помощью', 0, 4);
  TaskText('свойства Next), но и с предыдущим (с помощью ' +
    'свойства Prev). Свойство Prev', 0, 5);
  TaskText('первого элемента положить равным~\O. Вывести ' +
    'ссылку~{A}_2 на последний', 0, 0);
  TaskText('элемент преобразованной цепочки.', 0, 0);
  SetObjectStyle;
  MakerDemo8Data;
end;
```

Анализируя приведенные варианты процедур, легко заметить, что они отличаются лишь деталями формулировки задания. Алгоритмы генерации исходных и контрольных данных для традиционного и объектного вариантов совпадают, поэтому они выделены в отдельную вспомогательную процедуру MakerDemo8Data. В то же время *представления* динамических структур и связанных с ними указателей или объектов будут отличаться (см. рисунки, приведенные ниже). Необходимые корректировки в представлении динамических структур выполняются задачником автоматически, с учетом используемого языка программирования.

Однако для языка PascalABC.NET требуемую настройку необходимо выполнить явно, так как в нем можно использовать оба варианта представления динамических структур: традиционный (как для обычного Паскаля в системах Delphi и Free Pascal Lazarus) и объектный (как в языках C# и Visual Basic .NET). Для того чтобы представление динамических данных при выполнении задания в среде PascalABC.NET соответствовало объектному варианту, следует в начале процедуры, реализующей задание (перед вызовом любых процедур, связанных с указателями и динамическими структурами), вызвать специальную процедуру без параметров SetObjectStyle. Для остальных языков данная процедура не выполняет никаких действий.

Обратите внимание на возможность использования в формулировке задания более 5 экранных строк. Строки, которые не умещаются в области формулировки задания, следует добавлять к заданию процедурой TaskText, указывая в качестве последнего параметра процедуры число 0. При наличии подобных строк в окне задачника слева от области формулировки появятся кнопки, обеспечивающие прокрутку формулировки задания (кроме этих кнопок можно также использовать стандартные клавиши, в частности, клавиши со стрелками).

Для того чтобы имя нулевого указателя (или объекта) соответствовало используемому языку программирования, в формулировке задания применяются управляющие последовательности `\N` (имя нулевого указателя) и `\O` (имя нулевого объекта). Для языка PascalABC.NET обе эти последовательности генерируют текст `nil`.

Достаточно часто алгоритмы, разработанные учащимися для обработки динамических структур данных, дают неверные результаты в случае особых (хотя и допустимых) структур, например, состоящих только из одного элемента. Поэтому желательно предусмотреть появление подобных структур в тестовых наборах исходных данных. В наших заданиях исходный список, состоящий из *одного* элемента, будет предлагаться программе учащегося в среднем один раз при каждых четырех тестовых испытаниях.

При формировании односвязной структуры неиспользуемые поля Prev для каждого элемента структуры следует положить равными адресу «фиктивного» элемента (в нашем случае — переменной WrongNode), не связанного с данной структурой. Заметим, что для всех элементов, кроме первого, можно было бы положить значения поля Prev равными nil, однако это не подходит для первого элемента: если поле Prev первого элемента будет равно nil, то слева от него будет выведен «лишний» (в данной ситуации) текст `nil<`.

Характерной особенностью разработки заданий на динамические структуры является *обратный порядок* создания этих структур: вначале создаются *контрольные* структуры (которые сразу передаются в задачник), а затем они преобразуются в соответствующие *исходные* структуры, которые должны не только передаваться в задачник, но и *оставаться в памяти*, чтобы в дальнейшем их можно было использовать в программе учащегося, выполняющей это задание.

Если в группу включаются задания на динамические структуры, то необходимо *анализировать текущий язык программирования*, используемый задачником. Это обусловлено двумя причинами:

- имеется язык, не поддерживающий работу с динамическими структурами (Visual Basic);
- в языках платформы .NET необходимо использовать «объектный» стиль формулировок вместо традиционного стиля, основанного на указателях и применяемого для языков Pascal и C++.

Кроме того, следует определиться с выбором стиля для языка PascalABC.NET, поскольку в нем можно использовать как стиль указателей, так и стиль объектов. Можно, например, включить в группу заданий для языка PascalABC.NET оба варианта каждого задания.

Отмеченные обстоятельства приводят к тому, что для разных языков программирования создаваемая группа может содержать разное число заданий и, кроме того, для этих заданий будут использоваться разные инициализирующие процедуры.

С учетом этих замечаний изменим основную процедуру группы InitTask следующим образом:

```
procedure InitTask(num: integer); stdcall;
begin
  case num of
  1..2: UseTask('Begin',num);
  3: MakerDemo3;
  4: MakerDemo4;
  5: MakerDemo5;
```

```
    6: MakerDemo6;
    7: MakerDemo7;
    8: if CurrentLanguage and lgWithPointers <> 0 then
         MakerDemo8
       else
         MakerDemo8net;
    9: if CurrentLanguage = lgPascalABCNET then
         MakerDemo8Net;
    end;
  end;
```

В этой процедуре используется функция CurrentLanguage, позволяющая определить текущий язык программирования, используемый задачником. Если текущий язык относится к категории языков, поддерживающих указатели (в том числе PascalABC.NET), то в качестве задания номер 8 вызывается процедура MakerDemo8, в которой задание формулируется в терминах указателей. В противном случае вызывается вариант задания для .NET-языков, использующий объектную терминологию. Наконец, если текущим языком является PascalABC.NET, то для него будет доступно дополнительное задание номер 9, представляющее собой «объектный» вариант задания номер 8.

Функцию CurrentLanguage потребуется использовать и в начале процедуры inittaskgroup для того, чтобы правильно определить количество заданий в группе для разных языков программирования (обратите внимание на то, что теперь в качестве предпоследнего параметра процедуры CreateGroup используется переменная n):

```
var
  n: integer;
begin
  case CurrentLanguage of
  lgVB: n := 7;
  lgPascalABCNET: n := 9;
  else n := 8;
  end;
  CreateGroup('MakerDemo', 'Примеры различных задач',
    'М. Э. Абрамян, 2009', 'qwqfsdf13dfttd', n, InitTask);
```

Новые задания следует протестировать для различных языков программирования. Для этого можно воспользоваться еще одной новой возможностью программы PT4Demo, появившейся в версии 4.8: выбором языка программирования с помощью контекстного меню. В данной ситуации следует изменить список параметров главного приложения, положив параметр –n равным 0: `-gMakerDemo -n0`. Благодаря этому изменению при запуске проекта станет доступно окно программы PT4Demo, в котором можно будет изменить язык программирования (для этого достаточно нажать правую кнопку мыши и выбрать требуемый язык из появившегося контекстного меню). Используя программу PT4Demo, не удастся проверить группу лишь для языка PascalABC.NET. Подобную проверку можно выполнить только в среде PascalABC.NET, выбрав в ней в качестве текущего тот каталог, в котором располагается dll-файл с созданной группой.

Приведем вид окна задачника для новых заданий (см. рис. 6 и 7). Обратите внимание на кнопки, расположенные справа от формулировки задания и обеспечивающие ее прокрутку.

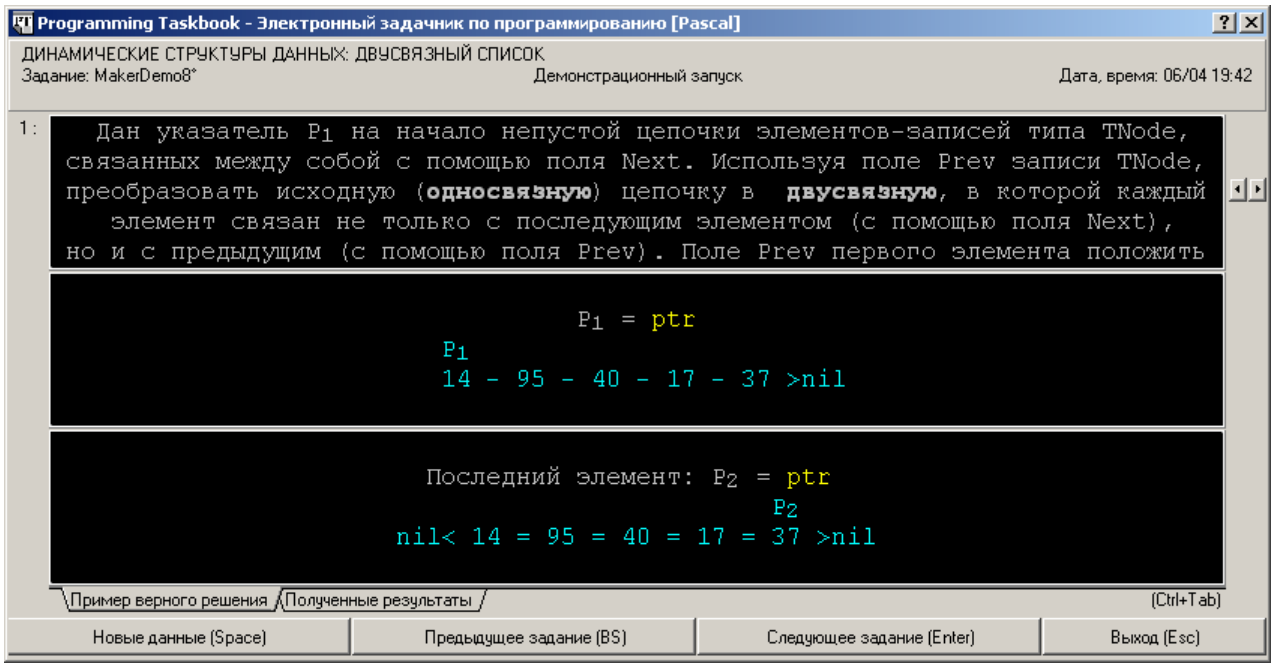


Рис. 6. Задание на обработку динамических структур для языка Pascal

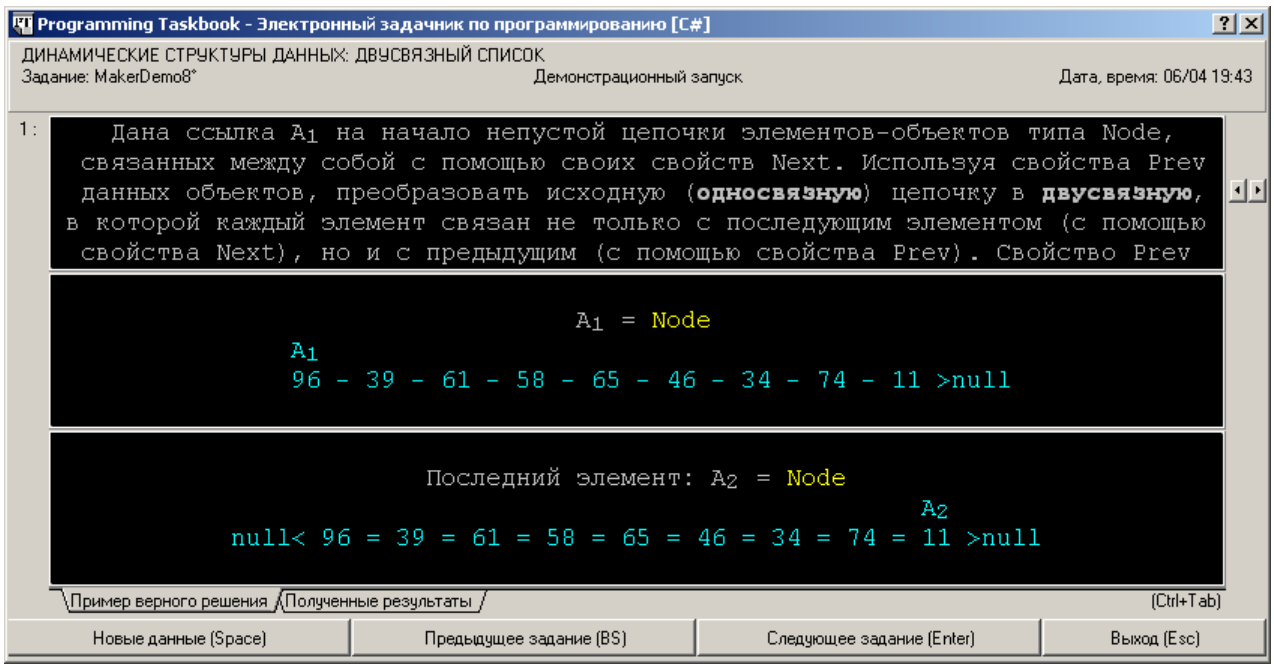


Рис. 7. Задание на обработку динамических структур для языка C#

Завершая оформление модуля PT4MakerDemo, добавим комментарии к новым подгруппам заданий (указанные операторы следует поместить в конец процедуры inittaskgroup):

```
Subgroup('Двумерные массивы (матрицы): вывод элементов');
CommentText('Данное задание дублирует задание Matrix7.');

Subgroup('Символьные и строковые файлы');
CommentText('Данное задание дублирует задание File63.');
CommentText('Оно демонстрирует особенности, связанные с двоичными');
CommentText('\Iстроковыми\i файлами.');

Subgroup('Текстовые файлы: основные операции');
CommentText('Данное задание дублирует задание Text16.');
```

```
Subgroup('Динамические структуры данных: двусвязный список');
CommentText('Данное задание дублирует задание Dynamic30.');
CommentText('\PВид задания различается для языков, поддерживающих');
CommentText('указатели (Pascal, C++), и .NET-языков. Для языка');
CommentText('Visual Basic это задание недоступно. В системе');
CommentText('PascalABC.NET доступны оба варианта задания.');
```

Приведем заключительную часть html-страницы с описанием данной группы после установки C# в качестве текущего языка:

Двумерные массивы (матрицы): вывод элементов

Данное задание дублирует задание Matrix7.

MakerDemo4°. Дана матрица размера $M \times N$ и целое число K ($1 \le K \le M$). Вывести элементы K-й строки данной матрицы.

Символы и строки: основные операции

MakerDemo5°. Дана строка S. Вывести ее первый и последний символ.

Символьные и строковые файлы

Данное задание дублирует задание File63. Оно демонстрирует особенности, связанные с двоичными *строковыми* файлами.

MakerDemo6°. Дано целое число K (> 0) и строковый файл. Создать два новых файла: строковый, содержащий первые K символов каждой строки исходного файла, и символьный, содержащий K-й символ каждой строки (если длина строки меньше K, то в строковый файл записывается вся строка, а в символьный файл записывается пробел).

Текстовые файлы: основные операции

Данное задание дублирует задание Text16.

MakerDemo7°. Дан текстовый файл. Удалить из него все пустые строки.

Динамические структуры данных: двусвязный список

Данное задание дублирует задание Dynamic30.

Вид задания различается для языков, поддерживающих указатели (Pascal, C++), и .NET-языков. Для языка Visual Basic это задание недоступно. В системе PascalABC.NET доступны оба варианта задания.

MakerDemo8°. Дана ссылка A_1 на начало непустой цепочки элементов-объектов типа Node, связанных между собой с помощью своих свойств Next. Используя свойства Prev данных объектов, преобразовать исходную *(односвязную)* цепочку в *двусвязную*, в которой каждый элемент связан не только с последующим элементом (с помощью свойства Next), но и с предыдущим (с помощью свойства Prev). Свойство Prev первого элемента положить равным null. Вывести ссылку A_2 на последний элемент преобразованной цепочки.

Дата генерации страницы: 06.04.2009.

5.8. Особенности оформления группы заданий в виде модуля системы PascalABC.NET

При добавлении новой группы заданий в задачник, интегрированный в систему PascalABC.NET, вместо dll-файла можно использовать pcu-файл, то есть откомпилированный модуль языка PascalABC.NET. Однако в этом случае созданную группу нельзя будет включить в универсальный вариант задачника.

Для оформления описанной выше группы MakerDemo в виде pcu-файла достаточно изменить файл PT4MakerDemo.dpr следующим образом:

- замените расширение данного файла на .pas;
- в первой строке файла (`library PT4MakerDemo;`) замените слово library на unit;
- в заголовке процедуры InitTask (`procedure InitTask(num: integer); stdcall;`) удалите модификатор stdcall вместе с завершающим символом «;»;

- в конец процедуры inittaskgroup добавьте оператор
  `RegisterGroup('PT4MakerDemo');`
- удалите строку, начинающуюся словом `exports` и добавьте в завершающий блок `begin-end` вызов функции inittaskgroup.

Завершающий фрагмент файла (после описания функции inittaskgroup) должен иметь вид:

```
begin
  inittaskgroup;
end.
```

Таким образом, основное отличие нового варианта состоит в удалении раздела exports и включении вызова процедуры inittaskgroup в секцию инициализации модуля PT4MakerDemo.

Вызов процедуры RegisterGroup, добавленный в конец процедуры inittaskgroup, позволяет зарегистрировать созданную группа заданий в модуле PT4Load, интегрированном в систему PascalABC.NET (см. п. 2.5).

Для того чтобы созданная группа была доступна для использования, необходимо, чтобы файл PT4MakerDemo.pcu размещался либо в рабочем каталоге учащегося, либо в подкаталоге Lib системного каталога PascalABC.NET. При этом в заготовке для выполнения заданий из этой группы потребуется дополнительно указать имя этого модуля, например:

```
uses PT4, PT4MakerDemo;
begin
  Task('MakerDemo2');
end.
```

После первого запуска программы, подключившей модуль PT4MakerDemo (то есть после того, как хотя бы один раз будет успешно выполнена функция RegisterGroup), создавать подобную заготовку можно будет с помощью программного модуля PT4Load.

В отличие от групп заданий, реализованных в виде dll-файлов, группы в pcu-файлах не распознаются программным модулем PT4Demo, интегрированным в систему PascalABC.NET. Следует также иметь в виду, что если группа заданий с одним и тем же именем реализована и в виде dll-файла и в виде pcu-файла, то программный модуль PT4Load использует dll-вариант группы как более универсальный (при этом к программе учащегося подключается только модуль PT4). В подобной ситуации для выбора pcu-варианта группы достаточно указать имя соответствующего pcu-модуля в списке uses (например, `uses PT4, PT4MakerDemo;`).

6. Сводные группы учебных заданий

6.1. Общее описание

С помощью конструктора учебных заданий преподаватель может не только разрабатывать новые задания, но и перегруппировывать задания имеющихся групп, создавая *сводные группы заданий*. Сводные группы могут оказаться полезными при подготовке индивидуальных наборов заданий или наборов, предлагаемых на проверочных работах. Помимо возможности изменения порядка следования заданий и объединения в одну группу заданий из нескольких базовых групп (или базовой группы и дополнительных групп, посвященных этой же теме), сводные группы позволяют

«скрыть происхождение» заданий (поскольку задания в них имеют другие имена) и тем самым затрудняют для недобросовестных учащихся использование готовых решений (по крайней мере, на проверочных работах, когда время на выполнение заданий ограничено). Кроме того, при составлении сводной группы преподаватель имеет возможность добавлять свои комментарии, связанные как с группой в целом, так и с ее подгруппами, то есть с наборами родственных заданий, объединенных общим подзаголовком.

Сводные группы заданий, в отличие от групп, содержащих новые задания, можно разрабатывать, не создавая для них специального программного кода, который затем должен компилироваться в dll-файл. Достаточно лишь подготовить *текстовый файл*, включающий всю необходимую информацию о создаваемой сводной группе, и обработать его в программе PTVarMaker («Конструктор вариантов»), входящей в комплекс Teacher Pack for PT4 [5]. В результате будет автоматически сгенерирован требуемый dll-файл, а также вспомогательный html-файл, содержащий полный текст созданной сводной группы. После создания dll-файла со сводной группой его достаточно скопировать в подкаталог Lib системного каталога задачника Programming Taskbook; в результате эта сводная группа будет интегрирована в задачник и станет доступной для выполнения наряду с базовыми группами заданий. При этом информация о сводной группе появится в программных модулях PT4Demo и PT4Load, что позволит просматривать задания данной группы в демонстрационном режиме и создавать программы-заготовки для выполнения требуемых заданий.

Можно также скопировать dll-файл со сводной группой в рабочий каталог учащегося; в этом случае задания этой группы будут доступны только для данного учащегося. Заметим, что для копирования файлов сводных групп в каталоги учащихся удобно использовать команду «Программы | Дополнительные файлы...» программы «Контрольный центр преподавателя», также входящей в комплекс Teacher Pack.

В последующих пунктах настоящего раздела описываются правила подготовки текстовых файлов, содержащих информацию о сводной группе заданий. При создании этих файлов в конструкторе вариантов им присваивается расширение .ptt.

6.2. Подготовка исходных данных для генерации сводной группы

6.2.1. Простейший набор исходных данных

Исходные данные для сводных групп удобнее всего создавать в редакторе конструктора вариантов PTVarMaker. В этом случае обеспечивается синтаксическое выделение различных элементов набора исходных данных; кроме того, созданный набор можно немедленно протестировать и создать на его основе как сам dll-файл сводной группы, так и html-файл с ее полным описанием (см. команду «Действия | Создать сводную группу заданий», п. 6.3.2).

Любой текстовый файл с данными для сводной группы должен содержать три обязательных элемента:

- **имя** создаваемой сводной группы (строка длины от 1 до 9 символов, содержащая только латинские буквы и цифры и оканчивающаяся буквой);
- **краткое описание** группы, которое одновременно будет заголовком html-файла с ее полным описанием (непустая текстовая строка длины не более 80 символов; точка в конце описания не ставится);

- **ключ** создаваемой сводной группы, позволяющий однозначно идентифицировать задания этой группы (непустая текстовая строка, содержащая не более 80 символов). На основе этого ключа генерируется уникальный *идентификатор группы*, который записывается в файл результатов вместе с информацией о выполнении задания из этой группы. Анализ идентификаторов групп дает возможность распознать ситуацию, когда учащийся выполнял задание из *другой* группы *с тем же именем* (для просмотра идентификаторов групп предназначена команда «Check-файлы | Просмотреть файл check.inf» программы «Контрольный центр преподавателя»).

Обязательные элементы сводной группы должны располагаться в указанном выше порядке, причем каждый элемент должен размещаться в отдельной строке после символа «=», например:

```
=GroupDemo
=Демонстрационная сводная группа
=1ret567fgd23KL56
```

Кроме этих элементов файл должен содержать **список заданий**, которые будут включены в данную сводную группу. Каждая строка в списке заданий должна начинаться с *названия группы*, после которого идет *список номеров* заданий из этой группы. Название группы должно соответствовать одной из существующих групп, регистр букв может быть произвольным. Номера заданий отделяются друг от друга и от названия группы одним или несколькими пробелами. Наряду с отдельным номером можно указывать *диапазон номеров*, разделяя начальный и конечный номер символом «–».

Приведем в качестве примера список заданий из двух базовых групп задачника:

```
Begin  1  2  4-6
Array 131-134
```

В сводной группе можно использовать не более 100 различных названий групп; общее количество заданий не должно превышать 999. Задания включаются в сводную группу в том порядке, в котором они указываются в наборе исходных данных.

Пустые строки или строки, состоящие только из пробелов, при обработке данных для сводной группы не учитываются.

Созданный набор данных, содержащий обязательные элементы сводной группы и список включаемых в нее заданий, надо сохранить в файле с расширением .ptt; наличие такого расширения указывает конструктору вариантов, что данный набор предназначен для создания сводной группы (в отличие от наборов, которые сохраняются в файлах с расширением .ptv и предназначены для создания файлов вариантов). По умолчанию созданный файл предлагается сохранить в специальном каталоге VARFILES (подкаталоге того каталога, в котором размещена программа «Конструктор вариантов»).

В редакторе конструктора вариантов для текста ptt-файлов используется специальное цветовое выделение: строки с обязательными элементами сводной группы выделяются зеленым цветом и полужирным начертанием, полужирное начертание применяется и для строк со списками заданий.

Вернемся к описанному ранее набору исходных данных для сводной группы:

```
=GroupDemo
=Демонстрационная сводная группа
=1ret567fgd23KL56
```

```
Begin  1  2  4-6
Array 131-134
```

Сводная группа, созданная на основе этого набора, будет иметь название GroupDemo и состоять из 9 заданий с именами от GroupDemo1 до GroupDemo9; при этом первые 5 заданий будут взяты из группы Begin, а последние 4 — из группы Array. Независимо от имени ptt-файла, содержащего приведенный набор исходных данных, dll-файл с созданной сводной группой будет иметь имя PT4GroupDemo.dll, поскольку имя любого файла с группой заданий должно совпадать с названием группы заданий, дополненным префиксом PT4 (суффикс, определяющий *локаль группы*, то есть язык ее интерфейса, в данном случае не используется). Этот файл будет создан в том же каталоге, в котором находится ptt-файл с исходными данными.

В этом же каталоге будет создан и файл PT4GroupDemo.html с полным описанием данной группы (html-файл, как и dll-файл, автоматически создается при успешном выполнении команды «Действия | Создать сводную группу заданий», при этом он сразу отображается на экране в веб-браузере, используемом на данном компьютере по умолчанию). Полное описание рассмотренной выше сводной группы будет иметь следующий вид:

Демонстрационная сводная группа

Ввод и вывод данных, оператор присваивания

GroupDemo1°. Дана сторона квадрата a. Найти его периметр $P = 4 \cdot a$.

GroupDemo2°. Дана сторона квадрата a. Найти его площадь $S = a^2$.

GroupDemo3°. Дан диаметр окружности d. Найти ее длину $L = \pi \cdot d$. В качестве значения π использовать 3.14.

GroupDemo4°. Дана длина ребра куба a. Найти объем куба $V = a^3$ и площадь его поверхности $S = 6 \cdot a^2$.

GroupDemo5°. Даны длины ребер a, b, c прямоугольного параллелепипеда. Найти его объем $V = a \cdot b \cdot c$ и площадь поверхности $S = 2 \cdot (a \cdot b + b \cdot c + a \cdot c)$.

Одномерные массивы: множества точек на плоскости

GroupDemo6. Дано множество A из N точек на плоскости и точка B (точки заданы своими координатами x, y). Найти точку из множества A, наиболее близкую к точке B. *Расстояние* R между точками с координатами (x_1, y_1) и (x_2, y_2) вычисляется по формуле:

$$R = ((x_2 - x_1)^2 + (y_2 - y_1)^2)^{1/2}.$$

GroupDemo7. Дано множество A из N точек (точки заданы своими координатами x, y). Среди всех точек этого множества, лежащих во второй четверти, найти точку, наиболее удаленную от начала координат. Если таких точек нет, то вывести точку с нулевыми координатами.

GroupDemo8. Дано множество A из N точек (точки заданы своими координатами x, y). Среди всех точек этого множества, лежащих в первой или третьей четверти, найти точку, наиболее близкую к началу координат. Если таких точек нет, то вывести точку с нулевыми координатами.

GroupDemo9°. Дано множество A из N точек (точки заданы своими координатами x, y). Найти пару точек этого множества с максимальным расстоянием между ними и само это расстояние (точки выводятся в том же порядке, в котором они перечислены при задании множества A).

Дата генерации страницы: 09.04.2009.

6.2.2. Добавление преамбулы к сводной группе и ее подгруппам

Сводные группы, помимо формулировок заданий, могут содержать поясняющий текст. Поясняющий текст к группе (*преамбула группы*) располагается непосредственно после ее заголовка. Кроме того, поясняющий текст может указываться и для каждой *подгруппы* сводной группы, то есть раздела группы, снабженного заголовком второго уровня. Подобный текст (*преамбула подгруппы*) располагается после заголовка подгруппы.

Раздел набора исходных данных, в котором указывается преамбула группы, необходимо предварять строкой, содержащей единственный символ «–» (тире). Такая же строка должна завершать фрагмент с текстом преамбулы, например:

```
-
Данная группа является примером \Iсводной группы\i,
включающей некоторые из заданий базовых групп Begin и Array.
-
```

Символы-тире, отмечающие начало и конец преабулы, выводятся в редакторе конструктора вариантов зеленым цветом, а сам текст преамбулы выделяется полужирным начертанием (подобно списку заданий, включаемых в сводную группу).

Текст преамбулы может содержать *управляющие последовательности*, позволяющие добавлять в текст индексы и специальные символы, форматировать фрагменты текста, переходить к новому абзацу в тексте и настраивать выравнивание абзацев. Большинство управляющих последовательностей начинаются с символа «\». В примере использованы парные управляющие последовательности `\I` и `\i`, выделяющие фрагмент, который будет выведен курсивом. Полный список управляющих последовательностей приводится в п. 4.2.

Каждая строка, задающая текст преамбулы, должна содержать не более 80 символов.

После добавления приведенного выше фрагмента в набор данных начальная часть описания сводной группы примет следующий вид:

Демонстрационная сводная группа

Данная группа является примером *сводной группы*, включающей некоторые из заданий базовых групп Begin и Array.

Ввод и вывод данных, оператор присваивания

GroupDemo1°. Дана сторона квадрата a. Найти его периметр $P = 4 \cdot a$.

GroupDemo2°. Дана сторона квадрата a. Найти его площадь $S = a^2$.

Просматривая описание созданной сводной группы, можно заметить, что все задания, взятые из группы Begin, объединены в подгруппу с заголовком «Ввод и вывод данных, оператор присваивания». Для этой подгруппы, как и для всей сводной группы, можно определить преамбулу. Преамбулы подгрупп должны определяться после преамбулы группы, при этом в строке, отмечающей начало преамбулы подгруппы, следует указать не только символ-тире, но и текст заголовка данной подгруппы, например:

```
- Ввод и вывод данных, оператор присваивания
Эта подгруппа содержит задания из группы Begin.
-
```

Текст заголовка в редакторе конструктора вариантов выводится зеленым цветом, как и предшествующий ему символ-тире.

После этого добавления начало описания сводной группы будет выглядеть следующим образом:

Демонстрационная сводная группа

Данная группа является примером *сводной группы*, включающей некоторые из заданий базовых групп Begin и Array.

Ввод и вывод данных, оператор присваивания

Эта подгруппа содержит задания из группы Begin.

GroupDemo1°. Дана сторона квадрата a. Найти его периметр $P = 4 \cdot a$.

GroupDemo2°. Дана сторона квадрата a. Найти его площадь $S = a^2$.

Строку, отмечающую начало текста очередной преамбулы, можно совместить со строкой, отмечающей конец предыдущей преамбулы. Например, фрагмент, задающий обе описанные выше преамбулы, можно оформить следующим образом:

```
-
Данная группа является примером \Iсводной группы\i,
включающей некоторые из заданий базовых групп Begin и Array.
- Ввод и вывод данных, оператор присваивания
Эта подгруппа содержит задания из группы Begin.
-
```

Количество строк с текстом всех преамбул, входящих в сводную группу, не должно превосходить 50.

Имеется возможность добавлять в преамбулу текст из преамбулы *уже имеющейся* группы или подгруппы. Для добавления текста преамбулы существующей *группы* достаточно указать в начале строки символ-звездочку, а затем — имя группы, преамбулу которой требуется добавить. В качестве примера добавим к преамбуле подгруппы «Ввод и вывод данных, оператор присваивания» текст из преамбулы группы Begin:

```
- Ввод и вывод данных, оператор присваивания
Эта подгруппа содержит задания из группы Begin.\P
* Begin
-
```

В результате заголовок данной подгруппы вместе с преамбулой примет следующий вид:

Ввод и вывод данных, оператор присваивания

Эта подгруппа содержит задания из группы Begin.

Все входные и выходные данные в заданиях этой группы являются вещественными числами.

Для перехода к началу нового абзаца в преамбуле была использована управляющая последовательность `\P`.

Как легко определить по описанию нашей сводной группы, включенные в нее задания Array131–Array134 взяты из подгруппы группы Array с заголовком «Одномерные массивы: множества точек на плоскости». Для добавления к этой подгруппе преамбулы из одноименной подгруппы группы Array достаточно определить раздел с заголовком этой подгруппы и указать в нем строку, содержащую *две звездочки* и имя исходной группы:

```
- Одномерные массивы: множества точек на плоскости
** Array
-
```

В результате подгруппа «Одномерные массивы: множества точек на плоскости» нашей сводной группы тоже получит преамбулу, импортированную из одноименной подгруппы группы Array:

Одномерные массивы: множества точек на плоскости

Для хранения данных о каждом наборе точек следует использовать по два массива: первый массив для хранения абсцисс, второй — для хранения ординат. Можно также использовать массив *записей* с двумя полями.

Напомним, что если бы мы указали не две, а одну звездочку, то в преамбулу был бы добавлен текст из преамбулы *ко всей группе* Array.

Итак, для того чтобы разработанная сводная группа и обе входящие в нее подгруппы были снабжены поясняющим текстом, достаточно добавить в ptt-файл следующий фрагмент:

```
-
Данная группа является примером \Iсводной группы\i,
включающей некоторые из заданий базовых групп Begin и Array.
- Ввод и вывод данных, оператор присваивания
Эта подгруппа содержит задания из группы Begin.\P
* Begin
- Одномерные массивы: множества точек на плоскости
** Array
-
```

6.2.3. Поправки к ссылкам и комментарии

При добавлении в сводную группу заданий, содержащих ссылки на другие задания, может возникнуть проблема, которую мы продемонстрируем на примере нашей сводной группы GroupDemo, добавив в нее еще один набор заданий из группы Array:

```
Array 116-118 120 124-126
```

Фрагмент описания сводной группы GroupDemo, соответствующий этим заданиям, будет иметь следующий вид:

Одномерные массивы: серии целых чисел

GroupDemo10°. Дан целочисленный массив A размера N. Назовем *серией* группу подряд идущих одинаковых элементов, а *длиной серии* — количество этих элементов (длина серии может быть равна 1). Сформировать два новых целочисленных массива B и C одинакового размера, записав в массив B длины всех серий исходного массива, а в массив C — значения элементов, образующих эти серии.

GroupDemo11. Дан целочисленный массив размера N. Вставить перед каждой его серией элемент с нулевым значением (определение серии дано в задании GroupDemo10).

GroupDemo12. Дан целочисленный массив размера N. Вставить после каждой его серии элемент с нулевым значением (определение серии дано в задании GroupDemo10).

GroupDemo13. Дан целочисленный массив размера N, содержащий по крайней мере одну серию, длина которой больше 1. Преобразовать массив, уменьшив каждую его серию на один элемент (определение серии дано в задании GroupDemo9).

GroupDemo14. Дано целое число K (> 0) и целочисленный массив размера N. Поменять местами последнюю серию массива и его серию с номером K (определение серии дано в задании GroupDemo6). Если серий в массиве меньше K, то вывести массив без изменений.

GroupDemo15. Дано целое число L (> 1) и целочисленный массив размера N. Заменить каждую серию массива, длина которой меньше L, на один элемент с нулевым значением (определение серии дано в задании GroupDemo6).

GroupDemo16. Дано целое число L (> 0) и целочисленный массив размера N. Заменить каждую серию массива, длина которой равна L, на один элемент с нулевым значением (определение серии дано в задании GroupDemo6).

Из приведенного описания видно, что если в тексте задания имеется ссылка на другое задание этой же группы, то в сводной группе эта ссылка корректируется таким образом, чтобы вместо имени исходной группы заданий (в нашем случае группы Array) указывалось имя создаваемой сводной группы (в нашем случае — GroupDemo). Кроме того, корректируется и номер задания. Однако этот номер будет указан правильно только в случае, если в сводную группу были добавлены *все задания*, заключенные между тем заданием, в котором имеется ссылка, и тем, на которое эта ссылка указывает. Так, в нашем примере ссылки на задание Array116 будут правильно пересчитаны для заданий Array117–Array118. В задании Array120 номер ссылки будет на 1 меньше, чем требуемый (GroupDemo9 вместо GroupDemo10). Еще больше будет отличаться от требуемого номер ссылки в заданиях Array124–Array126: вместо GroupDemo10 в них будет указана ссылка GroupDemo6.

Отмеченные несоответствия связаны с тем, что в сводную группу не были включены некоторые задания, расположенные между заданиями, которые содержат ссыл-

ки, и заданием Array116, на которое эти ссылки должны указывать. Так как между Array120 и Array116 было пропущено только одно задание, ссылка отличается от требуемой на 1. Поскольку между группой заданий Array124–Array126 и Array116 было пропущено 4 задания, именно на эту величину отличаются ссылки для заданий данной группы.

Для корректировки подобных ошибок следует дополнить номера ошибочных заданий, указав в них *поправку для номера ссылки* и отделив ее от номера задания символом #. В нашем случае для задания Array120 надо указать поправку, равную 1, а для заданий Array124–Array126 — поправку, равную 4:

```
Array 116-118 120#1 124-126#4
```

Теперь во всех заданиях подгруппы «Одномерные массивы: серии целых чисел» сводной группы GroupDemo будут указываться верные ссылки на задание GroupDemo10 — первое задание этой подгруппы.

Приведенный пример показывает, что поправку для ссылки можно указывать как для отдельного задания, так и для диапазона заданий. Допустимо указывать как положительные, так и отрицательные поправки; значения поправок должны лежать в диапазоне от –50 до 50.

Осталось отметить еще одну возможность, предусмотренную для ptt-файлов: это *строки-комментарии*. Если в начале строки указать символ % (знак процента), то данная строка будет считаться комментарием и не будет учитываться при обработке ptt-файла и создании на его основе сводной группы. Эту возможность удобно использовать для добавления в ptt-файл дополнительных пояснений, которые не требуется включать в описание созданной сводной группы. Кроме того, это позволяет временно закомментировать часть исходных данных при разработке новой сводной группы. Подчеркнем, что закомментировать можно только всю строку; если символ процента появляется в середине строки, то он считается обычным символом.

Текст комментариев выделяется в редакторе конструктора вариантов синим цветом и курсивным начертанием.

В заключение данного раздела приведем полный текст разработанной сводной группы, снабдив его комментарием:

```
% ======================================================================
% ДАННЫЕ ДЛЯ ДЕМОНСТРАЦИОННОЙ СВОДНОЙ ГРУППЫ УЧЕБНЫХ ЗАДАНИЙ.
% Предназначены для генерации сводной группы заданий с именем GroupDemo.
% Группа сохраняется в файле PT4GroupDemo.dll.
% ======================================================================

=GroupDemo
=Демонстрационная сводная группа
=1ret567fgd23KL56

Begin  1  2  4-6
Array 131-134
Array 116-118 120#1 124-126#4

-
Данная группа является примером \Iсводной группы\i,
включающей некоторые из заданий базовых групп Begin и Array.
- Ввод и вывод данных, оператор присваивания
Эта подгруппа содержит задания из группы Begin.\P
* Begin
- Одномерные массивы: множества точек на плоскости
```

```
** Array
-
```

Заметим, что файл GroupDemo.ptt с данной сводной группой, снабженный дополнительными комментариями, можно автоматически создать и загрузить в конструктор вариантов, выполнив команду «? | Демонстрационная сводная группа» (см. п. 6.3.4).

6.3. Команды конструктора вариантов, связанные с созданием сводных групп

6.3.1. Режим «Сводная группа»

В версии 2.4 программы «Конструктор вариантов» к имеющимся режимам «Набор заданий», «Дополнительный набор заданий», «Var-файл» и «Текстовый файл» добавлен новый режим «Сводная группа», позволяющий создавать новые сводные группы учебных заданий. Только в этом режиме доступна команда «Создать сводную группу заданий». Команды «Создать варианты», «Создать check-файл» и «Просмотреть текущий check-файл» в данном режиме недоступны. Программа находится в данном режиме, если текущий файл имеет расширение .ptt и не имеет атрибута «Только для чтения».

6.3.2. Команда «Действия | Создать сводную группу заданий»

Доступность: в режиме «Сводная группа».

Горячая клавиша: F9.

Кнопка:  «Создать сводную группу заданий (F9)».

Назначение: создание сводной группы заданий (dll-файла) на основе текущего набора исходных данных.

Перед созданием dll-файла выполняется автоматическое сохранение текущего ptt-файла с набором исходных данных (если он был изменен). Затем производится проверка правильности набора исходных данных; при обнаружении ошибки выводится соответствующее сообщение, и действие команды «Создать сводную группу заданий» отменяется.

Если текущий набор исходных данных является правильным, то на его основе генерируется dll-файл, который сохраняется в том же каталоге, что и ptt-файл с исходными данными. Имя dll-файла всегда состоит из двух частей: префикса «PT4» и имени созданной сводной группы; таким образом, имя dll-файла может отличаться от имени ptt-файла с исходными данными. Если в данном каталоге уже существует dll-файл с тем же именем, то прежний вариант файла заменяется на новый; запрос на подтверждение замены при этом не выводится.

Действия после успешного создания dll-файла зависят от того, установлен ли на данном компьютере универсальный вариант задачника Programming Taskbook версии не ниже 4.8. Если указанный вариант задачника отсутствует, то выводится сообщение о том, что dll-файл с требуемой группой создан, однако для его тестирования следует установить универсальный вариант задачника Programming Taskbook. Если же требуемый вариант задачника обнаружен, то запускается его программный модуль PT4Demo, который генерирует html-файл с описанием созданной сводной группы и немедленно отображает его с помощью html-браузера, используемого на данном компьютере по умолчанию (html-файл имеет то же имя, что и dll-файл, и сохраняется в

том же каталоге). После выполнения указанных действий выполнение команды «Создать сводную группу заданий» завершается

Если на компьютере установлен универсальный вариант задачника версии не ниже 4.8, то после успешного создания сводной группы ее можно дополнительно протестировать с помощью команды «Просмотреть задания в демо-режиме», описываемой далее.

Примечание: команда «Создать сводную группу заданий» доступна в программе «Конструктор вариантов», начиная с версии 2.4.

6.3.3. Команда «Действия | Просмотреть задания в демо-режиме»

Доступность: в любом режиме при условии, что на компьютере установлен универсальный вариант задачника Programming Taskbook 4. Если задачник не найден, то данная команда в меню не отображается.

Горячая клавиша: F11.

Кнопка:  «Просмотреть задания в демо-режиме (F11)». Данная кнопка отображается на панели инструментов только в том случае, если связанная с ней команда отображается в меню.

Назначение: просмотр в демо-режиме всех имеющихся групп учебных заданий. Для просмотра запускается внешнее приложение — программный модуль PT4Demo, входящий в состав универсального задачника по программированию Programming Taskbook 4. При просмотре задания отображается его формулировка, а также варианты его исходных и контрольных данных.

Если модуль PT4Demo имеет версию не ниже 4.8, то при его использовании доступны следующие дополнительные возможности:

- возможность просмотра не только базовых групп заданий, но и дополнительных групп, созданных с помощью конструктора PT4TaskMaker (в том числе всех сводных групп, находящихся в том же каталоге, что и файл, загруженный в программу «Конструктор вариантов»);
- просмотр описания любой группы заданий в html-формате; описание содержит не только формулировки заданий, но и дополнительные комментарии (*преамбулы*) к группе заданий и ее подгруппам;
- возможность выбора языка программирования перед демо-просмотром группы заданий. Данная возможность полезна в ситуации, когда формулировки заданий в группе зависят от текущего языка программирования (примером подобных групп являются группы Dynamic и Tree). Для выбора языка программирования следует использовать контекстное меню окна модуля PT4Demo.

После успешного создания сводной группы командой «Создать сводную группу заданий» можно сразу просмотреть созданную группу в демо-режиме (при условии, что используется модуль PT4Demo версии не ниже 4.8). В этом случае при выполнении команды «Просмотреть задания в демо-режиме» в окне модуля PT4Demo в качестве текущей выбирается только что созданная сводная группа.

Примечание: команда «Просмотреть задания в демо-режиме» доступна в программе «Конструктор вариантов», начиная с версии 2.4.

6.3.4. Команда «? | Демонстрационная сводная группа»

Доступность: в любом режиме.

Горячая клавиша: Shift+Ctrl+F1.

Назначение: открытие файла GroupDemo.ptt, предназначенного для создания демонстрационной сводной группы с тем же именем. Если текущий файл был изменен, то предварительно выводится запрос на сохранение изменений текущего файла; при варианте ответа «Отмена» действие команды «Демонстрационныая сводная группа» отменяется.

Файл GroupDemo.ptt ищется в подкаталоге VARFILES системного каталога программы; если данный файл отсутствует, то он автоматически создается. Если файл существует, но его текст отличается от стандартного демонстрационного текста, то выводится запрос на обновление его содержимого.

Ниже приводится стандартный текст файла GroupDemo.ptt.

```
% ============================================================================
% ДАННЫЕ ДЛЯ ДЕМОНСТРАЦИОННОЙ СВОДНОЙ ГРУППЫ УЧЕБНЫХ ЗАДАНИЙ.
% Предназначены для генерации сводной группы заданий с именем GroupDemo.
% Группа сохраняется в файле PT4GroupDemo.dll.
% ============================================================================
% ПРИМЕЧАНИЕ. Символ "^" используется далее в тексте как "стрелка вверх",
%             указывающая на определенную позицию предыдущей строки
%             или строку в целом.
% ============================================================================
% ^ К о м м е н т а р и й  начинается с символа "%", расположенного в начале
% строки. Комментарии и пустые строки при обработке данных не учитываются.

=GroupDemo
% ^ И м я  создаваемой группы (не более 9 символов - латинских букв и цифр;
% не должно оканчиваться цифрой).
=Демонстрационная сводная группа
% ^ К р а т к о е  о п и с а н и е  группы; точка в конце описания не ставится.
=1ret567fgd23KL56
% ^ К л ю ч  группы, позволяющий однозначно ее идентифицировать
% (произвольная последовательность символов).
% Все элементы данных, начинающиеся с символа "=", являются обязательными
% и должны задаваться в указанном порядке.

Begin   1 2 4-6
% ^ Имя темы. ^ Список номеров заданий; может содержать произвольное число
% элементов. Номера разделяются одним или несколькими пробелами;
% допустимо указывать  д и а п а з о н ы  номеров.
% Все задания в указанном порядке включаются в созданную сводную группу.
Array  131-134
Array  116-118 120#1 124-126#4
% ^ После символа "#" можно указать поправку для правильного вычисления ссылок
% на другие задания группы. Эту поправку, вместе с предваряющим символом "#",
% можно указывать как после отдельного номера, так и после диапазона номеров.

-
% ^ Метка начала  п р е а м б у л ы  к сводной группе, включаемой
% в ее описание (преамбулу указывать необязательно).
Данная группа является примером \Iсводной группы\i,
включающей некоторые из заданий базовых групп Begin и Array.
% ^ Текст преамбулы, который может содержать управляющие последовательности.
% Управляющие последовательности \I и \i выделяют курсивный фрагмент текста.
- Ввод и вывод данных, оператор присваивания
% ^ Метка конца преамбулы группы, одновременно являющаяся меткой начала
% преамбулы подгруппы с указанным именем.
```

```
Данная подгруппа содержит задания из группы Begin.\P
% ^ Управляющая последовательность \P обеспечивает переход к новому абзацу.
* Begin
% ^ Наличие звездочки с последующим именем группы означает подключение
% преамбулы, которая связана с указанной группой.
- Одномерные массивы: множества точек на плоскости
** Array
% ^ Наличие  д в у х  звездочек с последующим именем группы означает
% подключение преамбулы, которая связана с той подгруппой указанной группы,
% имя которой совпадает с именем определяемой подгруппы сводной группы.
-
% ^ Метка конца преамбулы подгруппы.

% ============================================================================
```

Примечание: команда «Демонстрационная сводная группа» доступна в программе «Конструктор вариантов», начиная с версии 2.4.

Литература

1. *Абрамян А. В., Абрамян М. Э.* Практикум по программированию на языке Visual Basic. — Ростов н/Д.: «ЦВВР», 2007. — 228 с.
2. *Абрамян М. Э.* 1000 задач по программированию. Часть I: Скалярные типы данных, управляющие операторы, процедуры и функции. — Ростов н/Д.: УПЛ РГУ, 2004. — 43 с.
3. *Абрамян М. Э.* 1000 задач по программированию. Часть II: Минимумы и максимумы, одномерные и двумерные массивы, символы и строки, двоичные файлы. — Ростов н/Д.: УПЛ РГУ, 2004. — 42 с.
4. *Абрамян М. Э.* 1000 задач по программированию. Часть III: Текстовые файлы, составные типы данных в процедурах и функциях, рекурсия, указатели и динамические структуры. — Ростов н/Д.: УПЛ РГУ, 2004. — 43 с.
5. *Абрамян М. Э.* Проведение практических занятий с использованием задачника Programming Taskbook. Методическая разработка для преподавателей программирования. — Банк компьютерных изданий РГУ, 2006. — 83 с. (http://open-edu.rsu.ru/upload/pub/37607100_1149508000Archive.zip)
6. *Абрамян М. Э.* Практикум по программированию на языках C# и VB .NET. 2-е изд. — Ростов н/Д.: «ЦВВР», 2007. — 220 с.
7. *Абрамян М. Э.* Практикум по программированию на языке Паскаль: Массивы, строки, файлы, рекурсия, линейные динамические структуры, бинарные деревья. — 6-е изд., перераб. и доп. — Ростов-на-Дону: «ЦВВР», 2008. — 227 с.
8. *Абрамян М. Э., Михалкович С. С.* Основы программирования на языке Паскаль: Скалярные типы данных, управляющие операторы, процедуры и функции, работа с графикой в системе PascalABC.NET. 4-е изд., перераб. и доп. — Ростов н/Д.: «ЦВВР», 2008. — 223 с.

Содержание