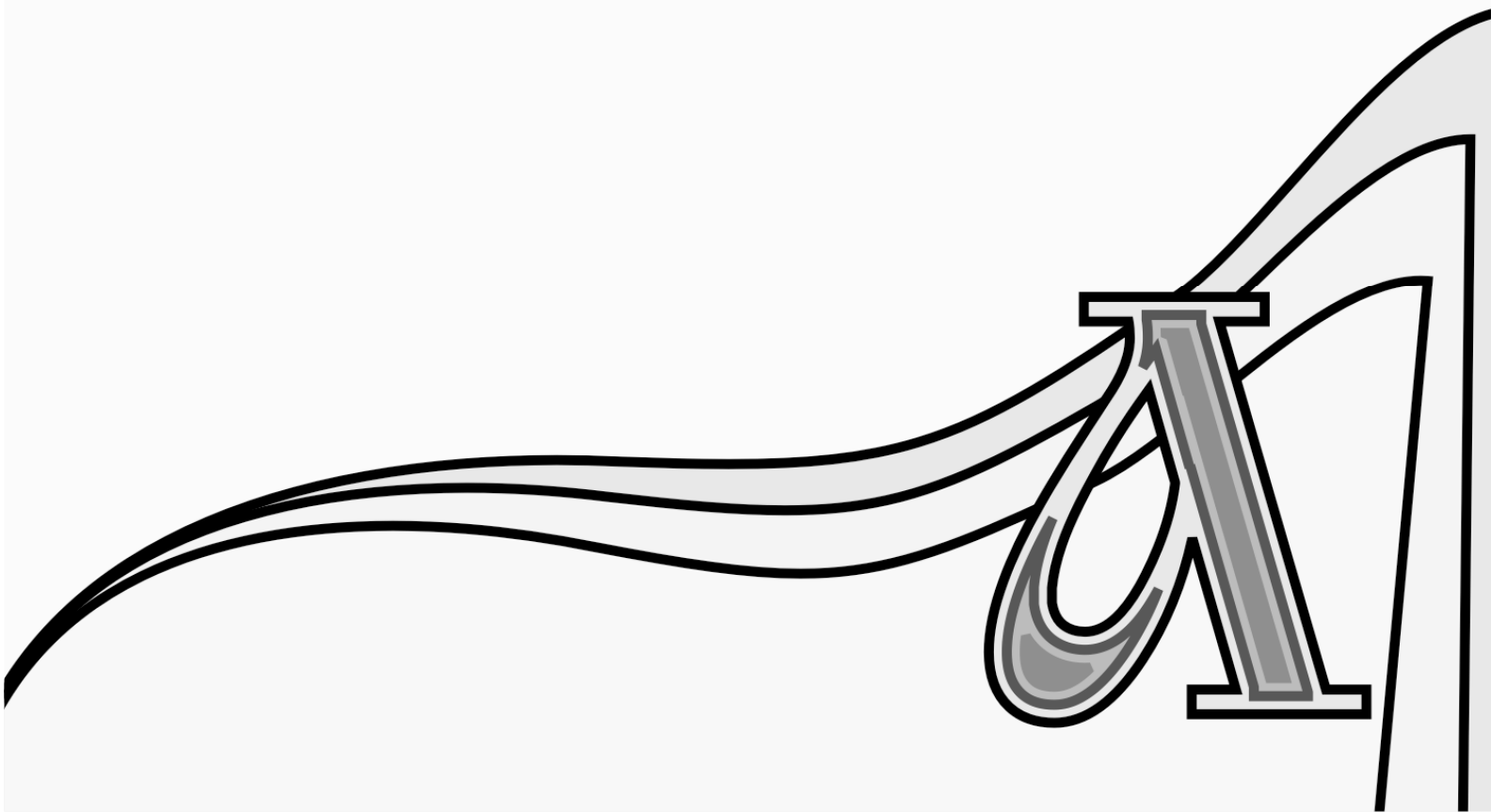
Кручинин В.В.

Генераторы в компьютерных учебных программах



УДК 378.16:004
ББК 65.9(2)32

Рецензенты:

В.А. Силич, доктор технических наук, профессор;
В.А. Якутенок, доктор физико-математических наук.

Кручинин В.В.
Генераторы в компьютерных учебных программах.– Томск: Изд-во Том. ун-та, 2003. –200с.

ISBN 5-7511-1763-8

В монографии рассматриваются математические основы, модели и алгоритмы генераторов тестовых заданий и вопросов. Описана технология построения и использования генераторов. Показаны конкретные разработки генераторов и внедрение их в систему контроля знаний в Томском межвузовском центре дистанционного образования.

Для студентов, аспирантов, специалистов и преподавателей, интересующихся вопросами построения программного обеспечения для дистанционного образования.

УДК 378.16:004
ББК 65.9(2)32

ISBN 5-7511-1763-8

Светлой памяти
Бондаря Владимира Антоновича
посвящается

ВВЕДЕНИЕ

Постоянное совершенствование компьютеров и компьютерных сетей существенно раздвигает границы их использования. Вместе с тем наблюдается тенденция усложнения программного обеспечения, вследствие переноса все большего числа человеческих функций на компьютеры. В организации современного учебного процесса программное обеспечение играет все возрастающую роль, особенно это касается дистанционной технологии обучения.

Опыт организации учебного процесса в Томском межвузовском центре дистанционного образования (ТМЦДО) по дистанционной технологии обучения показывает, что компьютерные учебные программы (КУП) играют существенную роль в повышении качества обучения [1]. Уже немало сделано: разработаны компьютерные учебники по математике, современному естествознанию и др. [2, 3]; разработаны и совершенствуются системы проведения компьютерных экзаменов и контрольных работ [3–5]; внедряются сетевые технологии обучения (проведение консультаций в форме чата, использование сайта ТМЦДО) [7]; внедряется информационная система электронного документооборота [8, 9].

Важнейшей составляющей дистанционной технологии обучения является система контроля знаний [10]. В конечном счете от качества системы контроля знаний зависит качество выпускаемых специалистов. В ТМЦДО создана довольно развитая система контроля знаний, которая включает [11]: организацию промежуточного контроля различной формы – выполнение компьютерных контрольных работ, написание письменных работ, рефератов, отчетов, представляемых в электронной форме; проведение компьютерного экзамена в форме тестирования. Для обеспечения оперативной обработки писем студентов, которые в

основном присылают письменные отчеты или протоколы компьютерных контрольных работ, создан диспетчерский отдел, где ежедневно принимается и обрабатывается до 500 писем, оправленных по электронной почте.

Анализ проведения экзаменационных сессий [7], который выполнял учебный отдел ТМЦДО, показал, что:

1) хакеры организуют атаки на систему проведения компьютерного тестирования;
2) для многих компьютерных экзаменов подготовлены шпаргалки;
3) шпаргалки и взломанные компьютерные экзамены размещаются на специальных сайтах.

Вследствие чего состоялась серия семинаров под руководством профессора В.А. Бондаря, на которых была выработана стратегия улучшения системы контроля знаний [8, 12]:

1) создание системы защиты от несанкционированного доступа;
2) организация защиты от шпаргалок;
3) организация работ по закрытию сайтов.
4) совершенствование системы проведения экзаменов и контрольных работ.

Одним из возможных направлений решения указанных проблем является внедрение в практику дистанционного контроля генераторов тестовых заданий. Генераторы, с одной стороны, решают проблемы защиты от несанкционированного доступа, т.к. не имеют заранее заготовленных ответов, с другой стороны, практически каждый студент получает индивидуальное задание. Это решает проблему шпаргалки, потому что программа генерирует правильный ответ в процессе проведения опроса. Отсюда вместо запоминания правильного ответа, необходимо знать алгоритм решения.

В.А. Бондарем перед лабораторией инструментальных систем моделирования и обучения (ЛИСМО) была поставлена научно-техническая задача – разработка методов генерации тестовых заданий и вопросов, реализация их в компьютерных учебных программах и внедрение в практику дистанционного контроля знаний.

В рамках госбюджетных тем 1.1.01, 1.03 раздела 06.01 Томского университета систем управления и радиоэлектроники (ТУ-

СУР) проводились теоретические исследования, связанные с данной научно-технической задачей. Реализация и внедрение генераторов финансируются ТМЦДО.

Было выявлено два основных направления исследований.

1. Моделирование деятельности преподавателя при приеме экзамена. Применение идей искусственного интеллекта для решения задачи синтеза вопросов. Построение интеллектуальной системы приема экзаменов.

2. Инженерный подход при построении генераторов, основанный на применении эвристических и комбинаторных алгоритмов генерации многовариантных тестовых заданий и вопросов.

Первое направление является наиболее перспективным, т.к. при его осуществлении будет решена задача качественного приема компьютерного экзамена. Однако первые попытки решения натолкнулись на задачи, которые не имеют удовлетворительного решения на современном уровне развития искусственного интеллекта. Одна из таких задач – понимание текста на естественном языке. Отсюда невозможность решения задачи синтеза корректного вопроса.

Результаты исследований в этом направлении могут привести к появлению интеллектуальных систем, помощников преподавателя при составлении тестовых вопросов [13].

Второе направление связано с построением алгоритмов, которые генерируют многовариантные тестовые задания и вопросы. Например, в математической задаче можно менять различные входные параметры и тем самым получать множество однотипных задач. Кроме того, имеются определенные результаты в этой области [14, 15], показывающие возможность использования таких алгоритмов.

Развивая второе направление, в ЛИСМО удалось:

1) разработать математические основы построения алгоритмов генерации тестовых заданий;

2) построить модели и алгоритмы генерации тестовых заданий.

3) разработать и внедрить оригинальную технологию создания генераторов;

4) разработать серию генераторов по различным дисциплинам и внедрить их в практику дистанционного обучения [16-18].

В данной монографии сделана попытка обобщения результатов исследований, связанных с построением и внедрением генераторов в практику дистанционного обучения.

В первой главе проводится системный анализ понятия «Генератор», рассматриваются различные схемы использования генераторов.

Во второй главе рассмотрены математические основы построения генераторов. Вводится понятие генерирующего алгоритма. Приводятся алгоритмы генерации комбинаторных объектов: перестановок, сочетаний, размещений. Исследуются алгоритмы генерации, основанные на использовании грамматик и деревьев И-ИЛИ.

В третьей главе вводятся модели и алгоритмы генерации конкретных тестовых заданий и вопросов. Рассматриваются генераторы для задач и вопросов типа меню. Вводится понятие модели предметной области для генерации вопросов. Рассматриваются модели реляционные, графовые и иерархические. Проводятся их исследования.

В четвертой главе приводится технология разработки генераторов, дан язык представления тестов, описаны системы проведения экзаменов и контрольных работ.

В пятой главе описано программное обеспечение: шаблоны для представления генерации комбинаторных объектов, система классов генерации операторов языка описания теста, обсуждены вопросы, связанные с встраиванием объектов в языки интерпретирующего типа, показана программная реализация конкретных генераторов и шаблонов.

Автор благодарит за помощь и содействие при написании монографии проф. В.П. Тарасенко, проф. А.В. Кобзева, А.Ф. Уварова, проф. Л.И. Магазинникова, проф. А.В. Шарапова, проф. Н.В. Тимченко, доцента Л.Е. Лычковскую, доцента Е.Н. Рыбалову, а также сотрудников лаборатории инструментальных систем моделирования и обучения О.Ю. Исакову, Ю.В. Морозову, М.Ф. Молочко, Н.В. Кашкареву, Е.А. Лоор.

Глава 1. СИСТЕМНЫЙ АНАЛИЗ

Системный анализ предполагает рассмотрение проблемы с самых общих позиций [19]. Термин «генератор» (латинское generator – производитель) возник в технических науках и означает устройство, производящее какой-либо продукт (парогенератор, электрический генератор, генератор импульсов и т.д.) [20]. В теории автоматического управления введено понятие генератора входных возмущений. Это устройство, которое позволяет генерировать по определенному закону входное возмущение для исследования какой-либо системы автоматического регулирования [21]. Рассматривая систему как черный ящик, можно предложить следующую схему (рис. 1.1). В данной схеме генератор является моделью внешней среды и служит инструментом исследования системы.

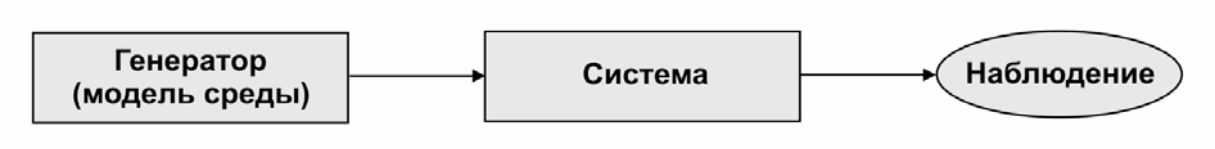


Рис. 1.1. Исследование модели черного ящика

Развитие систем передачи информации [22, 23], в которых происходит кодирование информации, также приводит к понятию генератора (рис. 1.2). Первоначально информационный объект кодируется, т.е. ему ставится в соответствие некоторое число (идентификатор). Далее это число передается по каналам связи и поступает на вход декодера. Декодер производит обратную операцию, по заданному коду восстанавливает информационный объект. В некоторых случаях декодер можно рассматривать как генератор, в том смысле, что по некоторому коду (числу, параметру) устройство генерирует информационный объект.

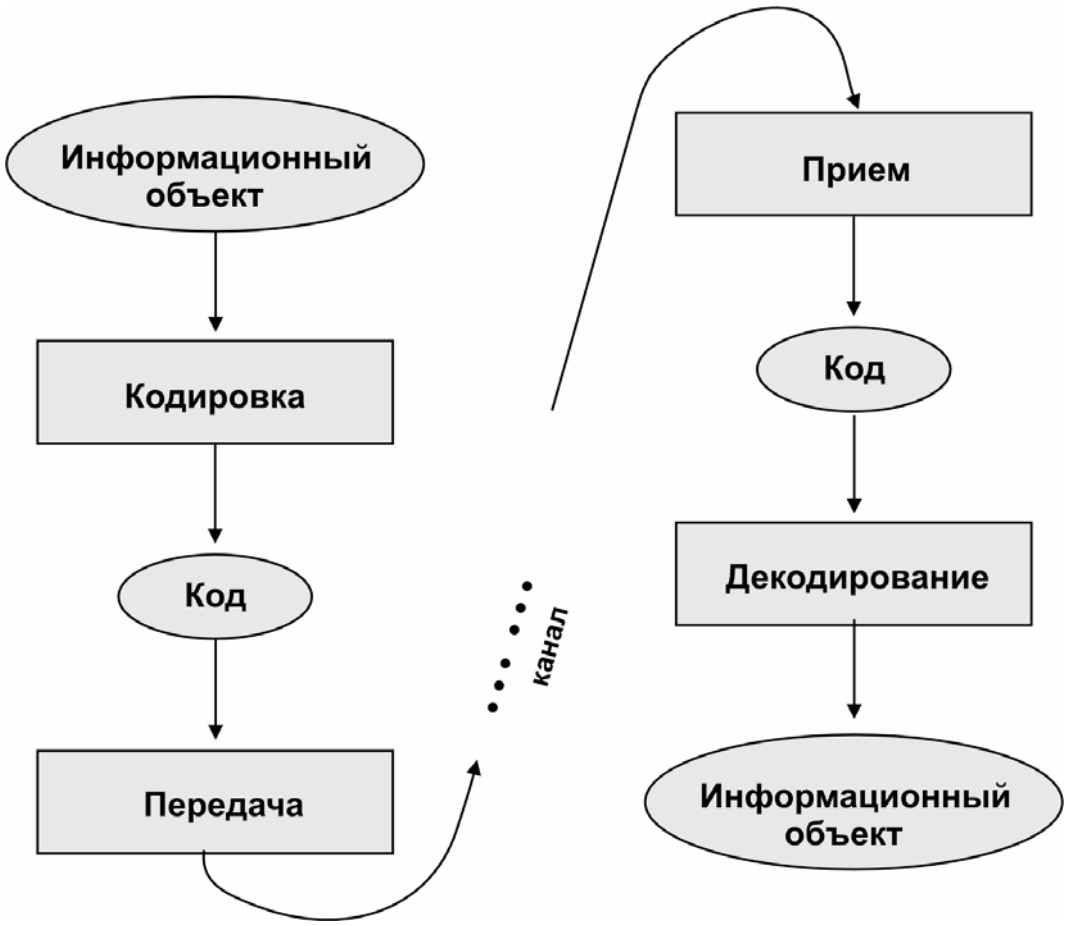


Рис. 1.2. Генераторы при приеме информации

Более сложные схемы использования генераторов возникли при создании технических систем с элементами обучения. На рис 1.3 показана обобщенная схема. Генератор на основе конкретных значений входных параметров генерирует входное воздействие и требуемый (желательный) выход системы. Система преобразует входной сигнал в выходной, блок обучения производит сравнение выходных сигналов генератора и системы, устанавливает структуру и значения параметров системы и выдает новые значения для генератора. Процесс обучения завершается на основе некоторого критерия оптимальности.

Примерами таких систем являются корреляционно-экстремальные системы навигации и наведения [24], робототехнические системы [25], нейросистемы и нейрокомпьютеры [26, 27].

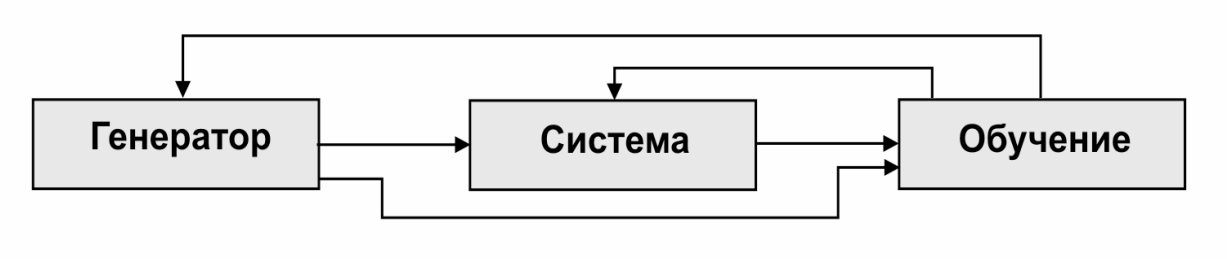


Рис. 1.3. Схема обучения систем

Исследования в компьютерных науках, особенно в области моделирования сложных систем, привели к появлению методов Монте-Карло [28], а затем к созданию теории имитационного моделирования [29]. Имитационное моделирование является эффективным инструментом исследования сложных систем. В этом случае строятся упрощенные программные модели, имитирующие как элементы исследуемой системы, так и существенные элементы среды, влияющие на поведение системы. Одним их важнейших элементов систем имитационного моделирования являются датчики случайных чисел (ДСЧ). ДСЧ – это программы, генерирующие псевдослучайные последовательности целых или вещественных чисел. Для исследования различного класса систем появились разнообразные программные генераторы, основанные на использовании ДСЧ. Примерами могут служить генераторы изображений с заданными статистическими свойствами [30], генераторы карт [31], генераторы сценариев [32].

Другими важными классами программных генераторов являются генераторы отчетов [33], широко применяемых в базах данных. Однако данный класс программ нельзя считать в чистом виде генератором, поскольку результат их работы не подается на вход системы, а служит конечным продуктом. Генераторы программ – еще один класс подобных систем, который по некоторому описанию на выходе получает программу на некотором языке программирования, например генератор различного рода трансляторов и интерпретаторов по описанию входного языка [34]. На сайте www.rvb.ru [35] дан список программ–генераторов текста различного назначения: генерации русскоязычных стихоподобных текстов («инструмент поэта»); генератор письменных жалоб Скотта Пейкина; программы генерации любовных писем; генератор псевдофилософских текстов; генератор случайных текстов на основе заданной грамматики для английского языка.

Развитие идей искусственного интеллекта, воплощение их в реальных системах привело к появлению более сложных

схем. На рис. 1.4 представлена обобщенная схема, состоящая из генератора, планировщика, решателя и советчика [14]. **Генератор** обеспечивает построение некоторого обучающего воздействия или учебного задания, которое поступает на вход системы. Генерация воздействия или задания осуществляется на основе параметров, которые устанавливает **Планировщик**. Последний выполняет построение заданной последовательности учебных заданий, после выполнения которых, можно сказать, что система обучена.

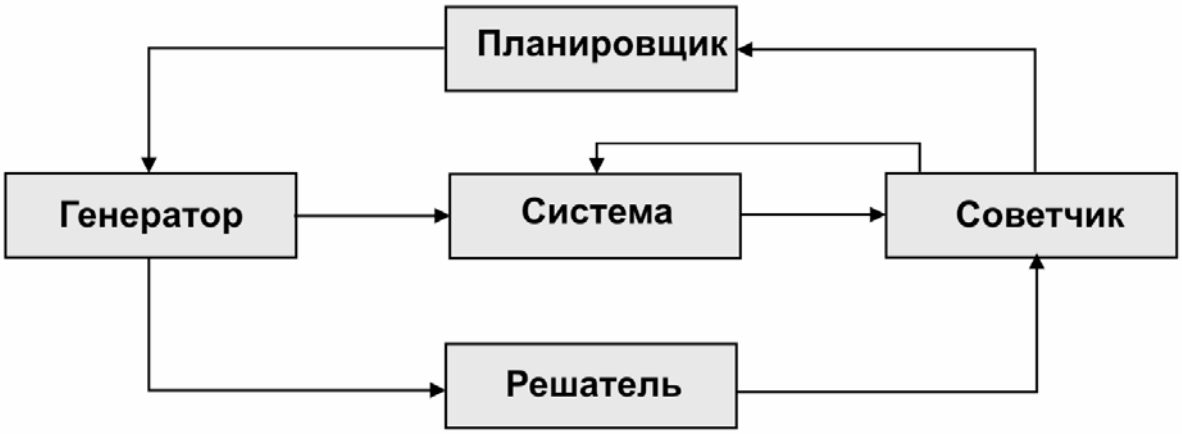


Рис. 1.4. Схема обучения интеллектуальных систем

Полученное задание поступает на вход **Системы** и **Решателя**. **Решатель** обеспечивает стандартное выполнение задания, состоящее из некоторой последовательности элементарных актов. Важно отметить, что эта последовательность не единственна, а может быть некоторое множество последовательностей.

Результаты выполнения задания системой и решателя поступают на вход **Советчика**. Он обеспечивает сравнение результатов работы системы и решателя, устраняет неточности и ошибки в работе системы при выполнении задания. После того как система обеспечивает заданное качество выполнения задания, **Советчик** выдает сообщение **Планировщику** для продолжения обучения. Завершение обучения осуществляет **Планировщик** в соответствии с прохождением плана обучения и достижения заданного критерия обучения.

Здесь за кадром осталась база знаний, существенный элемент интеллектуальной системы. Понятно, что ни генератор, ни планировщик, ни советчик, ни решатель не могут

обойтись без базы знаний. Предполагается, что каждый из указанных модулей обращается к своей или интегрированной базе знаний.

Еще одна область применения генераторов – стимулирование творческой (созидательной, креативной) деятельности человека [36, 37]. Создаются специальные программные системы, так называемые генераторы идей, которые обеспечивают стимулирование творческой деятельности человека. Например, программное обеспечение фирмы Creax [38] обеспечивает анализ и решение проблем, определение выгод и убытков от принятия решений в сфере бизнеса и менеджмента.

Итак, рассмотрев основные схемы использования генераторов, можно выделить следующее.

1. Генератор является важной частью схемы исследования систем, частным случаем таких исследований является тестирование. Тестирование может иметь различные цели, например, для технических систем – это может быть оценивание технических параметров, для программных систем – обнаружение ошибок или доказательство их отсутствия. Для человека – это определение его психофизического состояния или определение уровня знаний.

2. Генератор является важной частью схемы обучения систем. Он позволяет генерировать множество учебных примеров. Обучение систем также можно классифицировать на обучение технических, программных и организационных систем.

3. Генератор может использоваться в системах кодирования и декодирования информации. Здесь генераторы позволяют решать задачи декодирования при сжатии, шифровании и повышении надежности передачи информации.

4. Генератор идей используется для усиления творческой деятельности человека: решении проблем, изобретательстве и других областях. Существенно расширяет границы поиска решений.

Таким образом, генератор можно рассматривать как систему, на вход которого поступает описание свойств объекта, а на выходе получаем сам объект. Причем часть свойств или значений объекта генератор устанавливает сам. В этом состоит принципиальное отличие генератора от других систем.

Глава 2. МАТЕМАТИЧЕСКИЙ АППАРАТ

2.1. Введение

Множество одно из базовых понятий многих прикладных и теоретических дисциплин. Рассматриваются только конечные множества. В программировании любые конечные множества можно всегда представить линейным массивом и различать элементы по индексам. Поэтому ниже будем рассматривать множество индексов представленными числами {0,1,2, ... n-1}.

Определение 1. Под **объектом о** будем понимать конченую последовательность возможно повторяющих элементов данного множества **N**.

Определение 2. Генерирующий алгоритм – это алгоритм, обеспечивающий конструирование или выбор объекта из заданного множества объектов.

Определение 3. Алгоритм идентификации – это алгоритм, который объекту из заданного множества ставит в соответствие однозначное натуральное число.

Перечислим основные задачи при построении и исследовании генерирующих алгоритмов:

- подсчет количества объектов получаемых с помощью генерирующего алгоритма;
- построение и исследование алгоритмов нумерации объектов;
- построение и исследование алгоритмов управления генерацией.

Мощность множества объектов для данного генерирующего алгоритма является важной характеристикой, позволяющей оценить возможности генератора. Например, если объект является перестановкой некоторого конечного множества элементов, то мощность всех перестановок будет n!. В общем случае для исследования мощности множества объектов можно использовать теорию нумерации [39, 40].

Рассмотрим задачу построения алгоритма, который производит нумерацию объектов, т.е. каждому объекту из заданного множества объектов ставит в соответствие некоторое натуральное число, причем для каждого объекта должны выполняться свойства:

1) для **a** объекта это натуральное число единственно;
2) если есть объект с номером **i**, то есть объект с номером **i+1**.

Можно предложить следующие алгоритмы, связанные с нумерацией множества объектов:

1) по заданному объекту **a** поставить в соответствие натуральное число **i**;
2) по заданному натуральному числу **i** найти объект **a**;
3) зная **i**-й объект, необходимо найти **i**+1 объект (или **i**-1).

Ниже будут описаны алгоритмы генерации и идентификации для комбинаторных объектов: подмножеств, перестановок, сочетаний, размещений.

Отдельные исследования, посвященные алгоритмам генерации и идентификации комбинаторных объектов, можно найти в работах [41, 42].

2.2. Генерация подмножества

Общее число подмножеств для данного множества **E** может быть подсчитано на основе следующей формулы:

$$\mathrm{P}(n) = 2^n, \qquad (2.1)$$

где n – мощность исходного множества. Таким образом, каждому подмножеству можно поставить в соответствие некоторое число i. Двоичное представление этого числа даст однозначное соответствие между числом и подмножеством.

Например, дана следующая таблица для множества {a, b, c}:

Число	a	b	c
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1

6	1	1	0
7	1	1	1

Тогда все множество подмножеств без учета пустого множества будет {c, b, cb, a, ca, cb, cba}.

Можно предложить следующий алгоритм генерации подмножества некоторого множества E_n по номеру k:

1) каждому элементу множества ставим в соответствие некоторый индекс j.

2) записываем двоичное представление номера k.

3) в подмножество записываем те элементы множества, индексы которых соответствуют позиции единиц в двоичном представлении номера k.

2.3. Генерация перестановок

Перестановки – это множества, состоящие из одних и тех же элементов и отличающиеся только порядком расположения этих элементов. Возьмем n различных элементов $\{a_1, a_2, a_3, \ldots, a_n\}$ и переставим эти элементы всевозможными способами, оставляя без изменения числа элементов, меняя только порядок их расположения. Обозначим общее число полученных таким образом перестановок P_n. Число перестановок вычисляется по следующей формуле:

$$P_n = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n = n! \qquad (2.2)$$

Наша цель состоит в том, чтобы построить алгоритм генерации перестановки по заданному номеру и алгоритм нумерации перестановки. Для этого рассмотрим рис. 2.1, на котором показан фрагмент дерева для получения перестановок множества символов {a, b, c}.

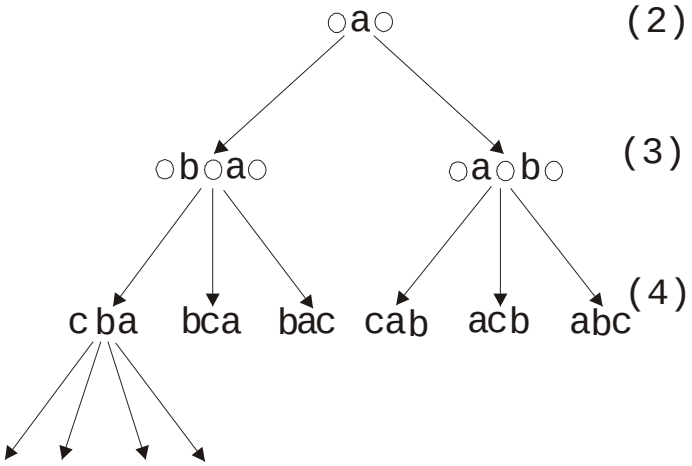


Рис. 2.1. Дерево для построения перестановок

Корень этого дерева содержит первый символ «a», при этом слева и справа от этого символа записаны кружки, которые показывают, где может находиться следующий символ «b» относительно «a». Выходные дуги из корня показывают варианты перехода (их всего два). Символ «c» для каждого варианта может находиться в трех позициях, отмеченных кружками. Соответственно из каждой вершины «ba» и «ab» следует по три дуги. Это дает 6 вариантов перестановок. Построение дерева перестановок можно продолжить для 4 элементов. Из каждого узла, содержащего 3 элемента, будет выходить по 4 дуги. Вариантов перестановок будет 24.

Задавая путь в таком дереве, можно получить некоторую перестановку. Алгоритм построения перестановки по дереву основан на операциях сдвига и вставки. Берется первый элемент и ставится в позицию 1, берется *i-й* элемент и определяется его позиция *j (0<j<i)*. Все элементы, стоящие справа от *j-го* элемента сдвигаются на одну позицию вправо, сдвиг начинается с *j-го*. На место *j-го* элемента становится данный *i-й* элемент.

Рассмотрим теперь вопросы генерации перестановки по ее номеру. Пусть задано множество $E = \{a_1, a_2, a_3, ..., a_n\}$ и некоторое целое число *k*, удовлетворяющее условию

$$0 < k \le n!$$

Необходимо найти *k-ю* перестановку. Для того чтобы найти заданную перестановку, необходимо по числу *k* найти позицию каждого элемента a_i и использовать вышеизложенный алгоритм

построения перестановки. Номера позиций вычисляются следующим образом:

$p_1 = 1$ для элемента a_1;

$p_2 = k \bmod 2$ для элемента a_2;

$k = k / 2$

$p_3 = k \bmod 3$ для элемента a_3;

$k = k / 3$

...

$p_n = k \bmod n$ для элемента a_n.

$k = k / n$

Тогда алгоритм генерации *k-й* перестановки множества $E = \{a_1, a_2, a_3, ..., a_n\}$ будет следующий:

1) i=1 В элемент массива *v[i]=* a_1*;*

2) если i>n конец;

3) вычислить $p_i = k \bmod i$, *k=k/i;*

4) выполнить операцию сдвига вправо в массиве *v* относительно позиции p_i;

v[p_i *]=* a_i*;*

5) *i=i+1* перейти на шаг 2.

Назовем этот алгоритм ***GPermutation.***

Ниже приведены функции для реализации предложенного алгоритма генерации:

```
void Insert(int v[],int n, int i, int a){
  for(int j=n-2; j>=i; j--) v[j+1]=v[j]; //сдвиг
  v[i]=a; //вставка
}
void GeneratePermutation(int v[], int n, int k){
 int pos;
 v[0]=1;
 for(int i=2; i<=n; i++) {
   pos=k%i;
   k=k/i;
   Insert(v,i,pos,i);
  }}
```

2.3.1. Алгоритм идентификации перестановки

Пусть задана перестановка P, необходимо поставить в соответствие некоторый номер. Этот алгоритм можно построить на основе алгоритма генерации, используя обратный механизм сдвига влево и удаления. Алгоритм идентификации следующий.

Дан массив *v[n]*, в котором записана перестановка (см. алгоритм ***GPermutation***).

Необходимо найти номер *k*:

1. *i=n, k=0*.
2. Если *i* равно 1, то конец.
3. Ищем в массиве v элемент, содержащий *i*.
4. Полученный индекс присваиваем *j.*
5. Вычисляем *k=k*(i+1)+j*.
6. Сдвигаем влево на одну позицию в массиве *v* относительно *j*.
7. *i=i-1*.
8. Переход на шаг 2.

2.4. Генерация сочетаний

Сочетанием элементов из $E=\{a_1, \ldots, a_n\}$ по *k* называется упорядоченное подмножество из *k* элементов, принадлежащих *E* и отличающихся друг то друга составом, но не порядком элементов.

Число сочетаний вычисляется по формуле

$$C_n^m = \frac{n!}{(n-m)!m!}. \qquad (2.3)$$

Существует множество различных алгоритмов генерации сочетаний, ниже будет описан алгоритм генерации, основанный на нумерации сочетаний. Для этого запишем следующие тождества:

$$C_0^0 = C_k^0 = C_0^k = C_k^k = 1; \qquad (2.4)$$

$$C_n^m = C_{n-1}^m + C_{n-1}^{m-1}. \qquad (2.5)$$

Применяя тождество (2.5) для заданного сочетания C_n^m, можно построить двоичное дерево (рис. 2.2).

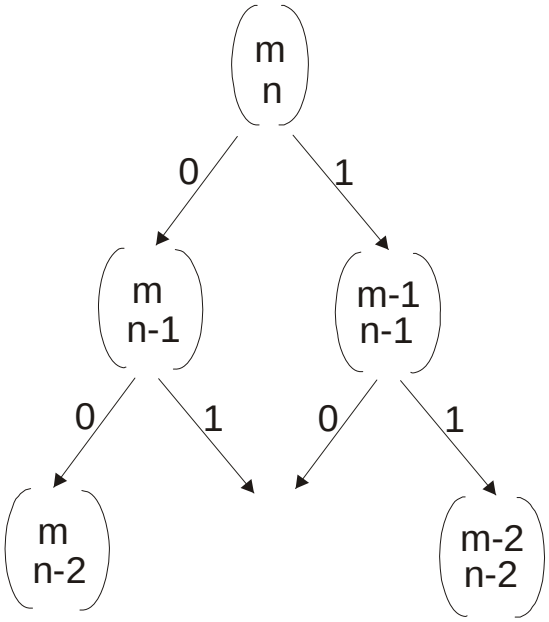


Рис. 2.2. Двоичное дерево декомпозиции сочетания

Дерево строится до тех пор, пока каждый лист этого дерева не станет элементом тождества (2.4). Очевидно, что число листьев такого дерева равно C_n^m. Пример такого дерева записан для представления сочетания C_4^2 на рис. 2.3.

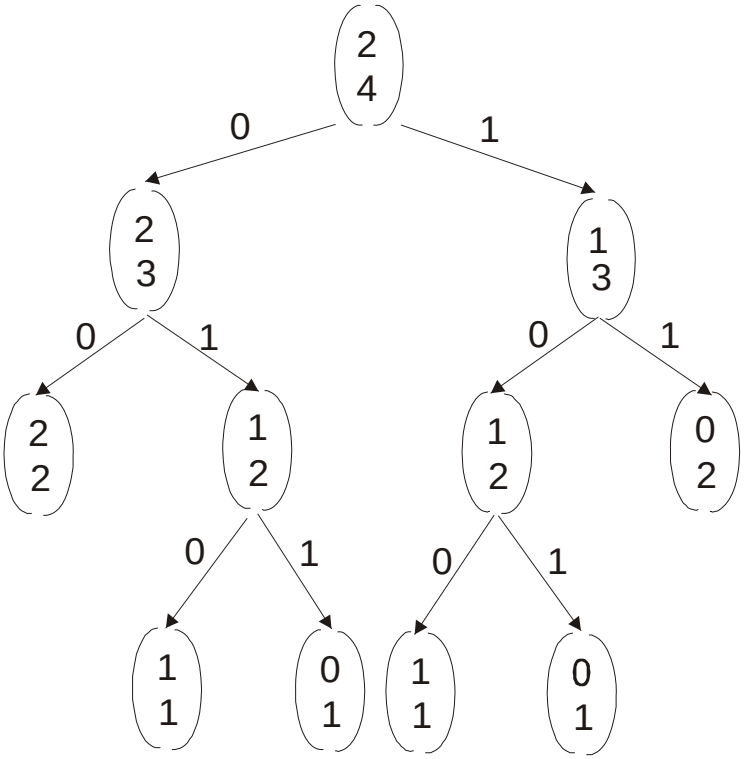


Рис. 2.3. Пример двоичного дерева декомпозиции для сочетания C_4^2

Количество листьев равно 6, что соответствует числу сочетаний. Заметим, что количество путей из корня этого дерева до листьев равно числу сочетаний. Можно попытаться связать конкретный путь в дереве с некоторым элементом сочетания. Для этого пронумеруем выходные дуги: для левой дуги поставим «0», для правой дуги – «1». Тогда, производя левосторонний обход дерева (см. рис. 2.3) и записывая соответствующие цифры для каждого пути, получим

00 (11)
010 (1)
011 (0)
100 (1)
101 (0)
11 (00)

Для первой строчки не хватает двух единиц, для второй – одной единицы, для третьей – нуля, для четвертой – единицы, для пятой – нуля, для шестой – двух нулей. Можно предложить следующее правило дописывания единиц или нулей. Если лист содержит *m,* равное *n*, то справа дописывается *m* единиц. Если лист содержит *m,* равное нулю, то справа дописывается *n* нулей. Используя эти простые правила, можно перечислить все сочетания для заданных значений *m* и *n*.

Теперь построим алгоритм генерации сочетания по трем параметрам , где *m*, *n* – параметры сочетания, а *k* – номер сочетания. Очевидно, что для *k* должно выполняться следующее соотношение:

$$1 \le k \le C_n^m .$$

Необходимо для заданного *k* найти *k-й* путь в двоичном дереве. Решающее правило будет следующее:

1) если $k \le C_{n-1}^m$, то записываем ноль и переходим на рассмотрение узла C_{n-1}^m;

2) если $k > C_{n-1}^m$, то записываем единицу, вычисляем $k = k - C_{n-1}^m$ и переходим на рассмотрение узла C_{n-1}^{m-1};

3) повторяем эти действия, пока не достигнем листа.

Назовем этот алгоритм ***GCombination.***

Тогда можно предложить следующий рекурсивный алгоритм определения сочетания по номеру:

```
int v[100];
int top=0;
void Set(int k,int n,int m)
{
 int b;
 if(m==0){
   for(int i=0; i<n; i++)     v[top++]=0;
   return;
  }
 if(m==n){
  for(int i=0; i<n; i++)     v[top++]=1;
  return;
  }
 b=Comb(n-1,m); //вычисление сочетания C(n-1)^m
 if(k<=b) { v[top++]=0; Set(k,n-1,m); }
 else
  {
    v[top++]=1; Set(k-b,n-1,m-1);
  }
}
```

2.4.1. Алгоритм идентификации сочетания

Алгоритм идентификации данному конкретному сочетанию ставит в соответствие номер. Этот алгоритм также основан на использовании представления сочетания в виде двоичного дерева. Имея двоичный вектор *v*, который описывает сочетание, легко вычислить соответствующий его номер. Вычисление производится, начиная с корня дерева, если первая компонента вектора содержит 0, то происходит переход по левой дуге, если компонента содержит единицу, то происходит переход по правой дуге и к текущему значению номера прибавляется число C_{n-1}^{m}, далее происходит корректировка m и n, и процесс повторяется уже для нижестоящего узла. Процесс вычисления завершается, когда бу-

дет достигнут некоторый лист двоичного дерева. Таким образом, натуральное число может быть представлено суммой C_{n-1}^{m}.

Запишем алгоритм нумерации сочетания.

Дано: *m,n* – параметры сочетания, *v[m]* – двоичный вектор, описывающий сочетание.

1. Переменной *num* присвоить 1, i=0.
2. Если *m* равно нулю, завершить процесс вычисления.
3. Если *m* равно *n*, завершить процесс вычисления.
4. Если *v[i]* равно 1, то $num=num+C_{n-1}^{m}$, *i=i+1*, *m=m-1*, *n=n-1*, переход на шаг 2.
5. Если *v[i]* равно 0, то *i=i+1*, *n=n-1*, переход на шаг 2.

```
int SetNum(int n,int m,int v[]){
 int num=1;
 int mm=n;
 for(int i=0; i<mm; i++)
  {
   if(m==0) return num;
   if(m==n) return num;
   if(v[i]==1) {
    num+=Comb(n-1,m);
    n--;
    m--;
   }
   else n--;
  }
}
```

2.5. Генерация размещений

Размещением элементов из множества ***E={a₁,...,aₙ}*** по *k* называется упорядоченное подмножество из *k* элементов, принадлежащих ***E***.

Число размещений обозначают A_n^k и вычисляют по формуле

$$A_n^k = n(n-1)...(n-k+1)$$ или

$$A_n^k = \frac{n!}{(n-k)!}. \qquad (2.6)$$

Легко заметить, что

$$A_n^k = \frac{n!}{(n-k)!} = \frac{n!k!}{(n-k)!k!} = C_n^k P_k. \qquad (2.7)$$

Таким образом, число размещений из *n* по *k* есть число сочетаний из *n* по *k*, умноженное на число перестановок размером *k*. Используя это свойство, можно предложить следующий алгоритм генерации размещений.

Дано: множество элементов, представленное вектором *v[n]*, *к* – количество элементов в размещении, i – номер размещения, для которого выполняется условие $0 \le i < A_n^k$.

Необходимо получить i-е размещение.

1. Вычисляем номер сочетания $j = i \bmod C_n^k$.
2. Вычисляем номер перестановки $l = i / C_n^k$.
3. Генерируем сочетание *z=GCombination(v, n, k)*.
4. Генерируем перестановку *s=GPermutation(z, k).*
5. Получаем s-i-е размещение из *n* элементов по *k*.

2.6. Генерация подмножеств с заданным нижним и верхним числом элементов

Пусть необходимо получить подмножество множества E_n, мощность которого не менее *k* и не более *m*. Причем $0 \le m \le k \le n$. При *k* равном *m* это будет просто сочетание, и количество подмножеств можно вычислить по формуле сочетаний C_n^k. Если *m* и *k* разнятся на единицу, то необходимо найти сумму сочетаний $C_n^m + C_n^k$, в общем случае количество подмножеств можно подсчитать по следующей формуле:

$$G_n(k,m) = \sum_{i=m}^{k} C_n^i. \qquad (2.8)$$

При *m=0* и *k=n* формула (2.8) приводит к известному тождеству:

$$\sum_{i=0}^{n} C_n^i = 2^n .$$

Алгоритм генерации следующий.

Дано: множество E_n, m – нижняя граница, k – верхняя граница и целое число j – номер подмножества. Для j справедливо неравенство

$$0 \le j < G_n(k,m) .$$

Необходимо построить искомое подмножество.

1. Для $i = m$, k найти такое значение, чтобы выполнилось неравенство

$$j < \sum_{h=m}^{i} C_n^h . \qquad (2.9)$$

2. Используя алгоритм генерации сочетаний, получить сочетание ***GCombination(n,i).***

3. Полученное сочетание будет являться искомым подмножеством.

2.7. Деревья И/ИЛИ

2.7.1. Основные понятия и определения

Деревом будем называть ациклический ориентированный граф *G*, обладающий следующими свойствами:

1) имеется узел r, у которого нет входных дуг, назовем его корнем дерева;

2) все узлы, кроме корня, имеют одну входную дугу и несколько выходных дуг (выходные дуги могут отсутствовать).

Узел, у которого нет выходных дуг, назовем листом.

Сыном узла *z* будем называть узел *w*, который имеет входную дугу выходящую из *z*. Потомком узла *z* будем считать узел *w*, к которому есть путь от *z* по выходным дугам. Предком узла

w будем называть узел z, от которого имеется путь по выходным дугам.

Поддеревом дерева G будем называть дерево, получаемое из G путем отсечения некоторого подмножества входных дуг.

Деревом И/ИЛИ является дерево, которое имеет два типа узлов (И-узел и ИЛИ-узел). Впервые понятие И/ИЛИ дерева появилось в работе Слайгла [43]. Или-узел означает, что задача может быть решена, если решать задачу 1, или задачу 2, или задачу 3 (рис 2.4).

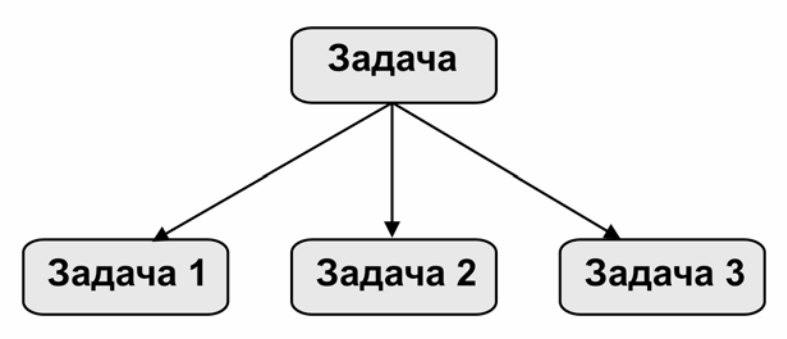


Рис. 2.4. Узел ИЛИ

И-узел означает, что задачу можно разложить на подзадачи и, решив все задачи из этого списка, получить решение исходной задачи (рис. 2.5).

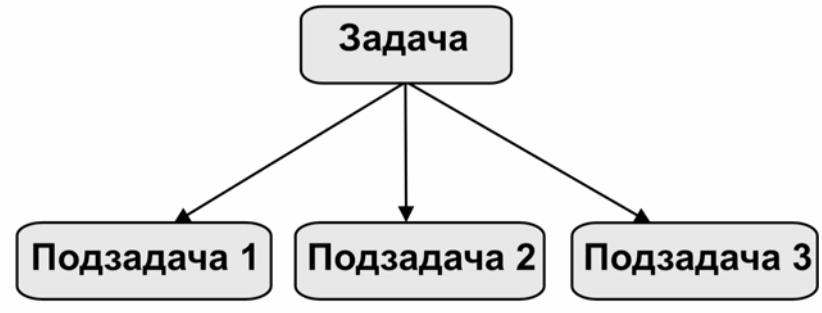


Рис. 2.5. Узел И

Вариантом дерева И/ИЛИ назовем поддерево, которое получается из заданного путем отсечения выходных дуг кроме одной, у всех ИЛИ-узлов. Вариант в терминах решения задачи задает вариант решения задачи.

Деревья И/ИЛИ получили распространение в исследованиях по искусственному интеллекту [44-47]. Ниже предлагаются эффективные алгоритмы для перечисления вариантов в дереве И/ИЛИ [14].

2.7.2. Представление деревьев

Скобочная запись представления дерева осуществляется следующим образом: записывается название узла, затем в скобках записываются сыновья. Например:

Задача (Подзадача 1, Подзадача 2, Подзадача 3).

Вложенность узлов может быть любой. Например:

```
1(1.1(1.1.1,1.1.2(1.1.2.1,1.1.2.2),1.1.3)),
  1.2,
  1.3(1.3.1,1.3.2)
 )
```

Здесь узлы дерева обозначены кодом Дьюи.
Ниже записана грамматика для скобочной записи дерева:

```
<R>=>r(<S><Next>
<Next>=>,<S><Next>
<Next>=>)
<S>=>s
<S>=><R>
```

Запись И/ИЛИ дерева похожа на предыдущую, отличие состоит в том, что для записи И-узла используются круглые скобки, а для ИЛИ-узла – фигурные. Например:
({Александр,(Александр,Сергеевич),А.)},Пушкин).
Здесь записаны следующие варианты:
Александр Пушкин
Александр Сергеевич Пушкин
А. Пушкин
Ниже записана грамматика для скобочной записи дерева И/ИЛИ:

(1) `<R>=>r(<S><ANext>`
(2) `<R>=>r{<S><ONext>`

(3) ***<ANext>=>,<S><ANext>***
(4) ***<ANext>=>)***
(5) ***<ONext>=>,<S><ONext>***
(6) ***<ONext>=>}***
(7) ***<S>=>s***
(8) ***<S>=><R>***

2.7.3. Представление дерева в виде древовидного списка

Другим важным представлением дерева в памяти компьютера являются древовидные списки. Каждый узел дерева представлен элементом списка, который содержит, в простейшем случае, два указателя: указатель на список сыновей и указатель на правого брата. Указатели представлены стрелками (рис. 2.6). Отсутствие указателей обозначено символом заземления.

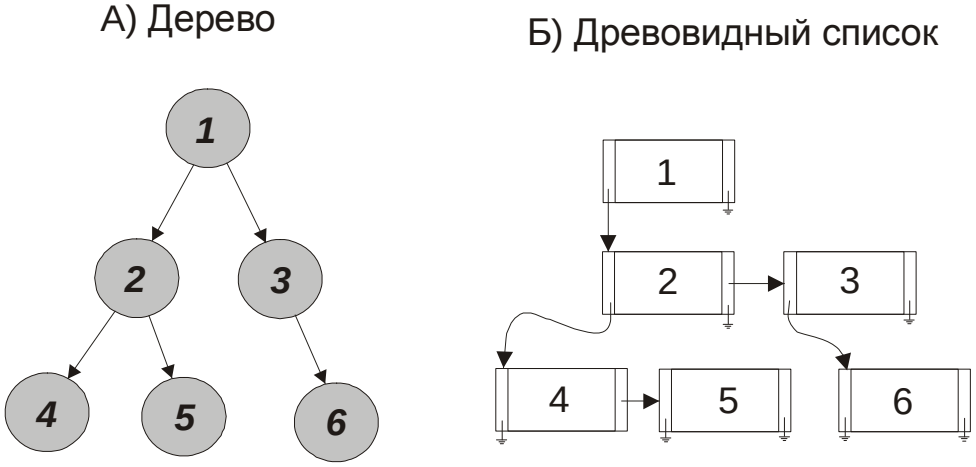


Рис. 2.6. Представление дерева в виде древовидного списка

2.7.4. Алгоритм подсчета вариантов

Рассмотрим алгоритм подсчета вариантов решений в дереве И/ИЛИ. Для этого запишем следующую рекурсивную функцию:

$$\varpi(z)=\begin{cases}\sum_{i=1}^{n}\varpi(s_i^z)\ , & для\ \ ИЛИ\ \ узла;\\ \prod_{i=1}^{n}\varpi(s_i^z)\ , & для\ \ \ И\ \ узла;\\ 1\qquad\quad , & для\ \ \ листа,\end{cases} \tag{2.10}$$

где z – рассматриваемый узел дерева;
$\{ s_i^z \}$ – множество сыновей узла z;
n – количество сыновей.

Подсчитав значение функции для корня дерева, можно получить общее число вариантов подстановок, имеющихся в данном дереве. При этом будет подсчитано количество вариантов подстановок для каждого узла всего дерева. Пример подсчета вариантов показан на рис. 2.7.

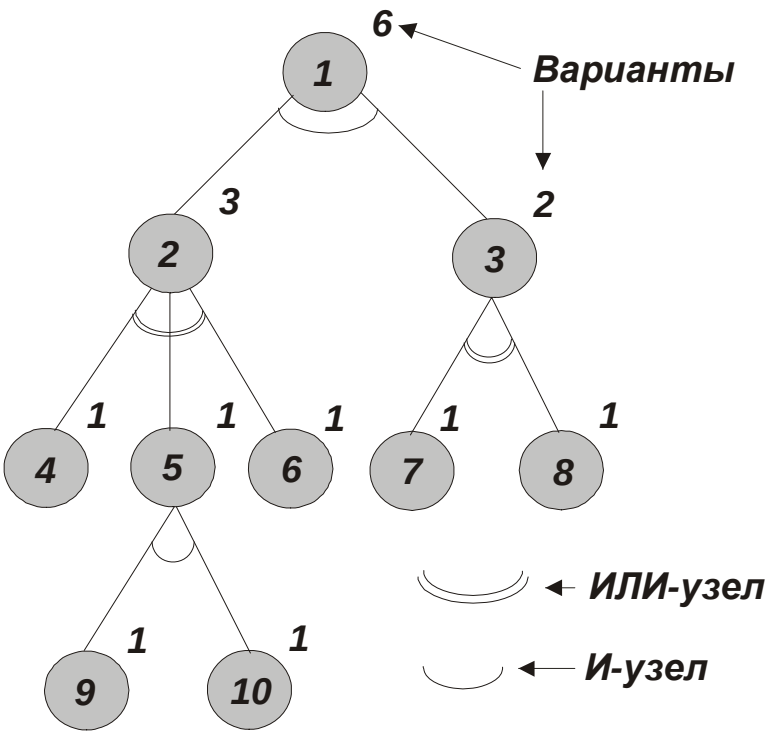


Рис. 2.7. Пример подсчета вариантов в И/ИЛИ дереве

2.7.5. Алгоритм нумерации вариантов

Далее, используя значения функции для каждого узла, можно построить алгоритм нумерации вариантов. Этот алгоритм по номеру из данного дерева получает необходимый вариант подстановки. Предварительно зададим две функции нумерации, которые по номеру варианта для рассматриваемого узла вычисляют номера вариантов для сыновей. Для И-узла будет следующая функция:

$$L_i^s(L^z)=\begin{cases}(L^z / \prod_{j=1}^{i}\varpi(s_j^z)) \bmod \varpi(s_i^z)\ , & i>1;\\ L^z \bmod \varpi(s_i^z)\ , & i=1,\end{cases} \qquad (2.11)$$

где z – рассматриваемый узел дерева;

$\{s_i^z\}$ – множество сыновей узла z;

$\omega(S_i^Z)$ – количество вариантов в поддереве с узлом S_i^Z.

Примечание. При выполнении операции деления используются алгоритмы целочисленной арифметики.

Рассмотрим пример использования этой функции. Пусть задан И-узел z, у которого имеется три сына (S_1,S_2,S_3), и известно количество вариантов $\omega(z)=6$, $\omega(S_1)=1$, $\omega(S_2)=3$, $\omega(S_3)=2$.

Тогда можно записать следующую таблицу значений номеров вариантов для сыновей в зависимости от номера варианта узла z:

L_z	$L(S_1)$	$L(S_2)$	$L(S_3)$
0	0	0	0
1	0	1	0
2	0	2	0
3	0	0	1
4	0	1	1
5	0	2	1

Для ИЛИ-узла необходимо определить не только номер варианта, но и номер соответствующего сына. Для этого запишем следующую функцию:

$$L^s_{k+1}(L^z) = \begin{cases} L^z \quad при \quad L^z < \varpi(s^z_1), \quad k = 0; \\ \min\limits_k (L^z - \sum\limits_{j=1}^{k} \varpi(s^z_j)) \quad при \quad L^s_{k=1}(L^z) \geq 0, \end{cases} \tag{2.12}$$

где z – рассматриваемый узел дерева;

$\{ s^z_i \}$ – множество сыновей узла z;

$\omega(S^Z_i)$ – количество вариантов в поддереве с узлом S^Z_i.

Рассмотрим пример. Пусть задан ИЛИ-узел с 3 сыновьями и $\omega(z) = 9$, $\omega(S_1) = 3$, $\omega(S_2) = 2$, $\omega(S_3) = 4$,

$$\omega(z) = \omega(S_1) + \omega(S_2) + \omega(S_3).$$

Тогда можно записать следующую таблицу значений для номера сына и номера варианта в зависимости от значения варианта узла z:

L_z	K	L_{k+1}	Сын
0	0	0	S_1
1	0	1	S_1
2	0	2	S_1
3	1	0	S_2
4	1	1	S_2
5	2	0	S_3
6	2	1	S_3
7	2	2	S_3
8	2	3	S_3

Алгоритм определения варианта подстановки по заданному номеру будет следующий.

1. Первоначально производится подсчет количества вариантов для каждого узла дерева.

2. В вариант записывается корень дерева, и он становится текущим узлом.

3. Определяется тип текущего узла. Если это И-узел, то переход на шаг 4, иначе переход на шаг 5.

4. Все сыновья рассматриваемого узла записываются в данный вариант подстановки и для них соответственно с функцией для И-узлов определяются номера вариантов для соответствующих поддеревьев (см. 2.11).

5. Если это ИЛИ-узел, то определяется единственный сын и номер варианта для соответствующего поддерева (см. 2.12 для определения сына и номера варианта для ИЛИ-узла).

6. Происходит переход на рассмотрение следующего узла из списка включенных в данный вариант подстановок.

7. Процесс определения варианта подстановок будет происходить до тех пор, пока не будут достигнуты листья для всех рассматриваемых узлов варианта подстановок.

На рис. 2.8 показаны все варианты для дерева И/ИЛИ, приведенного в разделе 2.7.4.

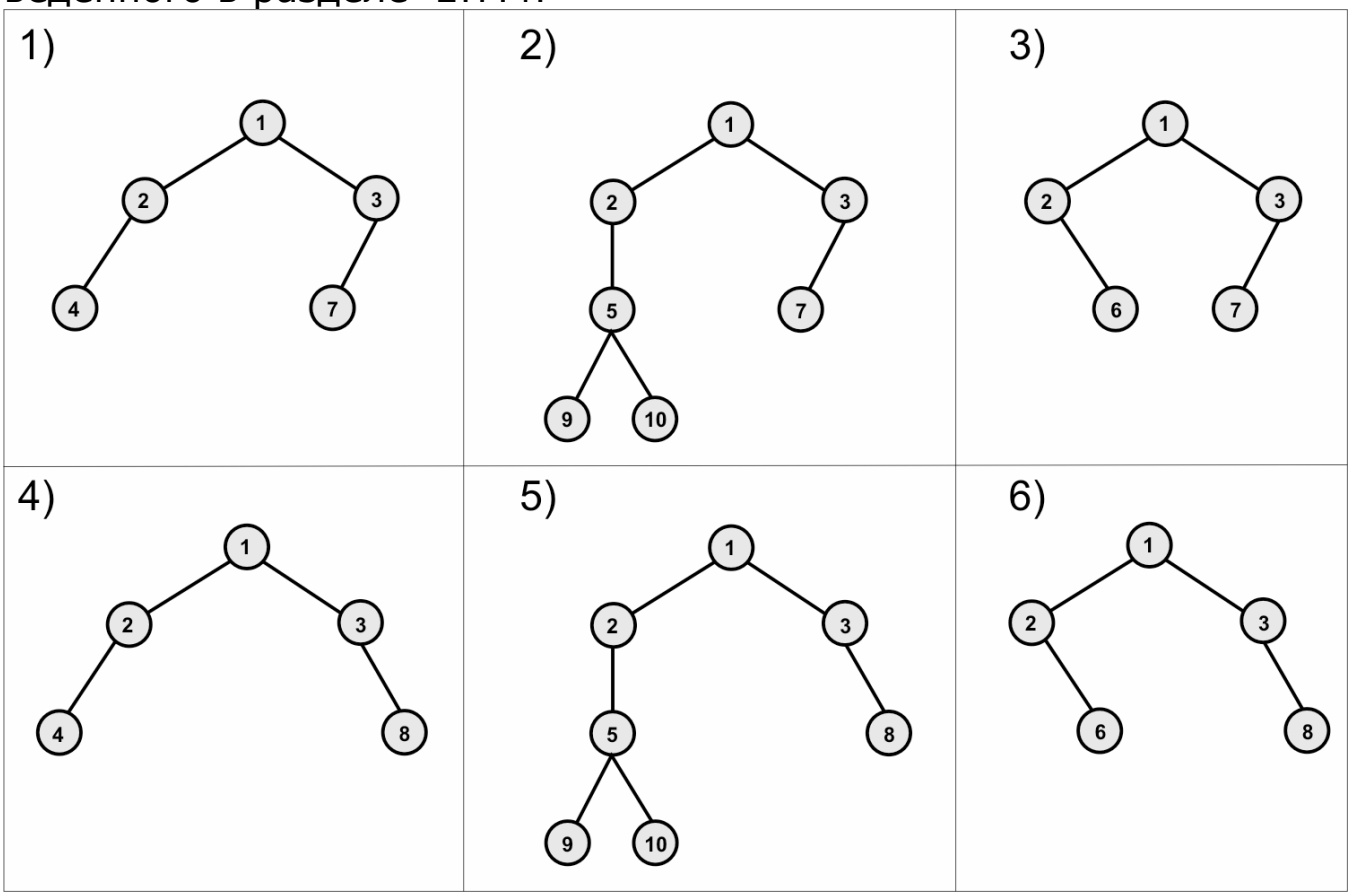


Рис. 2.8. Все варианты для примера И/ИЛИ дерева

2.8. Генераторы, основанные на грамматиках

Аппарат формальных грамматик можно использовать не только для анализа языков, но и для синтеза различных выражений описываемого языка.

Как известно из [48], грамматика задается четверкой (N,T,P,S),

где N – множество нетерминальных символов;
T – множество терминальных символов;
P – множество правил преобразований;
S – начальный нетерминал.

Накладывая различные ограничения на правила преобразований, получаем различные классы грамматик. Имеется классификация грамматик по Хомскому [49].

Рассмотрим простую контекстно-свободную грамматику для записи арифметических выражений:

1. {E,T,D} – множество нетерминалов.
2. {+,-,*,/,(,),x,a,b,c,fun} – множество терминалов.
3. Правила подстановок:

(p1) E→E+T
(p2) E→E-T
(p3) E→T
(p4) T→T*D
(p5) T→T/D
(p6) T→D
(p7) D→(E)
(p8) D→fun(E)
(p9) D→x|a|b|c

Примерами правильных выражений являются:

(1) x+a*b
(2) x/(a+b)
(3) x-fun(a+x/c)

Самый простой путь получения генератора состоит в прямом программировании правил p1-p9. На языке программирования Си это будет выглядеть примерно так:

```
GenerateE(){
 switch(rand(3)){
case 0: GenerateT(); Plus();  GenerateE(); break;
case 1: GenerateT(); Minus(); ParseE(); break;
case 2: GenerateT(); break;
 }
}
GenerateT(){
```

```
 switch(rand(3)){
case 0: GenerateD(); Mult(); GenerateT(); break;
case 1: GenerateD(); Div();  GenerateT(); break;
case 2: GenerateD(); break;
 }
}
GenerateD(){
 switch(rand(3)){
case 0: OpenBra();  GenerateE(); CloseBra();
break;
case 1: Func(); OpenBra();  GenerateE(); CloseBra();   break;
case 2: Name(); break;
 }
}
```

Функция rand(3) – это датчик случайных чисел, который генерирует числа в диапазоне 0,1,2. Остальные функции, тела которых не приведены, обеспечивают печать соответствующих терминальных символов грамматики.

Однако данный подход приводит к проблеме многократной рекурсии, которая в свою очередь приводит к достаточно длинным и однообразным выражениям. Например, многократно применяя правило *p1,* можно получить следующий вывод:

$$E \rightarrow E+T+T+T+T+\ldots$$

Очевидно, что при таком подходе управление генерацией осуществлять невозможно. Первым шагом к управлению генерацией сводится к заданию интервалов ветвления в функциях ***GenerateE(), GenerateT(), GenerateD().***

Возможны следующие варианты введения ограничений:

1) ограничение глубины рекурсии;
2) ограничение размера генерируемой строки символов.

2.8.1. Ограничение глубины рекурсии

Ограничение глубины рекурсии позволяет записать более простой вид грамматики. Например, если глубина рекурсии ограничена не более 2, то можно записать:

(p1) E→E1+T
(p2) E→E1-T
(p3) E→T
(p4) E1→E2+T
(p5) E1→E2-T
(p6) E1→T
(p7) E2→T
(p8) T→T1*D
(p9) T→T1/D
(p10) T→D
(p11) T1→T2*D
(p12) T1→T2/D
(p13) T1→D
(p14) T2→D
(p15) D→(x\|a\|b\|c)
(p16) D→fun(x\|a\|b\|c)
(p17) D→x\|a\|b\|c

Для программной реализации генератора можно в соответствующие функции ввести параметр глубины рекурсии:

```
GenerateE(int deep){
 int k=rand(3);
 if(deep==0) k=2; //запрет на рекурсию
  switch(k){
  case 0: GenerateT(deep); Plus();  GenerateE(deep-1);
break;
  case 1: GenerateT(deep); Minus(); GenerateE(deep-1);
break;
  case 2: GenerateT(deep); break;
 }
}
```

В общем случае управление генерацией можно осуществить заменой датчика случайных чисел на некоторую функцию управления, которая на основе исходной информации о классе

генерируемых выражений и глубине рекурсии строит выражение заданного класса.

2.8.2. Представление грамматик с ограниченной глубиной рекурсии деревьями И/ИЛИ

В общем случае задача нумерации всех выражений языка заданной некоторой грамматикой неразрешима. Однако для конкретного вида грамматики можно найти алгоритмы нумерации [50, 51]. Ниже предлагается метод, основанный на использовании преобразования грамматики в дерево И/ИЛИ и применении соответствующего алгоритма нумерации (см. 2.7.5).

Грамматику с ограниченной глубиной рекурсии можно представить грамматикой без рекурсивных правил (смотри грамматику предыдущего раздела). Тогда можно предложить алгоритм для представления такой грамматики в виде дерева И/ИЛИ.

1. Корнем дерева И/ИЛИ является начальный нетерминал S грамматики G, этот нетерминал заносится в список нетерминалов.
2. Берется нетерминал A из списка нетерминалов, ищутся все правила, левая часть которых является нетерминалом A. Для каждого найденного правила организуется узел P, который станет сыном узла A. Все символы правой части записываются как сыновья узла P. Узел, соответствующий нетерминалу A, становится ИЛИ-узлом. Узлы, соответствующие правилам P, становятся И-узлами. Нетерминал A из списка удаляется. Все нетерминалы, найденные в правой части, записываются в список.
3. Если список нетерминалов не пуст, то переход к шагу 2.

На рис. 2.9 представлено дерево И/ИЛИ для следующего фрагмента грамматики:

(p1)	E→E1+T
(p2)	E→E1-T
(p3)	E→T

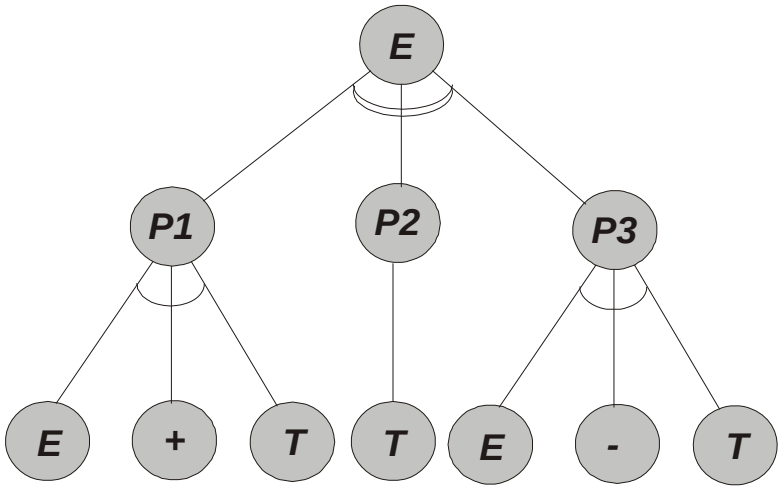


Рис. 2.9. Дерево И/ИЛИ для грамматики

Важно отметить, что дерево И/ИЛИ для ограниченной грамматики содержит все деревья вывода, и вариант дерева И/ИЛИ является деревом вывода.

Рассмотрим оценки размеров деревьев И/ИЛИ для грамматик с ограниченной глубиной рекурсии:

E->E+T
E->E-T
E->T

Количество узлов E в зависимости от глубины рекурсии *n* можно описать рекурсивной функцией

$$Fe(n) = \begin{cases} 1, & n = 0, \\ 2Fe(n-1) + 3, & n > 0. \end{cases} \qquad (2.13)$$

Тогда, исключив рекурсию, получим следующее выражение для функции:

$$fe(n) = 2^{n+2} - 3. \qquad (2.14)$$

Аналогичная формула будет и для правил:

T->T*D

T->T/D
T->D

Количество узлов дерева И/ИЛИ растет по экспоненциальной зависимости от глубины рекурсии.

2.8.3. Метод построения алгоритмов генерации сложных информационных объектов на основе деревьев И/ИЛИ

Пусть имеется некоторая информационная модель ***M,*** описывающая всю совокупность информационных объектов (возможно бесконечную). Запишем основные этапы построения генерирующего алгоритма:

1) проводятся исследования с целью прямого преобразования информационной модели в фиксированное дерево И/ИЛИ;
2) если прямого преобразования не найдено, то проводятся исследования с целью построения алгоритма построения дерева И/ИЛИ;
3) проводятся исследования свойств полученного дерева И/ИЛИ или алгоритма построения;
4) производятся исследования, связанные с удалением бесконечностей (рекурсии) и ограничением глубины рекурсии.

Пробные исследования, проведенные в разделе 2.8.2, показывают, что при глубине рекурсии равной 10, количество элементов в дереве И/ИЛИ будет 8192. Общее число вариантов может превышать 2^{64} вариантов. Поэтому необходимо использовать представление расширенных целых чисел.

Глава 3. МОДЕЛИ И АЛГОРИТМЫ ГЕНЕРАЦИИ ВОПРОСОВ И ТЕСТОВЫХ ЗАДАНИЙ

3.1. Введение

Опыт создания и использования компьютерных контрольных работ и экзаменаторов, накопленный в ТМЦДО, показал [6]:

1) студенты быстро приспосабливаются к небольшому (100 вопросов) экзамену, заготавливают шпаргалки;
2) происходит простое механическое запоминание ответов на данный вопрос, поскольку в ответ нужно ввести конкретное число или выбрать конкретный вариант;
3) при одновременной сдаче экзамена в компьютерном классе вопросы для разных студентов могут быть одни и те же.

Таким образом, уменьшается эффективность проведения экзаменационных сессий и контрольных точек. Для устранения указанных недостатков предлагается несколько вариантов действий:

1) расширить количество вопросов в компьютерных экзаменационных и контрольных работах, используя имеющуюся технологию;
2) создать новую технологию, базирующуюся на идее использования «генераторов».

Первый вариант решения проблемы имеет ряд существенных недостатков:

1) преподаватель-методист должен записать новые вопросы, это всегда связано с большими затратами;
2) количество вопросов существенно не увеличится, через определенный достаточно небольшой период времени вопросы необходимо будет переписывать заново.

Второй вариант решения проблем основывается на идее создания и использования генератора тестовых заданий и вопросов.

Проблемы построения генераторов уже обсуждались в литературе [14-18, 52-54]. Общие методы построения генераторов описаны в монографии Башмаковых [15]. Ниже предлагается

развить модели и методы генерации вопросов и тестовых заданий на основе алгоритмов перечисления вопросов и тестовых заданий с применением математического аппарата, разработанного в главе 2. С другой стороны, рассматриваются вопросы внедрения генераторов в практику дистанционного контроля знаний.

3.2. Генерация задач

3.2.1. Задачи и их свойства

Обучение решению задач является составной частью обучения практически для любой предметной области. В математике и физике – это обучение решению математических и физических задач. В медицине это может быть обучение постановке диагноза. В радиотехнике – обучение поиску неисправностей в радиоэлектронной схеме и т.д.

Обычно формулировка задачи состоит из двух частей: условие задачи и некоторое требование или вопрос. Условие задачи (или исходные данные) содержит описание некоторой начальной ситуации и может быть озаглавлено словами: ***дано***, ***имеется***. Требование может быть озаглавлено словами: ***найти***, ***определить, чему равняется***. Вместо требования может стоять некоторый вопрос. Такие вопросы начинаются словами: ***почему***, ***сколько***, ***какое*** и т.д.

Для ответа на вопрос или выполнения требования необходимо, как правило, найти некоторую последовательность действий, основываясь на условии задачи и знании приемов, правил и законов данной предметной области. Выполняя эту последовательность действий, в итоге находим решение задачи (выполнение требования ***найти*** или ответ на вопрос). Последовательность действий, приводящая к решению, называется ходом решения задачи или, более точно, алгоритмом решения задачи.

Рассмотрим некоторые общие свойства задач. Прежде всего, это касается необходимых и достаточных условий решения задачи. Необходимыми условиями считаются условия, невыполнение которых приводит к невозможности решения задачи. Достаточными условиями считаются условия, выполнение которых

заведомо приводит к решению поставленной задачи. С методической точки зрения, обучаемому могут быть предъявлены следующие варианты условий:

1) условие задачи не является необходимым;
2) условие задачи является только достаточным;
3) условие задачи и необходимое, и достаточное.

В первом случае, как правило, не хватает некоторых исходных данных, и обучаемый должен определить: во-первых, что задача не может быть решена и, во-вторых, каких исходных данных не хватает. Во втором случае в исходных данных имеется все для решения задачи, а также некоторые другие «мешающие» данные. Здесь обучаемый должен решить задачу и указать на «лишние» данные. В третьем варианте в исходных данных содержатся только те данные, которые необходимы для решения задачи. Почти все учебные задачи имеют необходимые и достаточные условия задачи.

Рассмотрим свойства решения. Возможны следующие варианты:

1) единственное решение;
2) решений несколько;
3) решений бесконечно много;
4) решения нет.

Для алгоритма решения задачи также может быть несколько вариантов:

1) алгоритм решения единствен;
2) имеется некоторое конечное множество алгоритмов;
3) имеется бесконечное множество алгоритмов;
4) алгоритма решения нет.

Необходимо еще раз уточнить: решение – это конечный результат, а алгоритм – некоторая последовательность действий, приводящих к результату. Очевидно, что если нет алгоритма, то и нет самого решения. При одном и том же алгоритме можно получить бесконечное множество решений. А одно и то же решение может быть получено бесконечно большим числом алгоритмов. Этот вопрос разработан в теории алгоритмов [55].

Построение алгоритма решения, как правило, носит поисковый характер. Имея исходные данные, необходимо найти некоторую конечную последовательность действий, приводящих к

решению. Существует два основных множества методов решения:

1) методы прямой волны (решение ищется, начиная с анализа исходных данных);

2) методы обратной волны (решение ищется, начиная с анализа конечного результата).

В первом случае рассуждают, примерно, так. «Имеется то-то и то-то, тогда можно найти нечто», далее, используя это нечто, ищется другое неизвестное и т.д., пока не будет получен искомый результат. Во втором случае поиск решения начинают с конца, здесь рассуждения строятся следующим образом: «чтобы найти конечный результат, необходимо использовать то-то и то-то, для этого необходимо определить такие-то неизвестные», далее определяется последовательность действий для нахождения неизвестных. И этот процесс производится до тех пор, пока все неизвестные не будут принадлежать множеству исходных данных. В некоторых случаях методы обратной волны могут быть неприемлемы, например для решения шахматных задач.

Сам процесс решения задачи, в общем случае, носит переборный характер. Один из основных методов решения задач – это метод «разделяй и властвуй», который заключается в следующем: решаемая задача разделяется на подзадачи, выявленные подзадачи решаются отдельно и решение всех выявленных подзадач дает решение искомой задачи. Для решения подзадач также можно использовать метод «разделяй и властвуй». Иногда для поиска решения используется дерево И/ИЛИ, это дерево строится следующим образом: если задача разбивается на подзадачи, то записывается И-узел; если данная подзадача имеет несколько вариантов, то записывается ИЛИ-узел, который означает, что если решен один из представленных вариантов задачи, то и вся задача решена. В математике иногда используют метод сведения задачи к задаче с известным решением.

3.2.2. Шаблон задачи

Шаблон – это эффективный инструмент символьных преобразований текста. Под шаблоном обычно понимают заготовку текста, в котором некоторые элементы можно изменять в соответствии с заданным алгоритмом. Шаблоны широко использу-

ются в программировании, например, шаблоны в C++, мощный и развитый механизм, на основе которого была развита и реализована идея генерирующего программирования [56].

Под шаблоном задачи будем понимать описание задачи, в котором исходные данные и/или часть задачи могут меняться. Рассмотрим эту идею на конкретном примере. Пусть имеется задача: ***У Пети было два яблока, а у Васи три. Сколько яблок было у Пети и Васи?***

Для того чтобы сделать из этой задачи шаблон, необходимо вместо конкретных чисел поставить параметры и алгоритмы, генерирующие значения этих параметров. Тогда эта задача может быть записана как: ***У Пети было gen(x) яблока, а у Васи gen(y). Сколько яблок было у Пети и Васи?***

Здесь:

1) ***gen(x)*** – программа, генерирующая значения для переменной x;
2) ***gen(y)*** – программа, генерирующая значения для переменной y.

В тестовых системах наряду с формулировкой конкретной задачи необходимо иметь правильное решение задачи или правильный ответ. Поэтому к шаблону нужно приложить программу решения задачи по сгенерированным параметрам. Тогда шаблон задачи будет выглядеть следующим образом:

Правильный ответ (rez=solv(x,y)),

где ***solv(x,y)*** – программа вычисления правильного ответа.

При формулировке конкретного вопроса студенту программа случайно выбирает число для *x*, далее случайно выбирает число переменной *y*, вычисляет правильный ответ и далее подставляет полученные числа *x* и *y* в задачу и выводит эту конкретную задачу студенту.

Если параметр *x* может принять 20 различных значений, а параметр *y* – 30, то общее число вариантов задач такого класса будет 600. Это уже достаточно большая выборка.

Рис. 3.1. Структура шаблона задачи

Для шаблона задачи необходимо (рис. 3.1):

1) выбрать некоторую задачу и выделить множество параметров, которое будет генерироваться;
2) записать алгоритм решения;
3) для каждого параметра записать множество изменения, это может быть список значений, интервалы или список интервалов;
4) для каждого параметра записать алгоритм генерации значения;
5) записать варианты формулировок задач. В некоторых случаях формулировка задачи может измениться в зависимости от значений параметров (в нашем примере: 11 яблок, но 2 яблока);
6) записать алгоритм формулировки задачи.

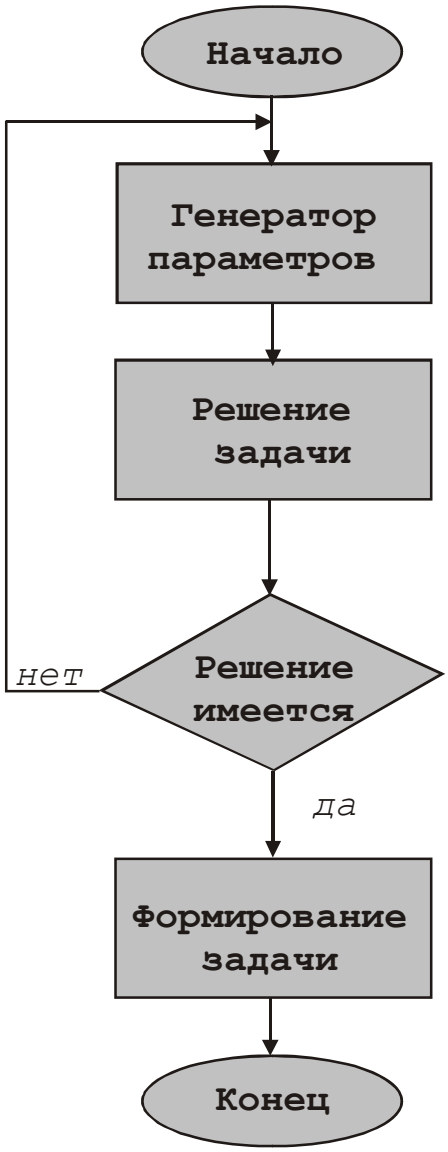


Рис. 3.2. Обобщенный алгоритм работы генератора задачи на основе шаблона

Тогда можно записать обобщенный алгоритм генерации задачи (рис. 3.2). Циклический характер алгоритма заключается в том, что при случайной генерации значений параметров задачи может быть ситуация, когда задача не имеет решения. Тогда процесс поиска значений параметров необходимо возобновить. Например, если в решении задачи встречается выражение

$$\frac{1}{a+b} f(...),$$

то необходимо, чтобы для параметров ***a*** и ***b*** не выполнялось условие ***a = - b.***

Рассмотрим еще пример. Шаблон для квадратного уравнения.

1. Формулировка задачи: ***дано квадратное уравнение*** $ax^2 + bx + c =$ ***0. Найти корни и в ответ ввести найденные значения.***

2. Параметры ***– a,b,c.***

3. Условия решения ***b*b>4*a*c, a не равно нулю.***

4. Значения параметров лежат в пределах ***a=[a1,a2], b=[b1,b2], c=[c1,c2].***

5. Алгоритм решения**:**

x1=(-b+sqrt(b*b-4*a*c))/(2*a);
x2=(-b-sqrt(b*b-4*a*c))/(2*a).

Возможен и такой вариант, когда условия задачи формулируются от значений решения. Для нашего случая генерируем значения корней ***x1*** и ***x2***, далее находим значения параметров ***a,b,c***. В данном случае диапазоны изменения ***x1*** и ***x2***. Записать выражения для нахождения ***a,b,c***. Если однозначного решения нет, то записать условия генерации для подмножества параметров. Для нашего примера будет

$$\begin{cases} ax_1^2 + bx_1 + c = 0, \\ ax_2^2 + bx_2 + c = 0. \end{cases}$$

Варьируя параметром ***c*** и решая систему линейных уравнений относительно ***a*** и ***b***, находим значения параметров ***a,b,c***. В этом случае при генерации значений ***x1***, ***x2*** и ***c*** необходимо учитывать условие решения системы линейных уравнений.

В некоторых случаях представление в шаблоне обратной задачи имеет меньшую трудоемкость для реализации алгоритма генерации.

В тех случаях, когда алгоритм задачи может быть достаточно сложным, для решения задачи можно предложить табличный способ, который заключается в следующем: заранее заготавливается таблица значений параметров и решений, где одна строка содержит значения параметров и правильные решения. Например, для примера с квадратным уравнением:

A	**b**	**C**	**x1**	**x2**
2	**3**	**-8**	**1**	**2**
3	**4**	**-9**	**2**	**3**

Тогда программа будет просто случайно выбирать номер строки и подставлять значения параметров и соответствующие решения. Однако такой подход имеет явно ограничения. Количество вариантов зависит от количества строк в такой таблице.

Рассмотрим некоторые примеры шаблонов задач для генератора заданий в компьютерном экзамене по математике, разработанные профессором кафедры математики ТУСУРа, Л.И. Магазинниковым.

Пример 1. Вычислить определитель 5-го порядка.

В общем случае для формулировки данной задачи необходимо сгенерировать соответствующую матрицу, дискриминант которой должен быть не равен нулю. Далее, используя алгоритм вычисления определителя, найти его значение. Использовать эту задачу для тестирования студентов автор нашел нецелесообразным и предложил вариант матрицы специального вида:

$$\begin{vmatrix} 0 & b & c & a_1 & a_2 \\ 1 & x & y & z & u \\ 0 & b & c+1 & a_3 & a_4 \\ 0 & b & c+2 & a_5 & a_6 \\ 0 & b & c+3 & a_7 & a_8 \end{vmatrix}$$

Для такого вида матрицы определитель считается по следующей формуле:

$$D = -b((a_5 + a_1 - 2a_3)(a_8 + a_4 - 2a_6) - (a_7 + a_3 - 2a_5)(a_6 + a_2 - 2a_4)), \quad (3.1)$$

где генерируются параметры ***b, c, x, y, z, u, {a},*** причем параметры ***c, x, y, z, u*** могут быть любыми, ***b*** не равно нулю. Множество параметров генерируется из условия не равенство нулю определителя. Зная свойства разложения матрицы при вычислении определителя, сравнительно легко найти решение поставленной задачи. Тогда алгоритм генератора будет следующий:

Шаг 1. Генерируем любые значения для параметров ***c, x, y, z, u***.

Шаг 2. Генерируем значения для ***b*** и множества параметров ***{a}.***

Шаг 3. Вычисляем значение ***D*** по формуле (3.1).

Шаг 4. Если ***D*** равно нулю, то повторяем шаг 2.

Шаг 5. Формируем задание с полученными параметрами и записываем правильное решение.

Этот пример показывает, что алгоритм решения задачи может быть существенно упрощен. Количество вариантов заданий в этом примере является огромным числом, даже при таком упрощении. Например, если каждый значимый параметр будет иметь по 10 значений, то общее число вариантов может быть порядка 10^9.

Рассмотрим пример шаблона задачи из компьютерного экзамена по курсу «Магнитные элементы», разработанного профессором кафедры «Промышленная электроника» ТУСУРа В.П. Обрусником:

«Определить плотность тока в обмотках МЭ (усредненную) j (А/мм²) при $K_{ЗК} = 0{,}35$ ***и параметрах катушек в следующей таблице:***

$V_k \times 10^{-4}, \text{м}^3$	$\Delta P_k, \text{Вт}$	$\rho_к \times 10^{-8}, \text{Ом} \cdot \text{м}$
p_1	p_2	p_3

Значение для параметра p_1 случайно выбирается из интервала {2,1;3,4}, для p_2 – *{0,3;15} для* p_3 – *{10,1;45,5}.* Формула для расчета ответа:

$$j = \sqrt{\frac{\Delta P}{V_к \cdot \rho_к \cdot K_{ЗК}}}.$$

Пример шаблона задачи для контрольной работы по циклу «Цифровые и микропроцессорные устройства» разработан профессором кафедры «Промышленная электроника» ТУСУРа А.В. Шараповым.

«Оценить время (мкс) выполнения фрагмента программы микроконтроллером семейства МК51 при частоте кварца Z МГц:

```
        MOV   R1,
              #M
M1:     MOV   R2,
              #N
        DJNZ  R2, $
        DJNZ  R1,
              M1
```

В условие задачи вставляются $Z=12/K$, M и N, причем переменные случайно выбираются из диапазона допустимых значений $K=1,2,3,4,6,12$; $M=\overline{(5,255)}$; $N=\overline{(5,255)}$. Ответ вычисляется по формуле $X=K(1+3M+2MN)$. Нетрудно подсчитать, что генерируется 378 тысяч вариантов данного вопроса, отличающихся исходными значениями численных величин.

3.2.3. Генерация вопросов на основе алгоритмов

Алгоритмы являются универсальными инструментами описания деятельности в некоторой предметной области. Существует достаточно много различных способов представления алгоритмов: описание по шагам; различные блок-схемы; графовое преставление; описание алгоритмов в виде программ на некотором языке программирования и др.

Огромное количество алгоритмов имеет свойство цикличности: повторять некоторую последовательность действий (шагов), пока не наступит некоторое условие. Основываясь на этом свойстве, можно построить генератор вопросов. Для этого дадим ряд определений.

1. Телом цикла назовем последовательность шагов, которая должна многократно выполняться в цикле.
2. Итерацией цикла назовем однократное выполнение тела цикла.
3. Номер итерации цикла номер по порядку итерации.
4. Состояние итерации – значение всех переменных, связанных с данным циклом.
5. Начальное состояние – значения переменных цикла до первой итерации.
6. Условие завершения – условие, выполнение которого завершает работу цикла.

Рассмотрим простейший пример нахождения суммы натурального ряда:

шаг 1: $i=0, S=0$,

шаг 2: $S=S+i, i=i+1$,

шаг 3: если $(i<n)$, *то перейти на шаг 2.*

Вопрос 1. Какое значение примет переменная S после завершения цикла, если *n=генерировать();*

Вопрос 2. Какое значение переменной n было установлено, если по завершению цикла значение *S =генерировать().*

Вопрос 3. Сколько итераций было выполнено, если по завершению цикла значение *S =генерировать().*

Вопрос 4. Какое значение примет переменная *S* на итерации *i= генерировать().*

Вопрос 5. На какой итерации значение *S* станет равным *k (k= генерировать()*).

Рассмотрим пример шаблона для контрольной работы по курсу «Информатика. Часть 3», разработанного профессором кафедры «Прикладной математики и механики» ТУСУРа С.В. Тимченко.

«*Дан целочисленный массив, длиной 8. Массив необходимо отсортировать по возрастанию. Массив дан после n шагов пузырьковой сортировки. Введите через пробел элементы массива после n+1 шага.*

n изменяется от 1 до 5, первые n элементов уже отсортированы. Элементы массива случайным образом задаются в диапазоне от 1 до 99».

Ниже приведен алгоритм сортировки на языке Паскаль.

```
{сортировка пузырьковым методом}
procedure Bubble;
var
    i,j: index;
    x: item;
begin
    for i := 2 to n do
    begin
        for j := n downto i do
            if a[j-1].key> a[j].key then
```

```
            begin
                x := a[j-1];
                a[j-1] := a[j];
                a[j] := x;
            end;
    end;
end; {конец сортировки пузырьковым методом}
```

В приведенном выше шаблоне меняются следующие параметры: входной вектор и номер итерации. Количество конкретных тестовых заданий будет вычисляться по формуле

$$v = 99^8 \cdot 5 .$$

Это позволяет практически каждому студенту получить индивидуальное задание.

Пример шаблона из тренажера по курсу «Алгоритмические языки и технология программирования», разработанный автором:

Дан следующий фрагмент программы

```
char *Skip(char *ptr) { while(*ptr==' ') ptr++; return ptr;}
void Wfunc1(char *b, char *e) { while(b!=e)
printf("%c",*b++); printf("\n"); }
void Word(char *exp){
 char *beg, *end=exp;
 do {
  end=beg=Skip(end);
  while(*end) if(*end!=' ') end++; else break;
  if(end!=beg) Wfunc1(beg,end);
 } while(*end);
}
```

Функция ***Word*** последовательно печатает подстроки символов, разделенных пробелами. Задавая строку символов, содержащую разнообразные слова, разделенные пробелами, можно формулировать следующий вопрос:

«Что будет напечатано в третьем вызове функции WFunc1 при вызове Word("x y z !!!");

Правильный ответ (z)»

3.3. Шаблоны для вопросов типа меню

В тех случаях, когда задачи с параметрами представить нельзя, например для гуманитарных дисциплин, предлагается разрабатывать генераторы для меню вопросов.

Рассмотрим построение генераторов для меню вопросов. В таком генераторе предлагается, что заранее заготовлены формулировка вопроса и два множества: правильных и неправильных вариантов. Важно, что эти два множества достаточно велики. Тогда конкретный меню вопрос можно построить следующим образом:

1) выдается формулировка вопроса;
2) из множества правильных вариантов выбирается подмножество;
3) из множества неправильных вариантов выбирается подмножество;
4) все записывается в конкретный вопрос и задается студенту.

Тогда, если множество правильных вариантов имеет 5 вариантов, множество неправильных имеет 6 вариантов, то общее количество вопросов, в случае, когда из каждого из множеств выбирается по два варианта, будет равно

$$n_{\text{var}} = C_5^2 C_6^2 = 150,$$

где C_m^k – число сочетаний из m элементов по k.

Как видно из формулы, количество вариантов может быть достаточно велико. Общая схема построения такого типа вопроса может быть следующая:

1) производится описание некоторой исходной ситуации;
2) записывается множество правильных утверждений относительно этой ситуации;
3) записывается множество неправильных утверждений относительно этой ситуации.

Рассмотрим несколько примеров вопросов.

Пример 1

Формулировка: *укажите верные утверждения для сказки «Колобок»*

Множество верных утверждений:

1) колобок испекли дед и баба;
2) колобок убежал от зайца;
3) колобок убежал от волка;
4) колобок убежал от медведя;
5) колобок пел песню;
6) колобка съела лиса;
7) колобок испекли из муки.

Множество неверных утверждений:

1) колобок съели дед и баба;
2) заяц пел песню колобку;
3) колобка съел волк;
4) колобок убежал от лисы;
5) колобка съел медведь;
6) колобок всего один раз пел лисе.

Тогда конкретные вопросы могут быть следующие:

Вариант 1

Укажите верные утверждения для сказки «Колобок»:

1) колобок испекли дед и баба;
2) колобок убежал от зайца;
3) колобок съели дед и баба;
4) заяц пел песню колобку.

Вариант 2

Укажите верные утверждения для сказки «Колобок»:

1) колобка съел волк;
2) колобок убежал от зайца;
3) колобок убежал от волка;
4) колобок убежал от лисы.

Всего вариантов может быть (7*6).2*(6*5)/2= 315 вариантов.

Пример 2

Формулировка: *среди предложенных укажите {фрукты, овощи}*

Множество для представления фруктов:

1) яблоки;
2) груши;
3) слива;
4) абрикос;
5) персик.

Множество для представления овощей:

1) огурец;
2) капуста;
3) тыква;
4) редис;
5) арбуз;
6) помидор.

Пример 3

Отметить команды микроконтроллера МК51, при трансляции которых используется прямой адрес байта

Список верных ответов:		Список неверных ответов:	
PUSH	*PSW*	*MOV*	*R*5, *A*
POP	*DPH*	*CLR*	*C*
MOV	*SBUF, A*	*CLR*	20
DJNZ	*P*1, *M*1	*ANL*	*C,* 20
XRL	*B,* #20	*MUL*	AB
INC	*TH*0	*DIV*	*AB*
ORL	TMOD**,** #0FH	*INC*	*DPTR*
MOV	*B,* @*R*0	*LCALL*	*TIME*
ANL	20, A	*MOV*	*DPTR,* #1000
CJNE	A, SP, M*2*	*AJMP*	*BEGIN*

В условие задачи вставляется 10 альтернатив (пять верных и пять неверных), которые случайным образом выбираются из предложенных списков, причем случайным образом изменяется и последовательность записи альтернатив в задании. Тогда общее число вариантов данного типа вопроса может быть $C_{12}^5 \cdot C_{12}^5 = 627294$.

3.4. Основные элементы модели предметной области для генерации вопросов

Модель предметной области (МПО) является важным элементом исследования различных программных систем [57]. В общем случае реализация МПО представляется в виде некоторой базы знаний [58, 108]. Ниже рассматриваются модели МПО, основанные на сравнительно простых структурах: таблицы, двудольные графы и деревья. Можно предложить следующую общую схему построения генератора:

1. Выбор подходящей модели или синтез модели предметной области.
2. Разработка алгоритма генерации вопросов.
3. Исследование генератора:

– подсчет количества вариантов;
– разработка алгоритмов нумерации вариантов;
– оценивание качества генерируемых вариантов;
– определение множества допустимых вариантов.

3.4.1. Управление генерацией

Алгоритм генерации на основе нумерации позволяет организовать учет и управление процессом генерации. Для хранения вопросов в протоколах можно просто хранить соответствующий номер и при необходимости сгенерировать его, используя соответствующий алгоритм генерации.

С другой стороны, можно гарантированно получить разные вопросы для некоторой группы студентов, находящихся в одном компьютерном классе.

В тех случаях, когда сгенерированные вопросы являются по каким-либо причинам неудачными, то можно задать множество исключений, в котором записаны номера таких вопросов, и не генерировать вопрос с таким номером. Это решается в процессе эксплуатации генератора.

3.4.2. Генерация вопросов для языка предметной области

Владение понятийным аппаратам изучаемой дисциплины является важнейшим элементом знаний студента. Обычно для описания понятийного аппарата предметной области используются толковые словари. Словарь представляется набором статей, каждая из которых содержит понятие и синонимы с кратким определением этого понятия. Методы построения словарей можно найти в работе [59]. Ниже рассматривается словарь как модель генератора вопросов.

Для проверки владения понятийным аппаратом изучаемой дисциплины можно предложить следующий генератор вопросов.

Словарь записывается в виде таблицы:

Термины	Определения
Термин *1*	Определение *1*
Термин *2*	Определение *2*
…	…
Термин *n*	Определение *n*

Используя эту таблицу, можно получить следующие типы вопросов:

1. Тип вопроса:

Укажите термин для следующего определения: «Определение i-го термина».

Варианты:

1. *Термин i.*
2. *Термин j.*
3. *Термин k.*
4. *Термин l.*

где *i, j, k, l* – номера строк словаря, причем не равные между собой.

2. Тип вопроса:

Укажите определение для следующего термина: « i-й термин».

Варианты:

1. *Определение i .*
2. *Определение j.*

3. Определение k.
4. Определение l,

где *i, j, k, l* – номера строк словаря, причем не равные между собой.

3. Тип вопроса:

Введите термин для следующего определения: «Определение i-го термина».

Пример 1

Дан фрагмент словаря

Байт	*Ячейка памяти размером в 8 бит*
Слово	*Ячейка памяти, с которой оперирует аппаратная часть вычислительной машины*
Регистр	*Ячейка памяти процессора*
Адрес	*Номер ячейки памяти*
Блок	*Фрагмент памяти, превышающий размер слова*

Тогда можно получить следующие вопросы.

1. Вопрос первого типа:

Байт это:
1) ячейка памяти процессора;
2) фрагмент памяти, превышающий размер слова;
3) ячейка памяти процессора;
4) ячейка памяти размером в 8 бит.
Укажите верный вариант.
Правильный ответ: четвертый вариант.

2. Вопрос второго типа:

Фрагмент памяти, превышающий размер слова, это:
1) байт;
2) слово;
3) блок;
4) регистр.

Правильный ответ: третий вариант (блок).

3. Вопрос третьего типа:

Ячейка памяти, с которой оперирует аппаратная часть вычислительной машины, это . . .
(Введите термин)
Правильный ответ: регистр.

3.4.2.1. Подсчет числа вариантов вопросов

Для подсчета количества вариантов для первого типа вопросов можно предложить следующую формулу:

$$v_1 = nC_{n-1}^{k-1}, \qquad (3.2)$$

где v_1 – количество вопросов первого типа; n – количество статей в словаре; k – количество вариантов в меню.

Например, если $n=10$, $k=5$, то общее число вариантов будет

$$v_1 = 10\frac{9!}{4!5!} = 1260.$$

Для второго типа вопросов формула будет аналогичной:

$$v_2 = nC_{n-1}^{k-1}. \qquad (3.3)$$

Для третьего типа вопросов можно сгенерировать только n вопросов. Тогда общее число вопросов, которое можно сгенерировать предложенным алгоритмом, будет следующее:

$$v = v1 + v2 + v3 = 2nC_{n-1}^{k-1} + n. \qquad (3.4)$$

3.4.2.2. Алгоритм генерации

Дано: словарь объемом в n словарных статей; некоторое число i, удовлетворяющее условию,

$$0 \le i \le v.$$

Необходимо: получить вопрос, если это меню, то количество вариантов должно быть k.

1. Если $i < v_1$, то генерируем вопрос первого типа:

– вычисляем $k = i\%n$ – номер словарной статьи;

– вычисляем *l=i/n* – номер сочетания;

– генерируем *l – сочетание (n-1,k-1)*, используя алгоритм генерации сочетаний (см. раздел 2.4);

– записываем вопрос «*Укажите термин для следующего определения*», вставляем *k-е* определение;

– формируем вектор вариантов ответов, используя вектор сгенерированного сочетания и *k-й* термин. Затем перемешиваем вектор ответов и храним позицию правильного ответа. Вектор сочетания хранит номера словарных статей, вошедших в сочетание с данным номером *l*, при этом необходимо, чтобы *к-я* словарная статья не попала в сочетание, все номера больше или равные *k* увеличиваются на единицу.

2. Если $i < v1 + v2$, то генерируем вопрос второго типа:

– вычисляем $k = i\%n$ – номер словарной статьи;

– вычисляем $l = i/n$ – номер сочетания;

– генерируем *l-сочетание(n-1, k-1)*, используя алгоритм генерации сочетаний ***GCombination*** (см. раздел 2.4);

– записываем вопрос «*Укажите определение для следующего термина*», вставляем *k-й* термин;

– формируем вектор вариантов ответов, используя вектор сгенерированного сочетания и *k-е* определение. Затем перемешиваем вектор ответов и храним позицию правильного ответа. Вектор сочетания хранит номера словарных статей, вошедших в сочетание с данным номером *l*, при этом необходимо, чтобы *k-я* словарная статья не попала в сочетание, все номера больше или равные *k* увеличиваются на единицу.

3. Если $i < v1 + v2 + v3$, то генерируем вопрос третьего типа:

– вычисляем $k = v1 + v2 + v3 - i$ – номер словарной статьи;

– записываем определение и формулируем вопрос;

– в эталонный ответ записываем термин.

3.4.3. Методы генерации, основанные на таблицах

Словарь является частным случаем табличного представления данных. Исследования, связанные с представлением таблиц в памяти компьютера, проводятся в рамках реляционной алгебры [60]. Ниже предлагается реляционная модель представления шаблона и алгоритм генерации вопросов.

Пусть имеется некоторая таблица, в которой систематизированы некоторые знания или факты из конкретной предметной области. Манипулируя полями такой таблицы, можно генерировать следующие вопросы:

1) меню;
2) заполнение пропущенного поля;
3) указание правильного варианта заполнения данного поля (меню вариантов);
4) расстановка перечисленных элементов в пропущенные поля таблицы (упорядочение).

Таблица

поле (1,1)	**поле (1,2)**	**…**	**поле (1,n)**
поле (2,1)	**поле (2,2)**	**…**	**поле (2,n)**
…			
поле (m,1)	**поле (m,2)**	**…**	**поле (m,n)**

Генерация меню вопроса по таблице:

1. Выбрать *k-ю* строку в таблице.
2. Выбрать 2 поля с номерами столбцов *i* и *j*, которые однозначно соответствуют друг другу.
3. Содержимое *i-го* поля поместить в формулировку вопроса, а содержимое *j-го* поля поместить как правильный вариант ответа.
4. Выбрать *l* полей в *j-м* столбце таблицы, причем номера строк соответствующих полей не должны равняться *k*.
5. Перемешать *l* выбранных полей с *j* полем.
6. Сформатировать вопрос.

3.4.3.1. Подсчет числа вариантов меню вопросов

Число вариантов меню вопросов можно вычислить по следующей формуле:

$$v = nm(m-1)C_{n-1}^{l},$$

где n – число строк;
m – число столбцов в матрице;
$m(m-1)$ – задает число размещений по два элемента, задает множество пар столбцов с учетом их следования;
l – число неверных вариантов в меню вопросе;
C_{n-1}^{l} – число сочетаний строк таблицы без выбранной строки, по l.

3.4.3.2. Алгоритм генерации меню вопросов

Пусть есть таблица [n строк, m столбцов], число неверных вариантов в вопросе l и задано некоторое целое число i, задающее некоторый искомый вопрос.

1. Найти номер строки:
   - $k = i \bmod(n)$;
   - $i = i / n.$
2. Найти пару столбцов:
   - $R = i \bmod(m(m-1))$;
   - сгенерировать размещение $A(2,m)$ (см. раздел 2.5);
   - $i = i/(m(m-1))$.
3. Найти сочетание $C_{n-1}^{l}(i)$ на множестве строк, в котором отсутствует k-я строка. Для этого необходимо использовать алгоритм генерации сочетания (см. раздел 2.4).
4. Объединить вектор сочетания со второй компонентой вектора размещения и перемешать.
5. Сформатировать вопрос.

Пример генератора вопросов на основе таблицы.
Пусть дана следующая таблица, описывающая периоды Палеозойской эры [61]:

ПЕРМЬ	**280 млн лет назад**	**Исчезновение трилобитов и примитивных кораллов**
		Расцвет звероподобных пресмыкающихся
КАРБОН	**345 млн лет назад**	**Появление первых голосеменных растений**
		Появление пресмыкающихся
		Первые крылатые насекомые
		Расцвет земноводных
		Первые леса
ДЕВОН	**395 млн лет назад**	**Первые земноводные**
		Развитие различных групп рыб
		Первые насекомые
		Первые древовидные растения
		Примитивная псилофитовая флора
СИЛУР	**445 млн лет назад**	**Появление челюстных рыб**
		Появление наземных членистоногих
		Появление наземных растений
		Многочисленные коралловые рифы
ОРДОВИК	**500 млн лет назад**	**Расцвет головоногих**
		Дальнейшее развитие членистоногих
		Эволюция кишечнополостных
		Расцвет граптолитов
		Бесчелюстные рыбы
КЕМБРИЙ	**570 млн лет назад**	**Появление иглокожих**
		Появление трилобитов
		Развитие беспозвоночных
		Появление сосудистых растений
		Эволюция водорослей

Генерация меню вопросов

Пример 1

Укажите название периода, который происходил 395 млн лет назад:

1) Девон;
2) Кембрий;
3) Карбон;
4) Пермь.

Пример 2

Укажите период, название которого Карбон:
1) 280 млн лет назад;
2) 345 млн лет назад;
3) 500 млн лет назад;
4) 570 млн лет назад.

Пример 3

Какой период характеризуется следующим:

Первые земноводные
Развитие различных групп рыб
Первые насекомые
Первые древовидные растения
Примитивная псилофитовая флора

1) Девон;
2) Кембрий;
3) Карбон;
4) Пермь.

Пример 4

Какая группа характеристик соответствует периоду «Силур»:
1) Первые земноводные.
Развитие различных групп рыб.
Первые насекомые.
Первые древовидные растения.
Примитивная псилофитовая флора.
2) Появление челюстных рыб.
Появление наземных членистоногих.
Появление наземных растений.
Многочисленные коралловые рифы.
3) Расцвет головоногих.

Дальнейшее развитие членистоногих.
Эволюция кишечнополостных.
Расцвет граптолитов.
Бесчелюстные рыбы.
Введите соответствующий номер.

Пример 5

Какая группа характеристик соответствует периоду, который происходил 500 млн лет назад:
1) Первые земноводные.
Развитие различных групп рыб.
Первые насекомые.
Первые древовидные растения.
Примитивная псилофитовая флора.
2) Появление челюстных рыб.
Появление наземных членистоногих.
Появление наземных растений.
Многочисленные коралловые рифы.
3) Расцвет головоногих.
Дальнейшее развитие членистоногих.
Эволюция кишечнополостных.
Расцвет граптолитов.
Бесчелюстные рыбы.
Введите соответствующий номер.

Генерация с пропусками

Пример 1

Введите значение в незаполненное поле (ячейку) следующей таблицы:

Пермь	*Карбон*	*Дивон*	*Селур*
_____	*345 млн лет назад*	*395 млн лет назад*	*445 млн лет назад*

Пример 2
Введите значение в незаполненное поле (ячейку) следующей таблицы:

Пермь	_____	*Дивон*	*Селур*
280 млн лет назад	*345 млн лет назад*	*395 млн лет назад*	*445 млн лет назад*

Генерация с использованием перемешивания

Пример 1

Дана таблица:

Пермь	*Карбон*	*Дивон*	*Селур*
395 млн лет назад	*345 млн лет назад*	*280 млн лет назад*	*445 млн лет назад*

Расставьте периоды в соответствии с названиями.

Пример 2

Даны следующие периоды Палеозойской эры:
1) Карбон;
2) Дивон;
3) Кембрий;
4) Силур.
Записать периоды, начиная с самого раннего (позднего). В ответ ввести последовательность номеров, разделенных пробелами.

3.4.4. Генератор вопросов для некоторой последовательности действий (процесса, технологии)

Пусть дана последовательность $\{x_1, x_2, x_3, ..., x_n\}$, например, некоторая последовательность действий. Назовем эту последовательность процессом, а элементарное действие – этапом (шагом). Для каждого этапа процесса имеется список начальных условий, необходимых для осуществления данного этапа, есть список особых условий завершения этапа процесса. Имеются также параметры выполнения этапа. И по окончании процесса получается некоторый результат *r*. На рис. 3.3 показано графовое представление этапа процесса. Есть начальный набор усло-

вий для выполнения этапа *{u0, u1, u2}*. После выполнения этапа получается промежуточный результат (*r1*).

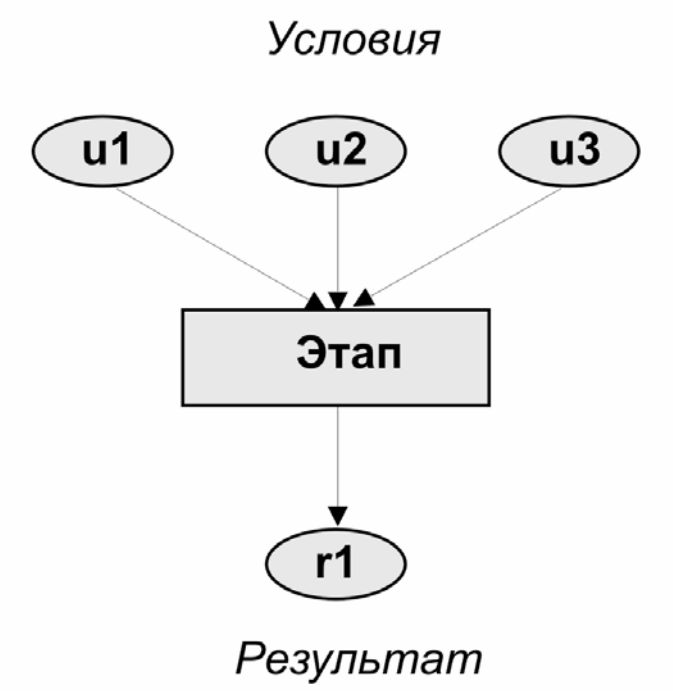


Рис. 3.3. Описание этапа процесса

Тогда весь процесс можно описать с помощью некоторого двудольного графа [62], в котором все вершины разбиваются на два непересекающихся множества. Первое множество – это этапы процесса. Второе множество – условия и результаты этих этапов. Поскольку результаты некоторых этапов могут являться условиями выполнения других этапов, то можно записать двудольный граф (рис. 3.4). Вершины *u1, u2 и u3* необходимы для выполнения первого этапа, вершины *u4* и *r1* являются условиями для выполнения второго этапа, а вершины *r2* и *u5* – для третьего этапа. Для четверного этапа необходимо получить *r3* и *r4* и задать *u6.* Результатом всего процесса будет вершина *r5*.

Примерами описаний подобных процессов могут быть разнообразные технологии приготовления продуктов, химических производств, сборки устройств, разнообразных операций и т.п.

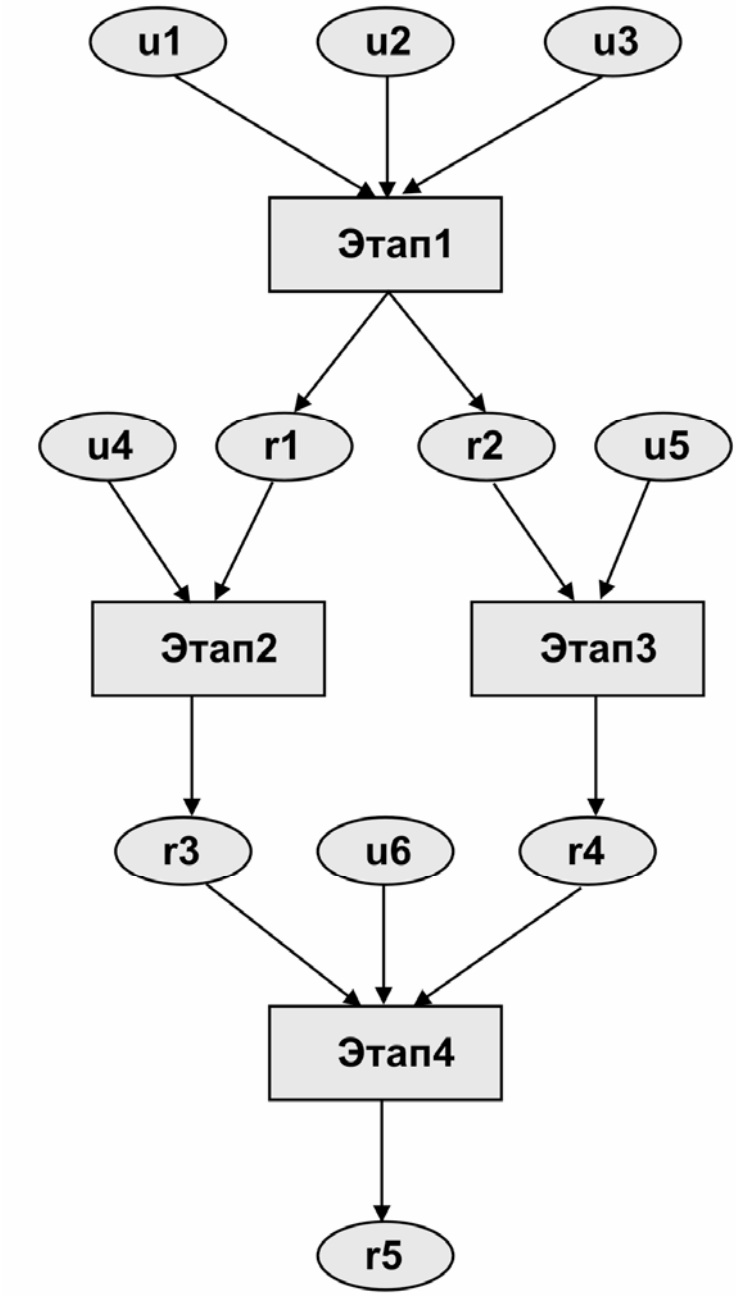


Рис. 3.4. Описание процесса

На основании данной модели процесса можно получать следующие вопросы:

1. Какой этап следует после *x[i]* … (название) или в форме меню { *x[k1], x[k2], x[k3], x[k4]* }.
2. Какой этап следует перед *x[i]* … (название) или в форме меню { *x[k1], x[k2], x[k3], x[k4]* }.

3. Даны следующие этапы процесса { *x[k1], x[k2], x[k3], x[k4]* }, указать правильную последовательность их выполнения.

4. Указать условия, выполнение которых необходимо для осуществления *x[k]* этапа процесса { *u[k1], u[k2], u[k3], u[k4]*}.

5. Дан промежуточный результат *r[k]* выполнения процесса. Указать этап, который получает данный результат { *x[k1], x[k2], x[k3], x[k4]* }.

6. Указать недостающее условия для осуществления этапа *x[k]*
{ *u[k1], u[k2], u[k3], u[k4]*}.

7. Дан список промежуточных результатов {*r[k1], r[k2], r[k3], r[k4]*}, упорядочить их в соответствии с заданным процессом.

Рассмотрим использование данного подхода на конкретном примере. Пусть дана технология (рецепт) приготовления блюда (ботвиньи) [63] (рис. 3.5), представленная в форме двудольного графа. Прямоугольниками отмечены технологические операции (отварить, перебрать, промыть, нарезать и т.д). Овалами изображены ингредиенты и результаты этих операций (рыба, вода, специи, вареная рыба, бульон и т.д.). Стрелка от отвала к прямоугольнику означает, что этот ингредиент необходим для выполнения данной операции. Стрелка от прямоугольника к овалу означает, что данный овал является результатом этой операции.

Тогда можно сформулировать следующие вопросы:

1. *Для того чтобы выполнить операцию «Припустить», необходимо иметь:*

1) бульон;
2) специи;
3) шпинат;
4) смесь.

Укажите верные варианты.

2. *Для того чтобы выполнить операцию «Отварить», необходимо иметь:*

1) сахар;
2) воду;
3) квас;
4) рыбу.

Укажите верные варианты.

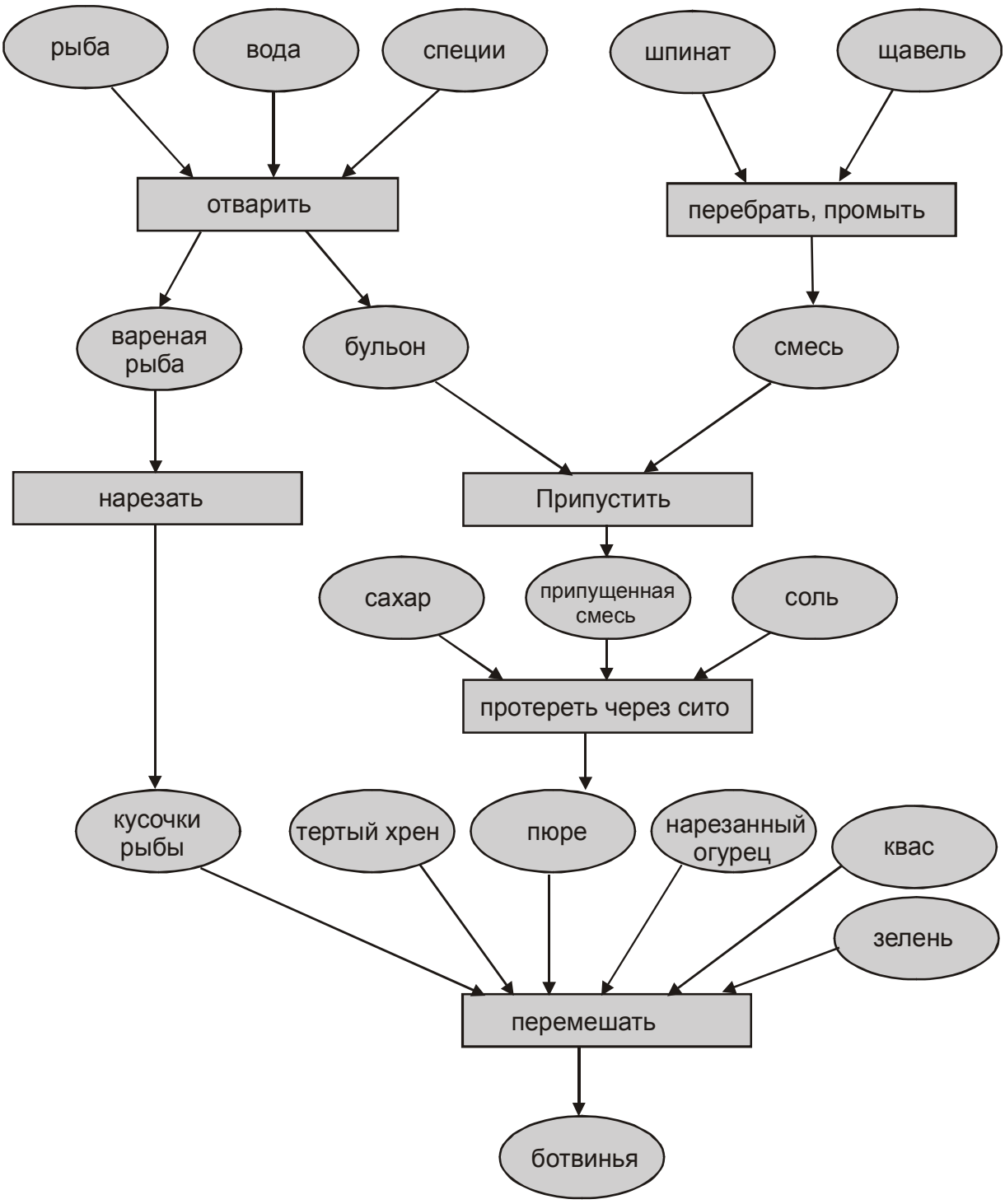


Рис. 3.5. Описание технологии приготовления ботвиньи

3. *Результатом операции «Припустить» будет:*
   1) пюре;
   2) припущенная смесь;
   3) тертый хрен;
   4) нарезанный огурец.

4. *Указать операцию, результатом которой будет бульон:*
   1) нарезать;
   2) перебрать, промыть;
   3) припустить;
   4) *отварить*.

5. *Даны следующие операции:*
   1) припустить;
   2) перебрать, промыть;
   3) перемешать;
   4) протереть через сито.

Укажите порядковыми номерами правильную последовательность выполнения операций.
Правильный ответ 2,1,4,3.

6. *Какие операции необходимо выполнить, чтобы осуществить операцию «припустить»:*
   1) отварить;
   2) нарезать;
   3) ***перебрать, промыть****;*
   4) *протереть через сито.*

7. *Для какой операции необходим полуфабрикат «смесь»:*
   1) отварить;
   2) припустить;
   3) перебрать, промыть;
   4) протереть через сито.

3.4.4.1. Подсчет количества вариантов

Пусть дан двудольный граф, описывающий некоторую технологию *G(U,Z)*, где *U* – множество вершин; *Z* – множество связей. Причем множество вершин *U* разбивается на два непересекающихся подмножества U_1 и U_2. Пусть *m* – мощность множества U_1, *n* – мощность множества U_2. Тогда множество вопросов типа 4 будет вычисляться по формуле

$$v = \sum_{i=1}^{m} (S_i \cdot C_{n-S_i}^3),$$

где S_i – количество узлов, связанных с узлом i входными дугами.

3.4.4.2. Алгоритм генерации вопроса

Пусть дан двудольный граф, описывающий некоторую технологию *G(U,Z)* и некоторое целое число j, удовлетворяющее условию $0 \le j \le v$.

Необходимо получить j-й вопрос заданного типа.

1. Вычисляем k – номер по формуле:

$$l = \min_k (\sum_{i=1}^{k} (S_i \cdot C_{n-S_i}^3) - j) > 0.$$

2. Находим *k-ю* вершину из U_1, определяем подмножество U_2^k вершин имеющие выходные дуги к *k-й* вершине. Удаляем указанное подмножество из U_2:

$$U_{\_} = U_2 - U_2^k.$$

3. Находим номер элемента множества U_2^k – это будет правильный вариант ответа:

$$n_v = l \bmod S_i.$$

4. Находим номер сочетания $n_c = l / S_i$.

5. Генерируем сочетание для множества $U_{\_}$ по 3 с номером n_c, используя алгоритм ***GCombunation.***

3.4.5. Генератор вопросов на основе иерархии (древовидной классификации)

Иерархические конструкции часто применяются для представления знаний некоторой предметной области. Примерами таких конструкций являются различные классификации, родословное дерево, структура некоторой организации, устройства и др. Используя иерархическое представление знаний, можно построить генератор меню вопросов. Идея генератора следующая:

1. Строится некоторое дерево *D*.
2. Случайно выбирается некоторый узел $d_i \in D$.
3. Случайно выбираются два подмножества D1 и D2 других узлов, одно из них удовлетворяет некоторому отношению, второе – не удовлетворяет.
4. Строится меню вопрос: «*Дан узел d, укажите те узлы, которые удовлетворяют (или не удовлетворяют) отношению <формулировка отношения>*».

Далее записываются список *D1* и *D2*.

На основании построенного дерева можно генерировать следующие типы вопросов.

1. Дана последовательность узлов, перечислите те, которые являются сыновьями узла *d*.
2. Дана последовательность узлов, перечислите те, которые относятся к одному уровню иерархии.
3. Дана последовательность узлов, перечислите те, которые принадлежат поддереву узла *d.*
4. Дана последовательность узлов, перечислите те, которые являются листьями.
5. Дана некоторая последовательность *a, b, {x, y, z}, d* (выберете один из узлов x,y,z, который превратит эту последовательность в след дерева).

На рис. 3.6 показана некоторая иерархическая классификация земноводных [63]. Используя эту иерархию, можно генерировать следующие типы вопросов:

Тип вопроса 1

Выбираются случайно элементы иерархии 1.1, 1.2, 1.3. Перечислить все относящиеся к заданной группе (классу).

Даны следующие земноводные:

1) рыбозмеи,
2) углозубы,
3) протеи,
4) свистуны.

Укажите среди них безногих.

Рис. 3.6. Пример иерархической классификации

Тип вопроса 2

Выбирается случайно один из листьев этой иерархии и перечисляются классы. Необходимо выбрать класс для выбранного элемента.

Укажите класс земноводных, к которому относятся рыбозмеи:

1) *безногие;*
2) *хвостатые;*
3) *бесхвостые.*

3.4.5.1. Подсчет числа вариантов вопросов

Приведем подсчет вариантов вопросов для первого типа.

Вариант 1. Среди предложенных узлов указать сына узла *d*. Для подсчета необходимо использовать узлы дерева, не являющиеся листьями. Пусть их количество будет *n*. Общее число узлов *m*. Общее число сыновей будет *m-1* (корень дерева не является сыном). Подсчет числа дуг в дереве даст количество вариантов отношения *«x сын y».* Список узлов, не являющихся сыновьями данного узла размером *k*, можно сформировать на основе сочетания общего числа узлов, не являющихся сыновьями данного узла по *k*:

$$v = \sum_{i=1}^{n} L_i C_{m-L_i-1}^{k},$$

где L_i – количество сыновей *i-го* узла;

$C_{m-L_i-1}^{k}$ – число сочетаний из множества узлов, не являющихся сыновьями *i-го* узла по *k*.

Вариант 2. Среди предложенных узлов указать сыновей узла *d.* В этом случае правильных вариантов может быть несколько. Тогда общее число вариантов можно подсчитать по формуле

$$v = \sum_{i=1}^{n} (2^{L_i} - 1) C_{m-L_i-1}^{k}.$$

При формулировке вопроса для *i-го* узла может быть сформировано $2^{L_i} - 1$ правильных вариантов ответов (формула подсчета множества подмножеств).

Вариант 3. Среди предложенных узлов указать всех сыновей узла *d*. В списке вариантов должны присутствовать все сыновья данного узла, тогда формула будет следующая:

$$v = \sum_{i=1}^{n} L_i C_{m-L_i-1}^{k} .$$

При условии, что альтернатив в меню должно быть фиксированное количество, формула будет немного сложнее.

3.4.5.2. Алгоритм генерации

Дано: дерево, где *m* узлов и *n* узлов, не являющихся листьями, некоторое число j, удовлетворяющее условию

$$0 \le j \le v .$$

Необходимо получить вопрос по типу 1 вариант 2 (это общий случай).

1. Находим такое l, чтобы $j < \sum_{i=1}^{l} (2^{L_i} - 1) C_{m-L_i-1}^{k}$, *l=1,n*; *l*-будет показывать номер узла для формирования меню вопроса.
2. Записать множество сыновей *l*-узла и подсчитать новое значение для *j* по формуле
$$j = j - \sum_{i=1}^{l-1} (2^{L_i} - 1) C_{m-L_i-1}^{k} .$$
3. Получить номер подмножества:
$$j_1 = j \bmod (2^{L_l} - 1) .$$
4. Получить номер сочетания:
$$C_l = j / (2^{L_l} - 1) .$$
5. Сгенерировать подмножество и сочетание, используя алгоритмы генерации из разделов 2.5, 2.7. Сформировать вектор и перемешать его.
6. Записать меню вопрос: «*Из приведенного списка узлов перечислите сыновей l-узла*».
   Узел *k1*, узел *k2*, узел *k3* … узел *Kk.*
7. Записать правильные варианты ответа.

Глава 4. СИСТЕМЫ И ТЕХНОЛОГИИ КОМПЬЮТЕРНОГО ТЕСТИРОВАНИЯ ТМЦДО

4.1. Технология проведения компьютерных экзаменов и контрольных работ

Компьютерный контроль знаний представляет собой реализацию контроля знаний средствами, предоставляемыми персональным компьютером. Для реализации такого контроля знаний необходимо наличие соответствующей программы и персонального компьютера. Назовем эту программу тестирующей.

Тестирующая программа выдает на экран монитора последовательность вопросов, на которые обучаемый отвечает (вводит ответ), используя манипулятор «мышь» или клавиатуру. После этого происходит анализ ответа на вопрос. После ответа на все вопросы производится подсчет правильных ответов и проставляется оценка. Оценка выдается одновременно на экран и заносится в протокол.

В ТМЦДО предусмотрены две формы тестового контроля знаний: промежуточный контроль в виде компьютерной контрольной работы и итоговый (экзамен) [65, 66].

Технология проведения компьютерных контрольных работ следующая:

1. Тестирующая программа контрольной работы высылается студенту вместе с методическими материалами по данной дисциплине.
2. Студент обязан выполнить данную контрольную работу на своем компьютере до определенного учебным планом срока и выслать протокол контрольной работы в центр.
3. В ТМЦДО протокол анализируется и при положительной оценке заносится в ведомость успеваемости.
4. Наличие всех выполненных контрольных работ является условием допуска студента к экзамену.

Технология проведения компьютерных экзаменов следующая:

1. В диспетчерской службе ТМЦДО преподавателю выдается компакт-диск с экзаменами.

2. На месте проведения экзамена преподаватель запускает необходимую тестирующую программу, вводит параметры студента и пароль в диалоговом окне регистрации. После чего программа формирует набор вопросов для данного студента.

3. Студент отвечает на вопросы, используя клавиатуру или манипулятор типа «мышь» компьютера. При этом:

– время экзамена 2 часа по 60 минут;

– студент может просматривать список вопросов;

– студент может изменять ответ на любой вопрос до нажатия клавиши получения оценки;

– если исчерпано время, то программа автоматически ставит оценку;

– после нажатия оценки, программа выведет диалоговое окно для вывода оценки.

4. Студент должен позвать преподавателя, и после ввода пароля выводится оценка студента.

5. Оценка фиксируется преподавателем в экзаменационную ведомость.

4.2. Основные элементы тестирующей программы

Для составления тестирующей программы необходимо составить банк вопросов, которые будут заданы студенту, и предоставить на них варианты правильных ответов. Наличие правильных ответов необходимо для осуществления сравнения с ответами, которые будет вводить обучаемый. Способы формулировок вопросов и формы ввода ответа могут быть разными. Они определяются преподавателем и ограничиваются возможностями инструментальной системы, в которой будет реализована тестирующая программа.

При построении программы необходимо решить следующие вопросы:

1. Формулировка вопросов, выбор способа ввода и анализа ответа.
2. Просмотр вопросов.

3. Генерация последовательности вопросов.
4. Установка времени.
5. Оценивание.
6. Ведение протокола.

Решение вышеперечисленных вопросов возлагается как на разработчиков тестирующей программы, так и на разработчиков методического обеспечения. Необходимо четко определить функции каждой группы и обеспечить их взаимодействие для совместного решения проблем, возникающих в ходе разработки. Методические вопросы создания теста решаются в рамках дидактической тестологии [67, 68]. Ниже рассматриваются технические аспекты создания тестов.

4.2.1. Формулировка вопроса, выбор способа ввода и анализа ответа

Существует достаточно большое разнообразие способов формулировки вопросов. Можно выделить три основные группы:

1. позиционный;
2. альтернативный;
3. инжекторный.

В первом случае ответом на вопрос является один из элементов множества выделенных букв, слов или словосочетаний, который необходимо выделить для осуществления ответа на вопрос. Для этого курсор манипулятора «мышь» устанавливается на выбранную область и нажимается его левая клавиша. При этом соответствующий элемент меняет цвет фона и шрифта. На рис. 4.1 представлен пример вопроса с позиционным выбором без выбранных элементов, а на рис. 4.2 показаны выделенные области с помощью манипулятора «мышь».

Ответом на вопрос альтернативного типа является несколько элементов предлагаемого списка (меню ответов), которые отмечают щелчком левой клавиши манипулятора «мышь» (рис. 4.3). Список возможных вариантов ответов может содержать один и более правильных ответов.

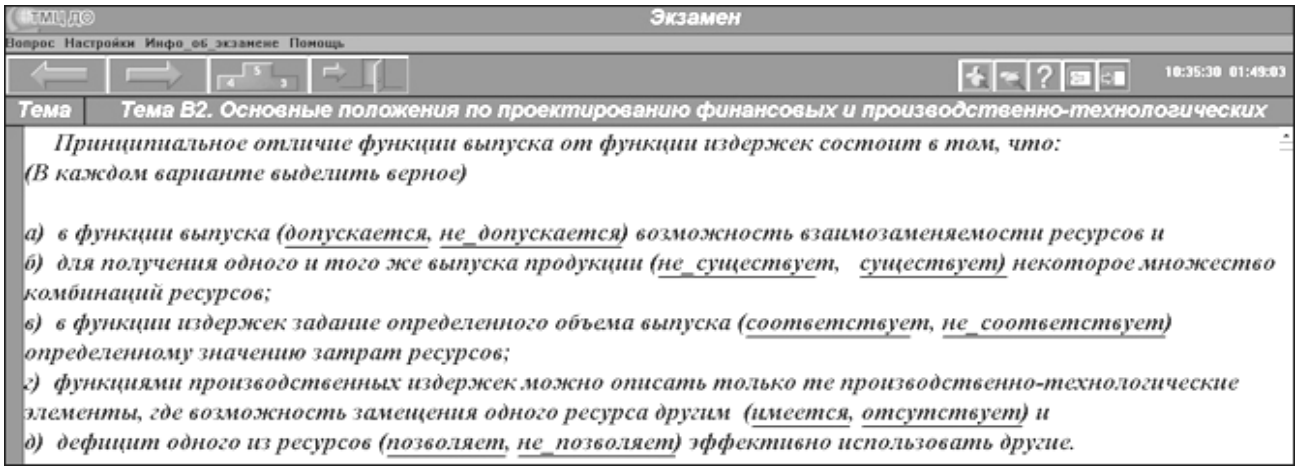


Рис. 4.1. Пример позиционного (меню) вопроса

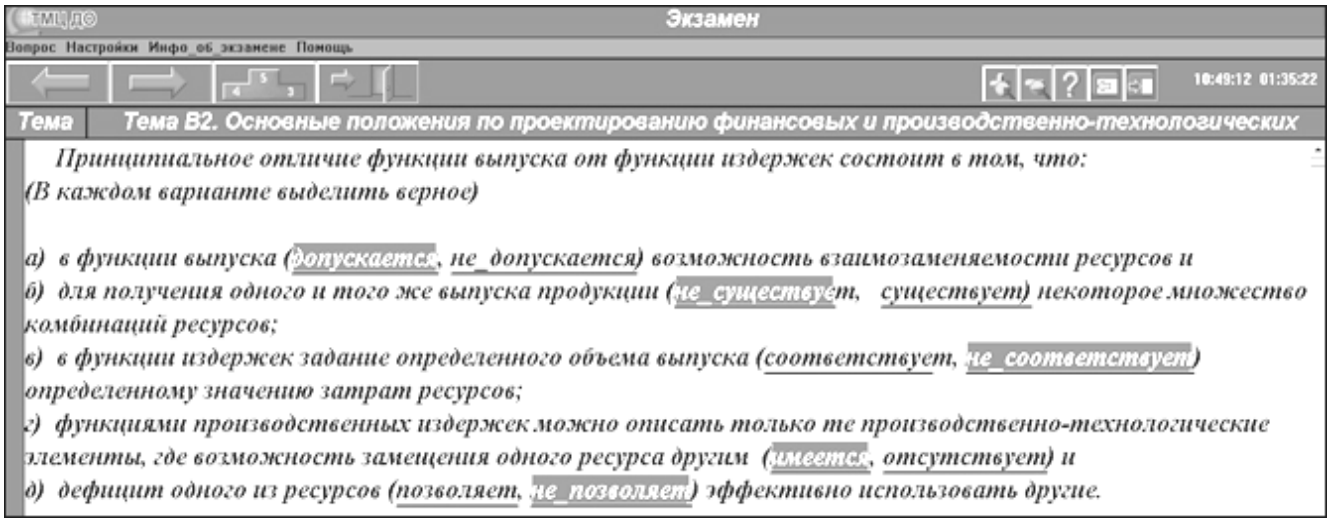


Рис. 4.2. Пример ответа на вопрос позиционного типа

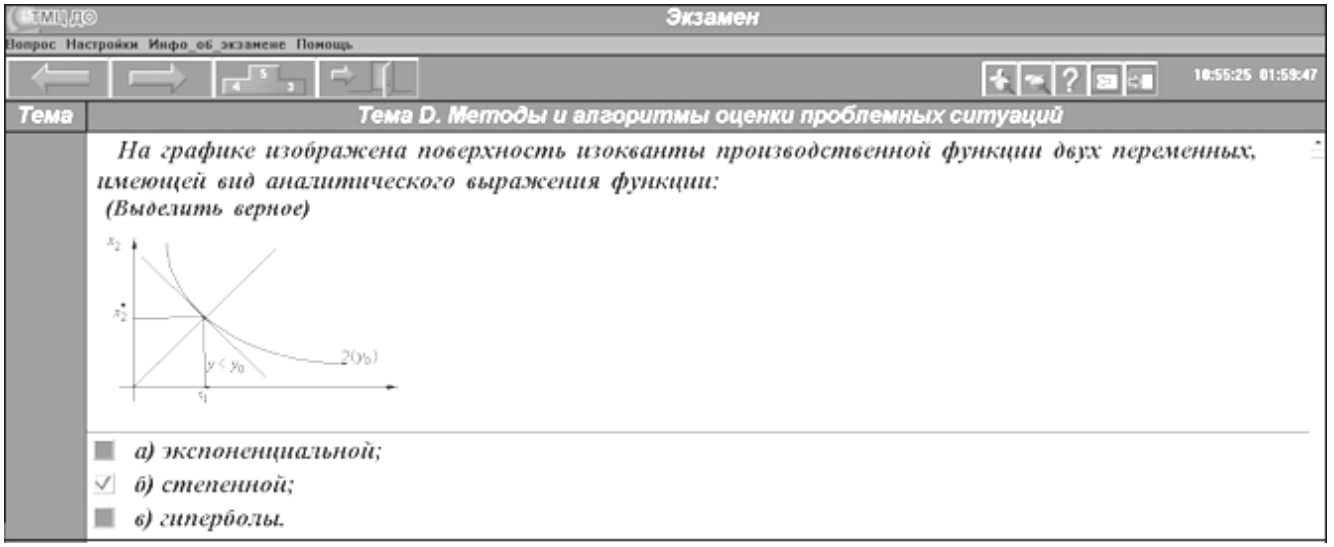


Рис. 4.3. Пример демонстрации выделения правильного ответа в альтернативном вопросе

И в первом и во втором случае ответ задан явно и необходимо совершить выбор из заданного набора.

Ответом на вопрос инжекторного типа является строка символов, набираемая на клавиатуре. Ответ в виде текстовой строки представляет собой последовательность символов, введенных с помощью клавиатуры, это может быть некоторое утверждение, термин, формула, программа и т.д. Строка редактируется с помощью клавиши «BackSpace» при необходимости, после чего нажимается клавиша «Enter». В этом типе вопросов обучаемый может записать правильный ответ различными способами. Скажем, если ему необходимо ввести ответ на вопрос «Кто изобрел?», то он может записать следующие правильные ответы: полное имя, фамилию и отчество; полное имя и фамилию; фамилию и т.д. Ответы могут заноситься строчными и прописными буквами. Поскольку в таблице символов они имеют различный код, то это также может стать источником ошибок. Кроме того, необходимо учитывать, что различные понятия и термины могут иметь синонимы, ответы могут быть записаны в разных падежах и т.д.

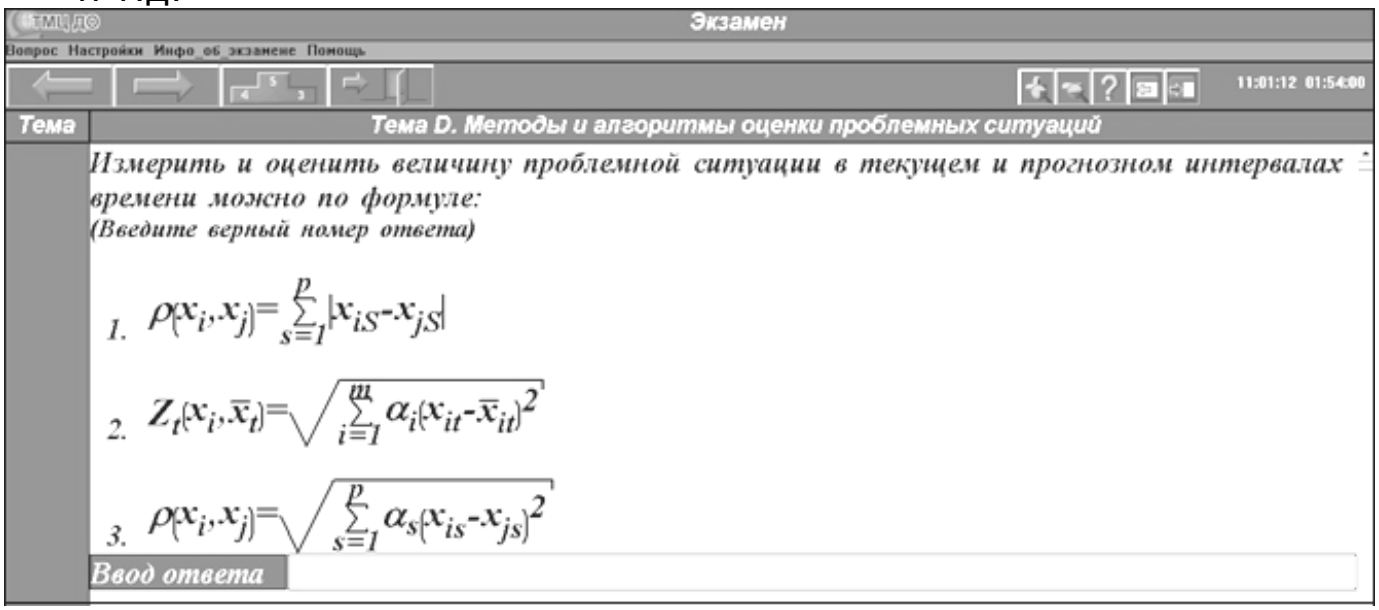


Рис. 4.4. Пример ввода ответа на вопрос инжекторного типа

Простое решение этой проблемы – это запись всех вариантов правильных ответов. Однако в некоторых случаях вариантов может быть достаточно много. Для решения такого рода проблем разработан ряд специальных методов, реализация которых

возлагается на разработчиков тестирующей программы. Пример вопроса показан на рис. 4.4.

4.2.2. Просмотр вопросов

Часто тестирующие программы организуются таким образом, что с вопросом обучаемый знакомится только тогда, когда необходимо уже вводить ответ. В некоторых случаях при проведении тестирования необходимо дать возможность студенту ознакомиться с вопросами заранее, особенно это касается программ, у которых задано время ответа. Существует несколько вариантов организации просмотра вопросов:

1) просмотр вопросов до тестирования;
2) выбор вопроса для ответа при просмотре;
3) смешанные варианты.

В первом варианте студенту дается возможность просмотра множества вопросов перед тестированием, а затем производится тестирование по традиционной схеме: вопрос-ответ. Во втором варианте студенту формулируется вопрос, а затем предлагается выбрать два варианта действия: ответить на вопрос или пропустить его. При пропуске этот вопрос будет задан студенту позднее. Такая организация тестирующих программ характерна при установке времени проведения тестирования. Например, дается один час на 20 вопросов и т.д. При смешанных вариантах предполагается, что все множество вопросов разбивается на подмножества и в каждом подмножестве организован просмотр вопросов. В частности, может быть организован следующий вариант: на экране терминала сразу предъявляется несколько вопросов и студенту предлагается ответить на все данные вопросы, при этом он может выбрать вопросы для ответа из предъявленных.

В инструментальной системе, используемой в ТМЦДО, выбран второй вариант просмотра вопросов: время экзамена фиксировано, студент сам выбирает, на какие вопросы отвечать и когда.

4.2.3. Методы оценивания знаний

Оценивание знаний в обучающих программах производится на основании анализа ответов обучаемого на некоторое множество вопросов или заданий. В простейшем случае каждый ответ оценивается оценкой «правильно» (1), если ответ совпал с образцом, и оценкой «неправильно» (0), если ответ не совпал с образцом. Таким образом, определяя количество правильных ответов, можно судить об уровне знаний обучаемого.

Существует несколько подходов к оцениванию знаний в обучающих программах:

– оценивание на основе идей психологического тестирования [69];

– оценивание с использованием теории размытых множеств [70];

– оценивание на основе статистической теории [71];

– эвристические методы оценивания [72].

Для перехода от оценки в виде количества правильных ответов на некоторое множество вопросов к некоторой бальной шкале (например, от 1 до 5) поступают следующим образом:

1. Определяют максимальное и минимальное возможные числа правильных ответов (очевидно, что максимально возможное число совпадает с числом вопросов, а минимально возможное число правильных ответов равно нулю).

2. Определяют b_1 – число правильных ответов такое, что если обучаемый ответит правильно на меньшее число вопросов, то получит оценку «1». Таким образом, строится интервал оценки «1» – $[0, b_1]$.

3. Определяются остальные числа b_2, b_3, b_4, которые задают границы интервалов соответствующих оценок «2», «3», «4».

4. Тогда для балльного оценивания можно ввести следующую интервальную шкалу:

$[0, b_1]$ – оценка 1;

$[b_1, b_2]$ – оценка 2;

$[b_2, b_3]$ – оценка 3;

$[b_3, b_4]$ – оценка 4;

$[b_4, n]$ – оценка 5,

где n – максимально возможное число правильных ответов.

При установке значений интервалов необходимо учесть следующее соотношение:

$$b_1 < b_2 < b_3 < b_4 < n.$$

На основании вышеизложенного можно записать следующую функцию оценивания:

$$\mu(x) = \begin{cases} 1, & x < b_1, \\ 2, & x < b_2, \\ 3, & x < b_3, \\ 4, & x < b_4, \\ 5, & x \le n, \end{cases}$$

где x – количество правильных ответов;

b_1, b_2, b_3, b_4 – границы интервалов;

n – количество вопросов.

Очевидно, что данный метод оценивания можно применять для разных порядковых шкал. Например, для шкалы «зачет» – «незачет» необходимо установить лишь границу интервала b_1, а для вузовской шкалы оценивания («неудовлетворительно», «удовлетворительно», «хорошо», «отлично») – границы интервалов b_1, b_2, b_3.

В случае, когда общее количество вопросов может изменяться, в функцию оценивания вводят нормировочный коэффициент. Если нормировочный коэффициент равен $1/n$, то нормированная функция оценивания будет следующая:

$$\mu_n(x) = \mu(\frac{1}{n}x),$$

где n – общее число вопросов.

В этом случае границы интервалов b_i будут означать долю правильных ответов от общего числа вопросов. Например, если обучаемый ответил менее чем на 1/3 вопросов, то его оценка «2», менее чем наполовину, то – «3» и т.д.

В некоторых случаях удобно ввести нормировочный коэффициент, равный $100/n$. Тогда функция оценивания будет следующая:

$$\mu_p(x) = \mu(\frac{100}{n}x).$$

В этом случае граница интервалов b_i будет означать процент правильных ответов. Например, если обучаемый ответил правильно менее чем на 30 процентов вопросов, то его оценка «2», если на 50 процентов, то – «3» и т.д. Таким образом, вводя нормирование, добиваются независимости функции оценивания от общего числа вопросов.

В инструментальной системе ТМЦДО используется следующая система оценивания:

1. Вопрос оценивается на основе анализа ответа студента с образцовым ответом. Если ответ правильный, то записывается единица, ели неправильный – ноль.

2. Находится сумма оценок всех ответов.

3. Определяется процент правильных ответов от общего количества заданных вопросов.

4. Определяется интервал, который соответствует вычисленному проценту правильных ответов.

5. Номер интервала соответствует оценке студента.

Стандартные границы интервалов следующие:

1) если процент правильных ответов меньше 50, то оценка «неудовлетворительно»;

2) если процент правильных ответов меньше 75, то оценка «удовлетворительно»;

3) если процент правильных ответов меньше 90, то оценка «хорошо»;

4) если процент правильных ответов больше или равен 90, то оценка «отлично».

При программировании тестирующей программы значения интервалов могут быть изменены.

Возможен также вариант, когда указаны количество правильных вопросов и соответствующая оценка. Например, если правильных ответов меньше 5, то «2»; если 5 или 6, то «3»; если 7, 8 – «4», если 9, 10 – «5».

4.3. Структура системы

Система тестирования уровня знаний является необходимым элементом учебного процесса по дистанционной технологии. В настоящее время имеется огромное число систем тестирования, причем практически каждый вуз имеет свою собственную [73].

Система тестирования ТМЦДО имеет свои особенности, которые выгодно отличают ее от остальных.

1. Система может работать на персональном компьютере и в локальной сети.

2. Система работает в операционных системах Windows 95/98/2000/XP.

3. Система имеет язык описания теста.

4. Система поддерживает работу с генераторами.

5. В системе реализованы внутренние механизмы защиты от несанкционированного доступа.

Система тестирования состоит из двух составных частей:

– подсистемы проведения компьютерных контрольных работ;

– подсистемы приема компьютерных экзаменов.

Обе подсистемы поддерживают язык описания теста и генераторы. Однако форматы кодирования различаются.

Первая система отправляется студенту и должна работать на его компьютере. Вторая работает в представительстве ТМЦДО. Причем в начале сессии она устанавливается, а в конце сессии удаляется. Подсистема проведения экзаменов ставит отметку непосредственно после окончания экзамена. Вместе с тем протокол проведения экзамена кодируется и записывается в базу протоколов. База протоколов создается на каждом локальном компьютере или одна в локальной сети. Эта база может быть использована для контроля.

На рис. 4.5 показана структура системы проведения контрольных работ. Запишем основные элементы этой структуры

1. Модуль регистрации

При запуске системы проведения контрольных работ модуль регистрации ищет регистрационный файл, в котором хранится следующая информация:

– учебное заведение;
– группа;
– фамилия, имя, отчество;
– идентификационный код студента.

Если файл не найден, то подсистема выдает окно для ввода регистрационной информации (рис. 4.6).

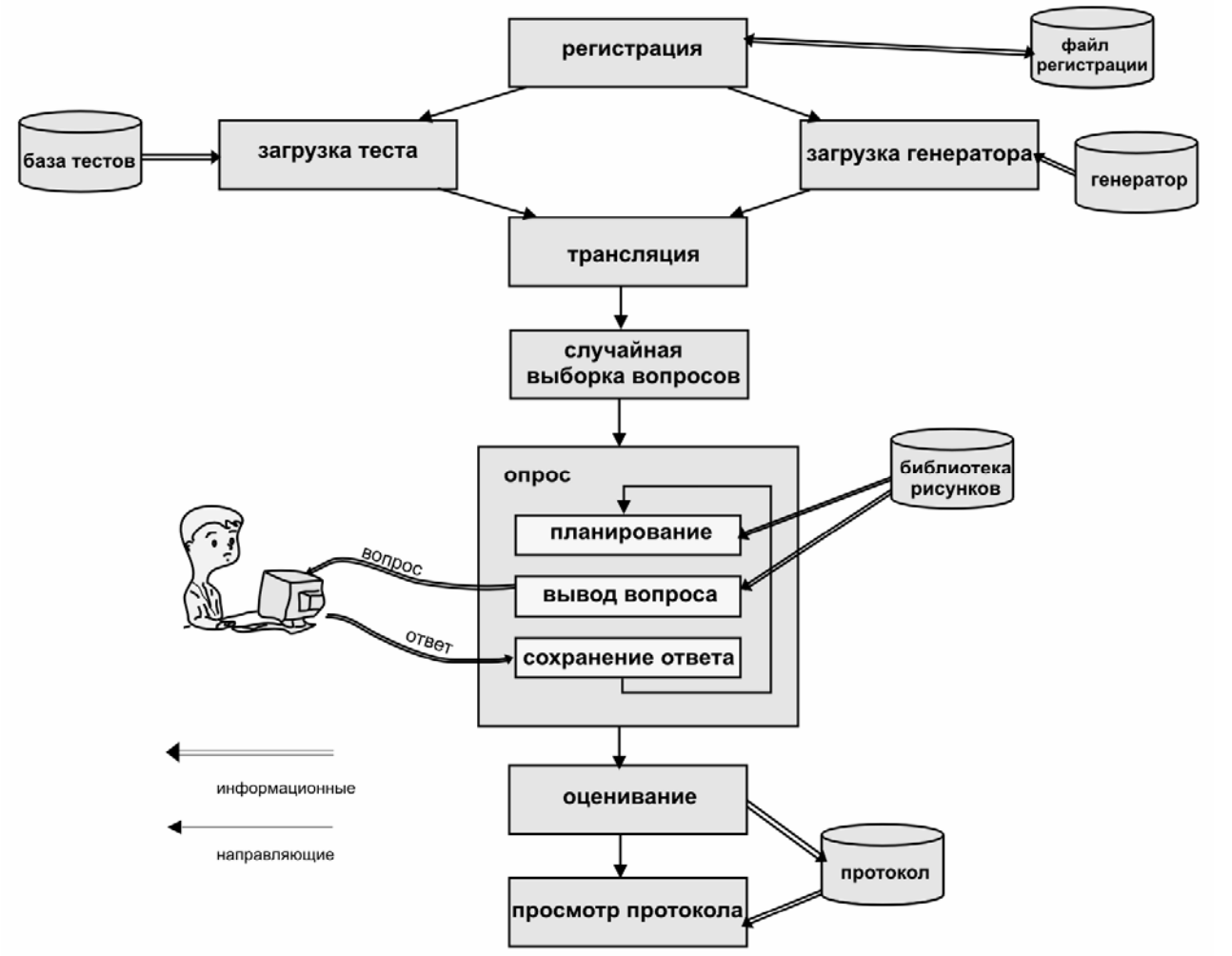


Рис. 4.5. Обобщенная структура системы проведения компьютерных контрольных работ

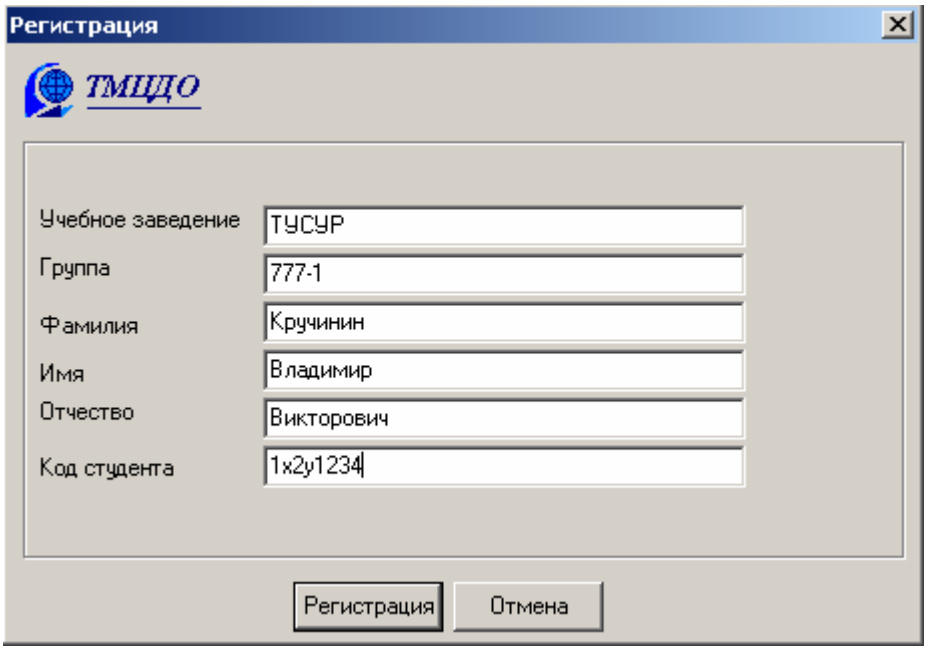


Рис. 4.6. Окно регистрации студента

2. Модуль загрузки теста

Данный модуль производит поиск, загрузку и декодирование файла, содержащего необходимый тест. Имя файла теста должно быть записано в командной строке при запуске системы. Если файл не буден найден, то модуль выдаст соответствующее сообщение об ошибке.

Способ кодирования файла может отличаться от версии системы. В настоящее время имеется несколько вариантов кодирования файла теста.

3. Модуль загрузки генератора

Данный модуль производит поиск, загрузку DLL, генерацию теста в формате системы. Имя файла DLL должно быть записано в командной строке при запуске системы. Если файл не буден найден, то модуль выдаст соответствующее сообщение об ошибке. Если загрузка прошла успешно, то производится генерация теста. После генерации модуль сохраняет параметры генерации теста в специальную структуру.

4. Транслятор

Транслятор производит синтаксический анализ описания теста и преобразование его во внутренний формат системы. Алгоритм разбора основан на методе рекурсивного спуска [74]. При обнаружении ошибки транслятор выдает соответствующее сообщение.

5. Модуль случайной выборки вопросов

Этот модуль производит анализ теста, для каждой темы случайно выбирает заданное число вопросов и формирует единый список вопросов для проведения тестирования. Выборка производится на основе перемешивания номеров вопросов, входящих в данную тему.

6. Модуль опроса

Модуль опроса выполняет следующие функции:

1) навигацию по списку вопросов (назад, вперед, на певый,на последний, по номеру вопроса);
2) вывод вопроса на экран с заданными параметрами отображения;
3) ввод и сохранение ответа студента.

Процесс вывода вопроса является самым сложным, т.к. он может содержать текст, таблицы, рисунки, формулы, окна ввода и области выбора. Размер шрифта и разрешающая способность экрана могут быть разными. Важным элементом модуля опроса является подсистема планирования размещения объектов в клиентском окне системы. Составной частью подсистемы планирования является программа планирования размещения формул. На вход этой программы подается строка символов с описанием формулы, записанной на языке представления формул (язык

описания формул см. в разделе 4.4) с заданным размером шрифта. На выходе выдаются размеры и параметры прямоугольной области отображения формулы. Результатом работы подсистемы планирования является список объектов отображения с указанием координат размещения в клиентском окне системы проведения контрольных работ.

Далее последовательно производится вывод списка объектов размещения в клиентское окно системы. Процесс планирования и отображения производится при любом изменении параметров отображения (листании, изменении размеров окна или шрифта).

После отображения вопроса модуль опроса переходит в режим ожидания. Студент должен ввести ответ или нажать одну из управляющих клавиш: «предыдущий вопрос», «следующий вопрос», «оценивание» и «выход».

При работе этого модуля студент может:

– изменить параметры отображения (увеличить или уменьшить размеры шрифта, изменить цвет фона и символов);

– вывести гипертекстовый файл помощи, в котором записаны способы ввода ответа и режимы работы систем;

– вести записи в специальное окно, содержимое которого по окончании тестирования записывается в протокол теста.

7. Модуль оценивания

Этот модуль предназначен для определения оценки уровня знаний по ответам студента. Происходит сравнение эталонных значений ответов с введенными ответами и выдается окно с результатом (рис. 4.7).

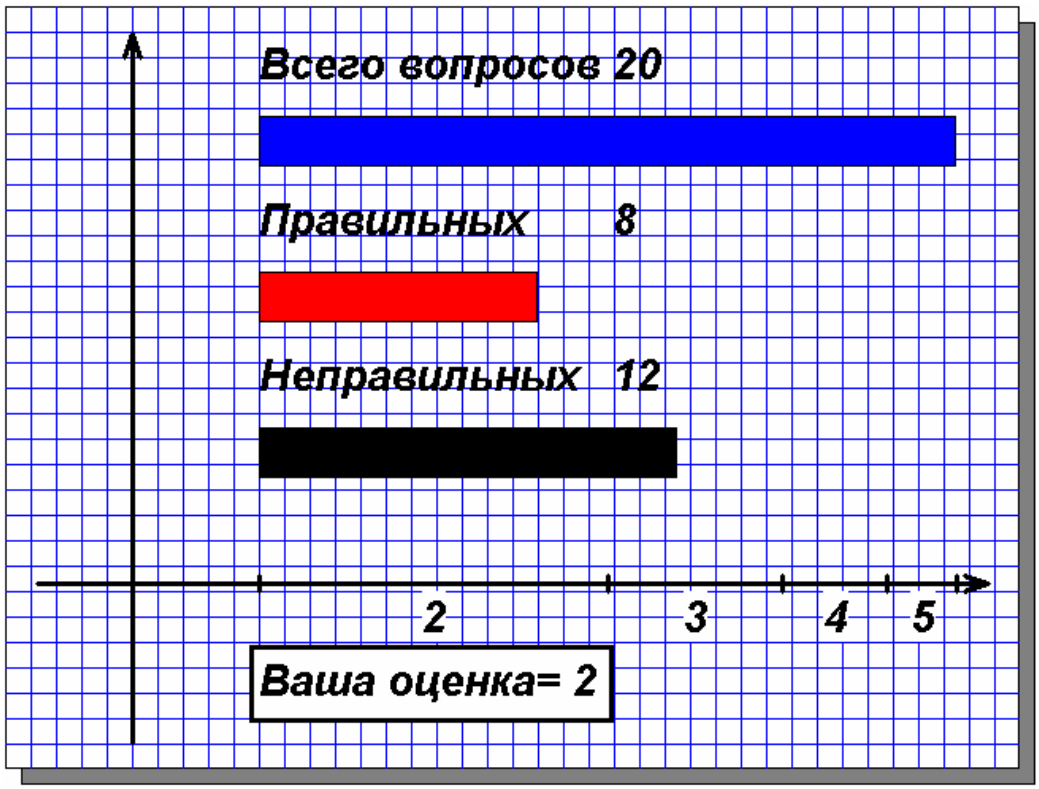


Рис. 4.7. Окно оценивания в подсистеме проведения контрольных работ

После вывода оценки программа по требованию студента производит запись протокола проведения контрольной работы в файл. При этом протокол кодируется специальным алгоритмом кодирования.

8. Модуль просмотра протокола

Студенту дается возможность просмотра протоколов после их создания. Структура протокола следующая:

1. Параметры контрольной работы: вуз, кафедра, разработчик контрольной работы, имя файла контрольной, название курса, номер контрольной работы, идентификационный номер контрольной в базе контрольных работ.
2. Параметры студента: фамилия, имя, отчество, номер группы, индетификационный номер в базе данных студента.
3. Дата и время начала и завершения контрольной работы.
4. Список номеров вопросов, ответы студента на каждый вопрос, указание об успешности сравнения.

5. Параметры оценки: всего вопросов, число успешно отвеченных, общая оценка за контрольную работу.
6. Заметки и замечания, которые студент вносил в специальное окно в процессе проведения контрольной работы.

Ниже показан пример заставки компьютерной контрольной работы (рис. 4.8).

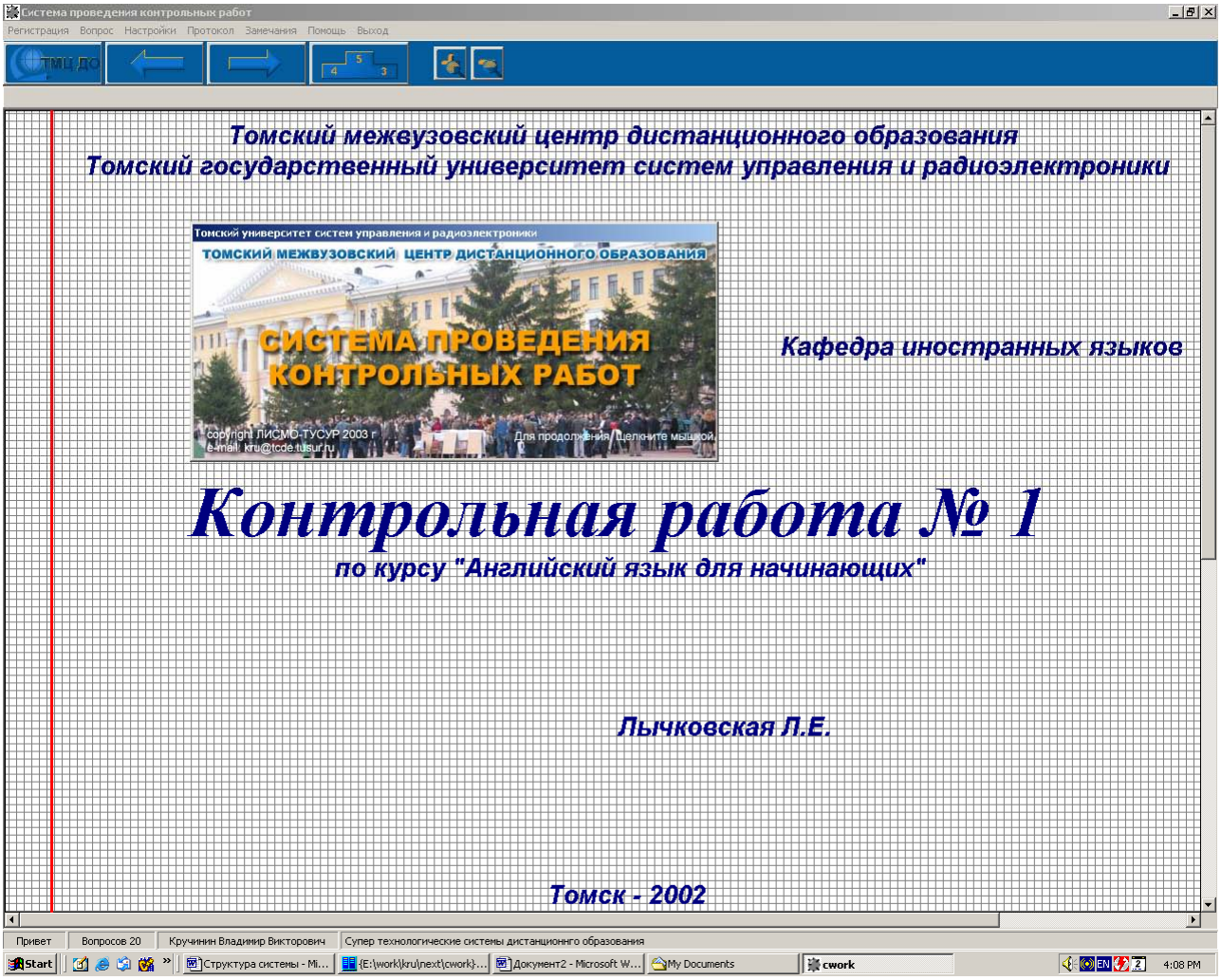


Рис. 4.8. Пример заставки контрольной работы по английскому языку

9. Особенности реализации подсистемы проведения компьютерного экзамена

Подсистема проведения компьютерных экзаменов имеет некоторые отличительные особенности:

1. Режим регистрации работает по иному: файл регистрации не создается, регистрация производится при входе в программу, помимо параметров студента необходимо ввести пароль на получение оценки.

2. По нажатию клавиши «Оценка» студенту выдается специальное окно с подтверждением пароля. Студент должен позвать преподавателя, проводящего экзамен, для ввода пароля. При правильном вводе пароля выводится отметка.

3. Система имеет многоуровневую защиту:

1) защита от многократного запуска экзамена;
2) защита от внесения изменений в код программы;
3) защита от выполнения программы в режиме отладчика.

4. Если экзамен проводится в классе, оборудованном локальной сетью, то подсистема может писать протоколы в специальную базу данных.

5. У преподавателя имеется специальная программа для просмотра протоколов в этой базе.

4.4. Описание языка представления тестов

Язык описания предметной области является важным элементом инструментальной системы. Язык позволяет отделить реализацию программы от исполнительной системы. Такой стратегии придерживаются практически все разработчики инструментальных систем. Первоначально предлагаются язык и инструментальная система, затем создается несколько конкурентных систем для данного языка, разрабатываются соответствующие стандарты. Например, для языка программирования C++ известны инструментальные системы: VisualC фирмы Microsoft [75], Cbuilder фирмы Borland [76] и другие. Следующий пример, язык интернета HTML поддерживают несколько десятков различных браузеров [76]. В настоящее время на этот язык разработаны стандарты [78].

Для инструментальных систем создания компьютерных учебных программ также имеются разнообразные стандарты [79]. Среди них наибольшее распространение получили стандарты организации IMS Global Learning Consortium [80], в кото-

рых даются спецификации на тесты и вопросы Q&TI [81, 109]. Модель QTI удобна для организации системы тестирования по следующим причинам [82]:

1. Спецификация Q&TI представляет собой не статический формат представления типизированных типов вопросов, а является расширяемым языком представления тестов, позволяющий за счет комбинации элементов языка получать нужные типы тестов и вопросов.

2. Спецификация Q&TI описывается на языке XML.

3. В спецификации Q&TI имеется стандартная таблица уровня поддержки этого формата, таким образом, возможен постепенный переход на этот формат и обмен тестами между системами обучения на указанном уровне.

4. Q&TI имеет богатый набор настроек тестирования, а также свойство расширяемости, т.е. добавления своих элементов и материалов, не поддерживаемых стандартом.

5. Иерархичность представления тестов в Q&TI позволяет разделение теста на секции по темам.

Ниже дается описание синтаксиса и семантики языка представления теста. Описание данного языка отличается от синтаксиса языка XML. Однако после описания языка будет дан набор правил преобразования теста с языка преставления тестов в формат HTML.

4.4.1. Структура теста

Для реализации компьютерных тестов и экзаменационных программ был разработан простой язык проектирования. Весь экзамен представляется в виде списка вопросов, сгруппированных по темам. Запишем грамматику данного языка:

<Тест>:= <Параметры теста> <Заставка> <Список тем>

<Параметры теста>:= <Библиотека рисунков>
<Параметры системы оценивания>
<Время теста>
<Код теста>

4.4.1.1. Заставка

<Заставка>:=#zast <ТМЦДО> <ВУЗ> <Кафедра> <Дисциплина> <Специальность> <Автор> <Место и дата> #end

Каждый компьютерный тест или экзаменационная программа имеет заголовок, который помещается между двумя ключевыми словами ***#zast*** и ***#end***:

#zast
#tmcdo *Томский межвузовский центр дистанционного образования* ***#end***
#vuz *Томский государственный университет* систем управления и радиоэлектроники ***#end***
#kaf *Кафедра компьютерных систем управления и проектирования* ***#end***
#name *Экзамен* ***#end***
#spec *по курсу Базы данных в САПР* ***#end***
#lismo *Губин И.Г.* ***#end***
#tomsk *Томск-2000* ***#end***
#end

#tmcdo #end – между ними вписывается центр-заказчик
#vuz #end – записывается вуз-заказчик
#kaf #end – указывается кафедра, для которой выполняется программа
#name #end – определяется экзамен – это тест или контрольная работа
#spec #end – записывается название предмета, по которому проводится экзамен, тест или контрольная работа
#lismo #end – указывается лаборатория, которая занималась изготовлением компьютерной учебной программы, или фамилия преподавателя, который занимался разработкой тестирующих вопросов
#tomsk #end – записывается год и место изготовления КУП

4.4.1.2. Список тем

Список тем записывается по следующим правилам:

<Список тем>:=<Тема><Список тем>

<Список тем>:=ε, где ε- символ пусто

<Тема>:=#them <Параметры темы><Список вопросов>
<Параметры темы>:=<Объем Выборки><Название темы>

Например:
#them #nrand[1]
#name *Сущность и содержание бухгалтерского учета* ***#end***
#them #end – записывается название подтемы (темы) и дополнительная информация
#nrand[i] – указывает объем случайной выборки, который будет производиться при тестировании студента (i – число вопросов случайной выборки).
Количество тем не ограничено.

4.4.1.3. Список вопросов

<Список вопросов>:=<Вопрос><Список вопросов>
<Список вопросов>:=ε
<Вопрос>:=#num <номер вопроса>#{<Формулировка>#}
<Формулировка>:=<Формулировка меню>|
<Формулировка инжекторного вопроса>|
<Формулировка числового вопроса>|
<Формулировка вопроса выбора строк>|
<Формулировка вставки строк>

4.4.2. Представление информации

Каждый вопрос должен иметь формулировку. Эта формулировка имеет две формы: простую, которая содержит только текстовое представление, и сложную, которая может содержать кроме текста еще и таблицы, рисунки, графики и формулы. Правила записи формулировки вопроса следующие:

<Текст вопроса>:=<Простая версия>|<Сложная версия>
<Простая версия>:=#quest <последовательность слов> #end
<Сложная форма>:=#TablQuest
<Текст>
<Список таблиц>
<Текст>
#end
<Текст>:=<Фрагмент><Список фрагментов>
<Список фрагментов>:=<Фрагмент><Список фрагментов>
<Фрагмент>:=<Рисунок>|
<Формула>|
<Специальные символы>|
<Словосочетание>

4.4.2.1. Рисунки

Рисунки выполняются в формате JPEG любым графическим редактором. Рисунки помещаются в библиотеку рисунков. Каждому рисунку присваивается имя. Правила записи для вставки рисунков следующие:

<Рисунок>:=#ris<name>#end

где <name>– это имя рисунка, записанного в библиотеки рисунков.

Например:

#ris shema1.jpg #end

Правила форматирования для рисунков следующие:

1. По высоте рисунок выравнивается по середине относительно основной линии шрифта.

2. Если рисунок по ширине не может быть вставлен в текущую строку, то происходит автоматический переход на следующую строку и рисунок вставляется в следующую строку.
3. Если ширина рисунка больше ширины окна, то рисунок вставляется в начало следующей строки и организуется горизонтальный скроллинг.

4.4.2.2. Формулы

Формулы вписываются в текст вопроса на основании следующего правила:

<Формула>:=#math<текст на языке описания формул>#end

А сама формула записывается на языке описания математических формул.

4.4.2.2.1. Описание языка математических формул

Язык записи математических формул был разработан ранее для учебника по математике «Математика. Часть I» по заказу Томского межвузовского центра дистанционного образования (ТМЦДО). Он имеет ряд ключевых слов для представления математических формул в текстовом виде. Рассмотрим их более подробно.

Основой для разработки явилась грамматика арифметических выражений следующего вида:

E->E+T
E->E-T
E->T
T->T*D
T->T/D
T->D
D->(E)
D->func(E)
D->name
D->const

Математическое выражение записывается на этом языке в виде строки символов, а исполнительная система преобразует эту строку в математическое изображение на экране компьютера. Например:

x+y*sin(x/b) -> $x + y \cdot \sin\frac{x}{b}$.

Рассмотрим основные элементы языка: имена переменных, специальные операции и функции.

4.4.2.2.1.1. Правила записи имен переменных

Имена переменных – это последовательность букв и цифр, начинающихся всегда с буквы. В предлагаемом языке есть множество зарезервированных имен, это набор букв греческого алфавита, некоторые специальные символы.

Таблица зарезервированных имен

Alpha	Α
Beta	Β
Gamma	Γ
Delta	Δ
Epsilon	Ε
Dzeta	Ζ
Eta	Η
Teta	ϑ
Iota	Ι

Kappa	Κ
Lamda	Δ
Mi	Μ
Ni	Ν
Ksi	Ξ
Omikron	Ο
Pi	Π
Ro	Ρ
Sigma	Σ

Tau	Τ
Ipsilon	Υ
Fi	Φ
Hi	Χ
Psi	Ψ
FI1	Θ
Omega	Ω

fi1	
alpha	α
beta	β
gamma	γ
delta	δ
epsilon	ε
dzeta	ξ
eta	η
teta	φ

iota	ι
kappa	κ
lamda	λ
mi	μ
ni	ν
ksi	ξ
omikron	ο
pi	π
ro	ρ

sigma	σ
tau	τ
ipsilon	υ
fi	ϕ
hi	χ
psi	ψ
omega	ω
beskon	∞
dobr	∂

gexist	∃
gfor	∀
nabla	∇
star	*
shtr	‘

	(штрих)
skoo	⟨
skoz	⟩
shlap	∧
@arr	→

skko	[
skkz	]
prob	пробел

4.4.2.2.1.2. Специальные операции

Ниже в таблице представлены специальные двухместные операции. Общая запись этих операций следующая: <E>@operation <E>,

где <E> – любое выражение;

@ – знак, указывающий на специальную операцию;

operation – принимает одно из значений:

tr	≅
per	⊥
vol	~
le	≤
div	/
ar2	↔
arl	←
aru	↑
arr	→
ard	↓
kru	°
pm	±
ge	≥
mul	×
alp	∝
nab	∂
krg	•
int	÷
ne	≠
to	≡
vo2	≈

xor	⊗
kad	⊕
nul	∅
min	∩
obe	∪
mgr	⊃
mge	⊇
ngr	⊄
mlt	⊂
mle	⊆
pri	∈
npr	∉
ang	∠
st2	⇔
stl	⇐
stu	⇑
str	⇒
std	⇓
tsz	;
vop	?
kav	“

В данном языке используется операция сцепления $, которая обеспечивает сцепление несвязанных выражений языка.

4.4.2.2.1.3. Функции

Функция языка	Изображение	Пример
sqrt(E)	$\sqrt{E}$	$\sqrt{x+\sin y}$
#I(E1,E1)	$E1_{E2}$	x_1
#v(E1,E2)	$E1^{E2}$	a^2
#2(E1,E2,E3)	$E1_{E2}^{E3}$	$H_k^{(S)}$
sum(E1,E2,E3)	$\sum_{E2}^{E3} E1$	$\sum_{i=1}^{n} x_i$
func(E1,E2)	E1(E2)	$\log(x+2)$
equel(E1,E2)	E1=E2	$ax+b=0$
matr(E,m,n)	$\begin{pmatrix} E_1^1 & E_2^1 \\ E_1^2 & E_2^2 \end{pmatrix}$	$\begin{pmatrix} a_1^1 & a_2^1 \\ a_1^2 & a_2^2 \end{pmatrix}$
opr(E,m,n)	$\begin{bmatrix} E_1^1 & E_2^1 \\ E_1^2 & E_2^2 \end{bmatrix}$	$\begin{bmatrix} a_1^1 & a_2^1 \\ a_1^2 & a_2^2 \end{bmatrix}$
#{(E1,E2,…,En)	$\begin{cases} E1 \\ E2 \\ E3 \\ \dots \\ En \end{cases}$	
#}(E1,E2,..En)	$\left\{ \begin{matrix} E1 \\ E2 \\ E3 \\ \dots \\ En \end{matrix} \right.$	
matE(m,n,E1,…,Ek)		
nsqrt(E1,E2)	$\sqrt[E2]{E1}$	
intgr(E1,E2,E3)	$\int_{E2}^{E3} E1$	
fuct(E)	E!	n!
color(E,c1,c2,c3)		
#cher(E)	E|	
#che2(E)	черта вертикальная	
#bm2(E1,E2,E3)	$E1 \underset{E3}{><} E2$	
#Next(E1,E2)	E1|-E2	
#Under(E)	$\underline{E}$	$\underline{x+y}$
#minpl(E)	$\mp E$	

#Not(E)	!E	
#Or(E1,E2)		
#And(E1,E2)		
#rect(E)	прямоугольник (E)	

Скобки

(E)	Выражение в круглых скобках
[E]	Выражение в квадратных скобках
r[E r]	Выражение в красных квадратных скобках
{E}	Выражение в фигурных скобках
|E|	Выражение в невидимых скобках
“E”	Выражение в прямых

Пример 1

#i(lim,x@arr omega)$func(f$prob,x)=Omega@pri #l(R,m)

$$\lim_{x \to \omega} f(x) = \Omega \in R_m$$

Пример 2

f$(m)=#{(1%prob$m=0,n*f$(n-1)%prob$m>0)

$$f(m) = \begin{cases} 1, & m = 0 \\ n \cdot f(n-1), & m > 0 \end{cases}$$

Пример 3

#l(a,1)*#v(x,n)+#l(a,2)*#v(x,n-1)+trit+#l(a,n-1)*x+#l(a,n)

$$a_1 \cdot x^n + a_2 \cdot x^{n-1} + \ldots + a_{n-1} \cdot x + a_n$$

4.4.2.3. Таблицы

Таблица – необходимый элемент представления учебной информации. Правила записи таблицы следующие:

<Таблица>:=#Tabl [m][n] <Параметры таблицы>
<Список элементов>
#endtable
<Параметры>:= <Заголовок><Границы><Подпись>

<Заголовок>:=#caption <Элемент> | ε
<Границы>:=#noborder | ε
<Подпись>:=#podpis <Элемент> | ε
<Список элементов>:=<Элемент><Список элементов>|ε
<Элемент>:=<Цвет фона><Цвет текста> <Текст элемента>
<Цвет фона>:=(R,G,B)
<Цвет текста>:=(R,G,B)
<Текст элемента>:=”Строка символов”

При описании правил записи таблицы использовались следующие обозначения:
m – количество столбцов;
n – количество строк;
(R,G,B) – целые числа в интервале [0,255], задают яркость R – красного, G – зеленого, B – голубого.

Ниже дан пример описания таблицы, у которой есть 2 столбца и 4 строки, описание параметров и цвета элементов отсутствуют. Этот пример взят из компьютерного экзамена по курсу «Исследование систем управления», разработанного доцентом кафедры «Автоматизированная обработка информации» Е.К. Рыбаловой.

```
#Tabl[2][4]
"Название понятия"
"1. Декомпозиция"
"2. <Дерево целей>"
"3. <Основание - декомпозиции>"
"Содержание понятия"
"1. Набор формальных элементов, обеспечивающих однозначное разбиение целого на части."
"2. Процедура формального разбиения целого на части (системы на составляющие ее элементы)."
"3. Метод, определяющий технологию получения строго иерархических структур путем деления целого на части."
#endtabl
```

4.4.3. Правила записи вопросов

4.4.3.1. Формулировка меню вопроса

Правила записи меню вопроса следующие:

<Формулировка меню>:=#menu <Количество вариантов>
<Текст вопроса>
<Список вариантов ответа>
#end

<Количество вариантов>:=#size <число>

Например:

#size04 – вопрос меню будет содержать четыре варианта ответа (количество вариантов всегда записывается двумя числами; если это не двузначное число, то первая цифра всегда должна быть нулем).

Дальше идет описание текста вопроса. Простой вопрос, состоящий только из текста, записывается с помощью ключевых слов ***#quest*** и ***#end.*** Например***,***

#quest *Антитоксические сыворотки применяются для лечения:* ***#end***

Для описания текста вопроса, содержащего формулы, рисунки и таблицы, необходимо использовать ключевые слова ***#TablQuest*** и ***#end.*** Правила записи показаны выше.

После описания текста вопроса записываются варианты ответов. Правила записи следующие:

<Список вариантов ответа>:=<Вариант><Список вариантов>

<Список вариантов ответа>:=<пусто>

<Вариант>:=<Правильный ответ>|<Неправильный ответ>

<Правильный ответ>:=#vars <Строка символов> #end

<Неправильный ответ>:=#varn <Строка символов> #end

Например,

Например:

```
#num 3
#{#menu#size03
#quest Что означает спецификация AS GMWT в запросе:
  SELECT P.P#, P.WEIGHT*445 AS GMWT FROM P #end
#vars a) вводит соответствующее имя результирующего столбца результирующей таблицы;  #end
```

#varn б) создает таблицу с именем GMWT в которой содержатся результаты запроса; #end
#varn в) используется для связи между отношениями. #end
#}

4.4.3.2. Инжекторный вопрос

Инжекторный тип вопроса обеспечивает ввод ответа в виде некоторой строки символов. Правила записи для первого типа следующие:

<Формулировка инжекторного вопроса>:=#Input
<Текст вопроса>
<Правильный ответ>
#end

Правила записи текста вопроса аналогичны правилам записи в меню вопросе. Рассмотрим правила записи правильного ответа:

<Правильный ответ>:=#stand <Переопределение знаков>
<Дерево И-ИЛИ>
#end
<Дерево И-ИЛИ>:= <Фигурная открывающая скобка>
<Или-список>
<Дерево И-ИЛИ>:= <Круглая открывающая скобка>
<И-список>
<Дерево И-ИЛИ>:= <Слово>
<Или-список>:= <Дерево И-ИЛИ> <Или-продолжение>
<Или-продолжение>:= <Запятая><Дерево И-ИЛИ>
<Или-продолжение>
<Или-продолжение>:= <Фигурная закрывающая скобка>
<И-список>:= <Дерево И-ИЛИ> <И-продолжение>
<И-продолжение>:= <Запятая> <Дерево И-ИЛИ>
<И-продолжение>
<И-продолжение>:= <Круглая закрывающая скобка>
<Слово>:=<Знак $>| <строка символов>
<Фигурная открывающая скобка>:= {
<Круглая открывающая скобка>:= (
<Запятая>:= ,

<Фигурная закрывающая скобка>:=}
<Круглая закрывающая скобка> :=)
<Знак $>:=$

Рассмотрим примеры записи многовариантного ответа.

Пример 1

Кто написал повесть «Дубровский»?

Запись соответствующего дерева И/ИЛИ будет следующая:

{({A.,A.C.,$},Пушкин),(Пушкин, {A.,A.C.,$})}

Тогда множество правильных ответов будет:

А. Пушкин
А.С. Пушкин
Пушкин
Пушкин А.
Пушкин А.С.

Пример 2

В ответ необходимо ввести несколько правильных последовательностей.

(1, 2, {(4,5),4,5} , {7,8}, 9)

Тогда правильные последовательности следующие:

1 2 4 5 7 9
1 2 4 5 8 9
1 2 4 7 9
1 2 5 7 9
1 2 4 8 9
1 2 5 8 9

Механизм переопределения знаков необходим на тот случай, если знаки, используемые при записи дерева И/ИЛИ, записываются в ответе. Например, в ответе требуется ввести последовательность, разделенную запятыми. Правила записи замены следующие:

<Переопределение знаков>:= key <Переопределение>
<Продолжение списка переопределений>
<Продолжение списка переопределений>:=<Переопределение>
<Продолжение списка переопределений>
<Продолжение списка переопределений>:=end
<Переопределение>:=<Знак>=<Знак замены>

<Знак>:={ | (| , |) | } | $

<Знак замены>:=любой знак не использующийся при записи ответа.

#stand key ,=+ end (5+8+11) #end

Тогда в ответе должен быть записан
5, 8, 11

Например:

```
#num 4
#{#input
#quest Назовите основные компоненты ЭЦ (ответ дайте в алфавитном порядке через пробелы). #end
#stand (({И,и},сточники),{и,$},приемники)#end
#}
```

4.4.3.3. Формулировка числового вопроса

Числовой вопрос аналогичен инжекторному, только в ответ необходимо ввести некоторое число. Это число может быть записано в формате числа с плавающей запятой. Правила записи этого вопроса следующие:

<Формулировка числового вопроса>:=#numinput
<Текст вопроса>
<Правильный ответ>
#end

Правила записи текста вопроса записаны в разделе 4.4.2.
Правила записи правильного ответа следующие:

<Правильный ответ>:=#Number<Эталонное значение>
<Погрешность ввода>
#end

<Эталонное значение>:= [число]
<Погрешность ввода>:=[число}
<Погрешность ввода>:=пусто

Правила для сравнения введенного ответа с эталонным ответом следующие:

1. Если эталонное значение равно нулю, то введенный ответ будет правильным, если тоже будет равен нулю.

2. Если эталонное значение не равно нулю и установлен интервал ошибки, то правильный ответ вычисляется по следующей формуле:

$$O = \left| \frac{A_Э - A_C}{A_Э} \right|,$$

где $A_Э$ – эталонный ответ;

A_C – ответ, введенный студентом;

O – ошибка ответа.

Если ошибка $O \le O_Э$, где $O_Э$ – погрешность ввода, то ответ считается правильным. В противном случае – неправильным.

3. Если значение ошибки ввода отсутствует в описании правильного ответа, то ее значение приравнивается 0,05 по умолчанию.

Такое представление правильного ответа позволяет управлять ситуацией с неточным и округленным ответом. Например, если правильный ответ 100, а студент ввел значение 99,9, то в зависимости от погрешности ввода этот ответ будет считаться правильным или неправильным:

```
#num 1
#{#numinput
#QuestTabl
#Text"Концентрация свободных электронов в металле равна #math
5@mul #v(10,22)$prob$#v(см,minus$3)#end. При температуре ме-
талла 23°C его удельное сопротивление равно #math
0%05$prob$мкОм*м#end. При напряженности электрического поля
2 В/м тепловая скорость электронов равна ... м/с. Массу электронов
металла считайте равной массе электрона в вакууме."#end
#Number [1.16e5] [0.05]
#}
```

4.4.3.4. Формулировка вставки строк

Вопрос с многократным вводом ответа аналогичен инжекторному. Их отличие в том, что в этом типе вопроса ответ вводится в указанном месте, а в инжекторном есть строка ввода. Ключевое слово, обозначающее этот тип вопроса, имеет вид:

<Формулировка вставки строк>:=#varinput <размер>
<varТекст>
<Список описаний ввода>

#end

<Размер>:=#size<цифра><цифра>

Параметр #size задает количество окон ввода в вопросе.

<varТекст>:=#quest<Словосочетание><Строка ввода>
<varПродолжение>

<varПродолжение>:=<Словосочетание><Строка ввода>
<varПродолжение>

<varПродолжение>:=#end

<Строка ввода>:=#<Число символов><Символы>

<Число символов>:=<цифра><цифра>

<Символы>:= строка символов указанной длинной

Например, #05xxxxx задает окошко ввода размером в 5 символов.

<Список описаний ввода>:=<Правильный ответ>
<Список описаний ввода>

<Список описаний ввода>:=пусто

Правила описания правильного ответа такие же, как в инжекторном вопросе. Пример оформления данного типа вопроса записан ниже:

```
#num 136
#{#varinput#size02
#quest Вставьте пропущенное слово.
 Оператор #10saaaaaaaaaa - конструирует новую таблицу посредством взятия горизонтального подмножества существующей таблицы, т.е. всех строк таблицы, которые удовлетворяют условию. #end
#stand ({s,S},elect)#end
#}
```

Запись *#10saaaaaaaaaa* означает, что на этом месте будет вводиться ответ, и размер строки ввода равен десяти символам.

4.4.3.5. Формулировка вопроса выбора строк

Данный тип вопроса позволяет в тексте выделять отдельные фрагменты, которые студент должен отметить как правильные. Первоначально это тип вопроса был разработан для проверки правильности расстановки ударений. Затем некоторые меню вопросы были переформулированы в данный тип. Рассмотрим правила записи:

<Формулировка вопроса выбора строк>:=#choice
<Отображение>
<choiceТекст>

#end
<Отображение>:=#show <Вид><Цвет>
<Вид>:=ColorRect| NullRect| DownLine| UpLine
<Цвет>:=(R,G,B)
<choiceТекст>:=<Слово><choiceТекст>
<choiceТекст>:=пусто
<Слово>:=<Префикс><Строка без пробела>
<Префикс>:=#<цифра><цифра><символ оценивания>
<Префикс>:=пусто

Фрагмент в тексте выделяется следующим образом:

1. Записывается префикс, в котором указывается число выделяемых символов.
2. Записывается символ, который указывает, что данный фрагмент является правильным (s) или неправильным (n). Этот фрагмент студент должен выбрать.
3. Для создания механизма выделения необходимо задать вид:
   - ***ColorRect*** – выделение цветным прямоугольником;
   - ***NullRect*** – выделение инверсией цвета;
   - ***DownLine*** – подчеркивание;
   - ***UpLine*** – надчерк.

Пример оформления вопроса множественного выбора:

#{#choice #show DownLine(0,0,256)
#quest Концепция корпоративного управления основана на том, что:
(В каждом пункте выделить верное утверждение)
а) деятельность сотрудников оценивается по (#11sрезультатам, #10nколичеству) отработанного времени;
б) сотрудники знают (#04sцели, #08nкритерии, #07nусловия, #08nинтересы) организации и стремятся к их достижению;
в) сотрудники (#05sимеют, #08nне_имеют) право формулировать и отстаивать свои собственные цели;
г) является эффективным средством управления по (#08nпроблеме, #05sцелям).#end
#}

4.4.4. Трансляция с программ тестирования в HTML формат

Переход на сетевые технологии предполагает создание системы тестирования, работающей в среде Интернет. С другой стороны, язык XML – это расширение языка HTML. Ниже запишем форматы для перевода скрипт-программ языка описания теста в формат языка HTML.

Приведенные ниже фрагменты программного кода показывают преобразование внутреннего представления скрипт-программы теста в формат HTML. Вся скрипт-программа представлена в виде последовательности элементарных объектов. Рассмотрим преобразование объектов представления информации:

1. Преобразование текста. Производится простая посимвольная вставка в выходную строку ***shtml***. Если в объекте встречается символ перевода строки, то производится вставка тега ***
*** :

```
for(int i=0; i<o.body.size(); i++){
            if(o.body[i]=='\n') shtml.append("<BR>");
           else
            shtml.append(1,o.body[i]);
}
```

2. Преобразование рисунков. В этом случае производится поиск рисунка в библиотеки и запись его в специальный каталог **images,** формируется тег ***<IMG>:***

```
shtml.append("<img src=http://");
shtml.append(host);
shtml.append("/images/");
shtml.append(n_jpg);
shtml.append("  border=1  bgcolor=wheat
align=middle>");
```

3. Преобразование формул. В настоящее время имеются разнообразные методы вставки формул в формат HTML [83]. Однако наиболее надежным является преобразование формул в

рисунки, т.к. многие браузеры не поддерживают форматы для формул. Ниже приведен код, обеспечивающий формирование тега <IMG> для преставления формул. Функция ConvertFrml2Jpeg преобразует заданную формулу в формат JPEG:

```
char scli[50];
char simg[50];
itoa(i_client,scli,10);
itoa(exm->image++,simg,10);
char PathData[256];
char n_jpg[100];
GetCurrentDirectory(255,PathData);
lstrcat(PathData,"\\images\\");
lstrcpy(n_jpg,scli); lstrcat(n_jpg,"_");
lstrcat(n_jpg,simg);
lstrcat(n_jpg,".jpg");
lstrcat(PathData,n_jpg);
ConvertFrml2Jpeg(o.body.c_str(),PathData,14,0);
shtml.append("<img src=http://");
shtml.append(host);
shtml.append("/images/");
shtml.append(n_jpg);
shtml.append(" bgcolor=wheat align=middle>");
```

4. Преобразование таблиц. Данное преобразование выполняется достаточно просто: для всей таблицы записывается тег ***<TABLE>,*** для каждой строки записывается тег ***<TR>***, для каждого элемента таблицы вызывается функция преобразования ***TranslateTablElementHtml:***

```
shtml.append("<TABLE border=1 cellspacing=0>");
 shtml.append("<CAPTION>"+
caption+"</CAPTION>");
  list<KruTableElement*>::iterator it;
  it=t.elems.begin();
  for(int i=0; i<atoi(t.m.c_str()); i++){
    shtml.append("<TR>");
    for(int j=0; j<atoi(t.n.c_str()); j++){
```

```
        TranslateTablElementHtml(*(*it));
        it++;
       }
      }
      shtml.append("</TABLE>");
```

Интерактивные элементы преобразуются в соответствующие теги ***<FORM>***. Ниже приведены фрагменты кода.

1. Преобразование окон ввода для оператора ***varinput*** :

```
 char snum[20];
 char *answer=ptr_answer;
 shtml.append("<INPUT TYPE=\"TEXT\" NAME=\"bmenu");
 itoa(i_varinput+1,snum,10);
 shtml.append(snum);
 shtml.append("\"");
 shtml.append("MAXLENGTH=100 SIZE=");
  itoa(o.body.size(),snum,10);
  shtml.append(snum);
    { //уствновть значение
      int len, k;
      memcpy((char*)&k, answer, sizeof(int));
answer+=sizeof(int);   //номер поля
       if(k==i_varinput){
          memcpy((char*)&len,answer,sizeof(int));
answer+=sizeof(int);   //длина
          if(len>0){
          shtml.append(" VALUE=\"");
          shtml.append(answer,len);   answer+=len;
          shtml.append("\"");
          ptr_answer=answer;
         }
       }
      }
      shtml.append(">");
      i_varinput++;
```

2. Преобразование оператора Input :

```
 shtml.append("<BR>Ввод ответа: <INPUT TYPE=\"TEXT\"
NAME=\"Input\"");
 shtml.append("VALUE=\"");
 char *ptr=&((char*)((void*)exm))[exm-
>offset_answer+exm->quest_num* SIZE_ANSWER];
 if(*ptr!='\0') shtml.append(ptr);
 shtml.append("\"");
 shtml.append(">\n");
 shtml.append("</P></LEFT>");
```

3. Преобразование меню:

```
int type, n, pos=0, len;
 int size;
 string val;
 char snum[20];
 char *answer;
 answer=&((char*)((void*)exm))[exm-
>offset_answer+exm->quest_num*SIZE_ANSWER];
 standard.copy((char*)&size,n=sizeof(int), pos); pos+=n;
 for(int i=0; i<size; i++){
  standard.copy((char*)&type,n=sizeof(int),pos); pos+=n;
  standard.copy((char*)&len,n=sizeof(int),pos); pos+=n;
  val=standard.substr(pos+1,len-1); pos+=len; //+1 -1
  shtml.append("<BR>");
  shtml.append("<INPUT                    TYPE=\"CHECKBOX\"
NAME=\"bmenu");
  itoa(i+1,snum,10);
  shtml.append(snum);
  shtml.append("\" ");
   if(answer[i]=='v') shtml.append(" CHECKED ");
  shtml.append("VALUE=\"Select\">");
  shtml.append(val);
 }
shtml.append("<BR>");
shtml.append("</LEFT>");
```

В настоящее время реализация транслятора с языка преставления теста в формат HTML завершается. Определяются варианты использования данного транслятора при организации учебного процесса.

4.5. Технология создания компьютерных контрольных работ и экзаменов

Технология создания компьютерных контрольных работ и экзаменов является важным элементом учебно-методической работы системы дистанционного образования. Необходимо отметить, что такие работы могут вестись децентрализованно, когда каждый методист создает сам компьютерный экзамен, используя инструментальную систему. В ТМЦДО пошли по другому пути, все работы по созданию компьютерных экзаменов были переданы в лабораторию инструментальных систем моделирования и обучения (ЛИСМО). Эта централизация позволила создать группу квалифицированных специалистов: программистов, дизайнеров и методистов, которые, используя оригинальную технологию, поставили на поток разработку компьютерных контрольных работ и экзаменов. В настоящее время в базе содержится свыше 350 компьютерных контрольных работ и экзаменов. В приложении 2 приведен полный список компьютерных экзаменов ТМЦДО. В настоящее время наметился переход к созданию генераторов компьютерных экзаменов. В разделе 5.4 приведен список генераторов контрольных работ и экзаменов, которые внедрены в практику дистанционного обучения.

Процесс создания компьютерных контрольных работ и экзаменов можно разбить на следующие этапы (рис. 4.9):

- прием задания;
- анализ множества вопросов на предмет реализации с помощью инструментальной системы проектирования КУП;
- доработка инструментальной системы;
- планирование работ;
- реализация на языке инструментальной системы;
- отладка КУП средствами инструментальной системы;
- внутреннее тестирование;
- внешнее тестирование;

- формирование программы проведения контрольной работы или экзамена на основе итоговой программы проверки;
- рецензирование;
- передача готовой программы в диспетчерский отдел ТМЦДО и подписание акта приемки-сдачи.

Рассмотрим каждый этап более подробно.

Рис. 4.9. Технология разработки

4.5.1. Прием заданий

Задания на разработку компьютерного теста поступают из учебно-методического отдела ТМЦДО. Однако в тех случаях, когда методист конкретного курса не знаком с технологией по-

строения компьютерных тестов и генераторов, проводится предварительное обучение. Это обучение включает:

1) технологию проведения компьютерных контрольных работ и экзаменов;

2) технологию построения шаблонов к генераторам компьютерных экзаменов и контрольных работ;

3) знакомство с системой проведения компьютерных экзаменов и контрольных работ.

После проведения обучения методист формирует банк вопросов или шаблонов, передает в учебно-методический отдел. Учебно-методический отдел фиксирует поступление банка вопросов и передает этот банк в ЛИСМО вместе с технологической картой (рис. 4.10).

Технологическая карта (ТК) подготовки компьютерного экзамена (КЭ)

Специальности, на которых экзамен может быть использован

Все	Только номера

Автор экзамена:

_____

Кафедра:

_____

Контактные телефоны (e-mail):

_____

Кол-во вопросов на экзамене:

_____

Диапазон оценок:

_____

График прохождения экзамена

Этап подготовки	Дата	Роспись
1. Прием задания на изготовление КЭ		
2. Окончание кодирования КЭ		
3. Уведомление преподавателя о готовности КЭ		
4. Проверка КЭ преподавателем		
5. Завершение исправлений и корректура		
6. Тестирование КЭ в лаборатории		
7. Прием КЭ в диспетчерской службе		
8. Возврат ТК в отдел УМПО		

Рис. 4.10. Технологическая карта подготовки компьютерного экзамена

В этой карте должно быть указано:

- организация заказчик;
- кафедра;
- название курса;
- автор экзамена;
- контактные телефоны и адрес электронной почты;
- количество вопросов;
- параметры системы оценивания.

По мере изготовления теста или генератора теста фиксируются соответствующие этапы выполнения работ. Это позволяет проследить за ходом выполнения работ.

4.5.2. Анализ теста на предмет реализации с помощью инструментальной системы проектирования КУП

На данном этапе тест анализируется на предмет реализации средствами инструментальной системы (ИС). Анализируются следующие элементы:

1. Формулировки вопросов, с точки зрения представления информации на экране компьютера (рисунки, формулы, таблицы).
2. Анализируются способы ввода ответа.
3. Анализируются способы представления ответов.

Если возникают проблемы с реализацией компьютерного экзамена, то:

1) предлагается переделать вопрос;
2) предлагается внести изменения в инструментальную систему.

4.5.3. Доработка при необходимости инструментальной системы

Может возникнуть потребность в изменении самой инструментальной системы: добавить новый вид вопроса и т.д. Например, в тесте по курсу «Радиоматериалы и радиокомпоненты» возникла необходимость добавить ввод ответа с учетом погрешности – появился новый тип вопроса ***numinput***. Эта информация также фиксируется.

Для проверки правильности работы ИС после внесения в нее изменений существует тестовый набор исходных текстов компьютерных экзаменов, в которых отражены все типы вопросов в большом количестве. После того как система была доработана, она тестируется на корректность работы, и только потом предыдущая версия заменяется новой.

В зависимости от сложности изменений пересматриваются сроки сдачи готовой программы.

4.5.4. Планирование работ

На данном этапе происходит распределение работ, связанных с:

1) преобразованием формул с помощью языка представления формул;

2) созданием библиотеки рисунков;

3) кодированием множества вопросов с помощью языка представления тестов;

4) тестированием ИС;

5) если тест был передан в текстовом виде, то учитывается еще и его набор и проверка в редакторе MicrosoftWord [84]. Оцениваются сроки выполнения работ, строится график работ по созданию данного теста.

4.5.5. Реализация на языке инструментальной системы

Процесс кодирования теста также можно разделить на ряд этапов:

– определение параметров теста – время и параметры оценивания;

– запись заставки;
– кодирование вопросов по темам;
– создание библиотеки рисунков.

Сами рисунки создаются любым графическим редактором, например FotoShop фирмы Adobe [85]. Формат представления рисунков должен быть Jpeg [86]. Далее необходимо использовать программу для работы с библиотекой рисунков, которая выполняет следующие функции: создает библиотеку рисунков; добавляет новый рисунок; удаляет рисунок из библиотеки; замещает рисунок на новый; выводит список рисунков.

Рис. 4.11. Схема работы инструментальной системы

На рис. 4.11 показаны основные этапы использования инструментальной системы для создания и отладки теста.

4.5.6. Отладка КУП средствами инструментальной системы

По окончании реализации теста начинается этап отладки готовой программы. Программу запускают на выполнение и

смотрят, безошибочно ли работает транслятор, все ли запускается. Если транслятор выдает ошибку, то происходит исправление этой ошибки.

4.5.7. Внешнее тестирование

После осуществления отладки и внутреннего тестирования разработанный тест передается преподавателю разработчику вопросов на проверку. Для этого используется специальная программа, которая позволяет просмотреть все вопросы и получить правильные ответы. Кроме того, преподаватель получает и программу проведения экзамена или контрольной работы в зависимости от вида теста.

Результатом внешнего тестирования могут быть:

1) замечания и неточности, обнаруженные преподавателем и требующие внесения изменений в проверяемый тест;
2) удаление или замена отдельных вопросов;
3) изменение параметров системы оценивания.

Этот процесс носит итерационный характер, но, как правило, на второй итерации процесс тестирования заканчивается.

4.5.8. Рецензирование

После изготовления в некоторых случаях тест передается на рецензирование. Рецензентом назначается специалист из данной предметной области для оценки содержания теста. Рецензент получает тот же набор программ, что и разработчик теста, и пишет рецензию. В рецензии он оценивает качество вопросов и системы оценивания. Если рецензент обнаружил неточности в формулировках вопросов, то процесс создания теста возвращается на этап отладки. Это процесс также носит итерационный характер.

4.5.9. Передача в диспетчерский отдел

После завершения исправлений замечаний рецензента формируются файлы теста и подписывается акт приемки/сдачи. Этот акт подписывается представителями учебно-методического отдела, лаборатории инструментальных систем моделирования и обучения, диспетчерского отдела и автором банка вопросов. По-

сле подписания акта файлы передаются в диспетчерский отдел для внесения в базу тестов.

В тех случаях, когда инструментальная система дорабатывается, новая версия также передается в диспетчерский отдел.

4.6. Технология разработки генераторов

Этапы создания генераторов те же самые, что в технологии создания компьютерных контрольных работ и экзаменов. Однако содержание некоторых этапов существенно отличается. Перечислим следующие отличия:

1. Принимается для реализации не банк вопросов, а банк шаблонов (заготовок). Каждый шаблон содержит описание генерируемого вопроса в соответствии с выбранной моделью (см. главу 3).

2. На этапе анализа оцениваются возможности каждого шаблона по генерации вопросов, отсеиваются те, которые генерируют небольшое число вариантов или довольно трудоемки для программной реализации.

3. Выбирается система программирования. Это может быть CBuilder, VisualC и другие, которые создают DLL библиотеки (более подробное описание дано в главе 5).

4. Создается программный образ генератора и производится тестирование и отладка генератора. Важным элементом отладки генератора является планирование тестирования, поскольку он может генерировать огромное число вопросов. Здесь предлагается поступить следующим образом: в тех случаях, когда общее число вопросов менее 100000, можно перечислить все вопросы, используя алгоритмы генерации по номеру; если общее число вопросов очень велико (см. пример 1 в главе 5, общее число вопросов $5 \cdot 10^7$), то можно использовать случайную выборку в разумных пределах.

Опыт создания генераторов показал, что наиболее важной проблемой является проблема зацикливания, поскольку при создании генератора часто программируется случайная выборка параметров в цикле. Например:

```
генерировать(a);
цикл
 генерировать(b);
```

пока(a>b)

Если границы изменения параметров **a** и **b** одинаковы, то при принятии **a** максимального значения цикл будет бесконечным. Данный пример очевиден, однако в реальной практике встречаются более сложные случаи. Выходом из данного положения являются:

1) ограничение числа итераций в цикле по счетчику или по времени и в случае зацикливания отказ от использования данного шаблона в данном процессе генерации;

2) использовать алгоритмы генерации по номеру вопроса и те номера, которые некорректны, записывать в список исключений.

5. Проверка готового генератора преподавателем также носит итерационный характер. Причем необходимо многократно запускать генератор и проверять сгенерированные вопросы.

В целом следует отметить, что создание генератора вопросов намного сложнее процесса создания теста из готового банка вопросов.

Отдельные вопросы создания генераторов, а также примеры готовых шаблонов вопросов рассмотрены в главе 5.

Глава 5. ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

5.1. Введение

Программное обеспечение в конечном итоге выполняет функции по генерации тестовых заданий. Существует довольно много различных подходов к проектированию программного обеспечения [87-89]. В нашем случае предлагается использовать проверенный на практике подход, основанный на объектно-ориентированной методологии проектирования программного обеспечения [90]. Другой важной особенностью является создание программного обеспечения без ориентации на стандартное программное обеспечение инструментальной среды. Привязка к инструментальной среде приводит к ситуации, когда невозможен перенос разработанного программного обеспечения из одной операционной системы в другую. Мобильность – чрезвычайно важная характеристика современного программного обеспечения. Опыт показывает, что для обеспечения мобильности практически все базовое программное обеспечение необходимо иметь в исходных кодах.

Для реализации программного обеспечения необходимо иметь инструментальные среды. На сегодняшний день уже имеется достаточно много разнообразных систем моделирования, проектирования и кодирования программного обеспечения [74, 75, 91, 92].

В качестве языка программирования для генераторов выбран язык программирования C++. Это объясняется следующими обстоятельствами:

1. C++ наиболее распространенный язык программирования.

2. Для C++ разработан стандарт ISO/IEC 14882 [93], придерживаясь этого стандарта создается мобильное программное обеспечение.

3. Важным элементом этого стандарта является стандартная библиотека шаблонов (STL) [94].

Все программное обеспечение генераторов можно представить в виде трех уровней:

1) базового;
2) среднего;
3) генераторов описания тестовой программы.

В базовый уровень входят датчики случайных чисел и программы генерации комбинаторных объектов. В средний уровень – классы для генерации описаний на языке представления тестов.

5.2. Датчики случайных чисел

Датчик случайных чисел – это обычно библиотечная функция (подпрограмма), при обращении к которой она выдает случайное число в некотором заданном диапазоне. Датчики случайных чисел имеют большое значение в разнообразных предметных областях: моделирование, тестирование, криптография и т.д. Датчики случайных чисел различаются:

1) по типу генерируемого числа (целое, вещественное);
2) по методам генерации;
3) по длине генерируемой случайной последовательности;
4) по скорости генерации.

Любой ДСЧ можно представить в общем виде

$$X[i+1]=R(S,X[i],P),$$

где i – номер итерации;

X[j] – значения генерируемых чисел;

R() – функция генерации;

S – начальное значение датчика;

P – параметры датчика.

ДСЧ целого типа генерируют числа в диапазоне [0,MAXINT], а ДСЧ вещественного типа – в диапазоне [0,1]. Важным свойством ДСЧ является возможность повторения случайной последовательности. Это достигается за счет того, что при одинаковых значениях исходных параметров ДСЧ выдает одну и ту же последовательность чисел.

Методы (алгоритмы генерации)

1. Линейно – конгруэнтный метод:

x[i+1]=(ax[i]+b) mod n;

2. Для генерации вещественных чисел:

z=cube(x[i]+s) //возведение в куб плавающий формат

x[i+1]=z-[z].

3. Для генерации вещественных чисел:

x[i+1]=x[i]/sin(x[i]).

Приведенные алгоритмы генерации одни из многих. Рассмотрим пример библиотечных функций для генерации чисел. В библиотеке языка программирования Си имеются следующие функции: ***rand(), randomize(), srand(int s), random(int n).*** Рассмотрим каждую в отдельности:

1) ***int rand()*** – функция генерации целого числа в диапазоне **[0, MAX_RAND],** Константа **MAX_RAND** определена как ***#define RAND_MAX 0x7FFFU;***

2) ***randomize()*** – установка начального значения для датчика случайных чисел, причем эта установка значений случайная;

3) ***srand(int s)*** – установка начального значения ДСЧ, параметр **s** явно задает начальное значение;

4) ***random(int n)*** – генерировать случайное число в диапазоне [0, n-1].

Ниже описан класс генерации случайных чисел, ориентированный на стандартный датчик:

```
class MRandNum
{
 int b0, b1; //границы
 public:
  MRandNum() { b0=b1=0; }
  MRandNum(int n) { b0=0; b1=n; }
  MRandNum(int bd0, int bd1) { b0=bd0; b1=bd1; }
    //установить случайно начальные значения датчика
  void MRandomize() { randomize(); }
    //получить случайное число без параметорв
  int MRand() { return rand(); }
    //установить начальный параметр датчика (ceed)
  void MSrand(int ceed) { srand(ceed); }
    //получить случайное число с заданными парами.
```

```
 int MRandom() { return random(b1-b0)+b0;}
   //то же самое, только используются скобки
 int operator()() { return random(b1-b0)+b0; }
   //изменить параметры датчика
 void Set(int n) { b0=0; b1=n; }
 void Set(int bd0, int bd1) { b0=bd0; b1=bd1; }
};
```

Проблемы использования стандартных датчиков случайных чисел

При переносе программного обеспечения с одной вычислительной среды в другую, например, с ОС Windows в Unix, могут возникнуть проблемы совместимости стандартных датчиков. Лучшим вариантом решения этой проблемы является создание собственного датчика случайных чисел, который можно переносить из одной вычислительной среды в другую. Для этого необходимо воспользоваться известным методом генерации случайных последовательностей [95-98].

5.3. Перемешивание

Часто необходимо иметь некоторую случайную последовательность чисел, значения которых не должны повторяться. (Как известно, в датчике случайных чисел числа могут повторяться). В тех случаях, когда множество невелико и его можно представить в виде массива, предлагается следующий алгоритм:

1. Массив заполняется числами с заданными значениями (например, от *0* до *n-1*).
2. Выбираются случайно два числа и меняются местами.
3. Шаг 2 повторяется n раз.

Приведенный выше алгоритм перемешивания на Си++ выглядит так:

```
class Mix {  //класс перемешивания
 int *mix;   //массив для хранения чисел
 int size;   //размерность
 int top;    //текущая позиция
public:
 Mix(int n):size(n){  //первоначальная установка
 mix=new int[n];
```

```
for(int i=0; i<n; i++) mix[i]=i;
  top=0;
  Mixture();
 }
void Mixture() {   //перемешать числа в массиве
  int k,l, t;
 top=0;
  for(int i=0; i<n; i++){
   k=random(n);
  - l=random(n);
   if(k!=l) {
      t=mix[k];
      mix[k]=mix[l];
      mix[l]=t;
    }
 }//кц
}//
int operator[](int i){
if(i<n) return  mix[i];
return n;
}
int Get(){  //получить текущее число
  if(top<n) return mix[top++];
 return n;
}
~Mix() {
  delete []mix;
}

...
Mix quest(10); //перемещать 10 чисел
int num[3];
for(int i=0; i<3; i++) num[i]=quest.Get();
```

В тех случаях, когда множество достаточно велико, можно предложить следующий алгоритм:

1. Множество сгенерированных чисел пусто, $N=0$.
2. Генерируем случайное число r.
3. Проверяем, имеется число r в множестве N.
4. Если есть, переходим на шаг 2.
5. Если нет, то заносим его во множество N.

Этот алгоритм еще можно использовать и для случая, когда некоторые значения генерировать нельзя. Для этого заранее в *N* заносятся те значения, которые запрещены для генерации.

5.4. Шаблоны для представления комбинаторных алгоритмов

Стандартная библиотека шаблонов является мощным инструментом разработки переносимого программного обеспечения [99]. Некоторые авторы рассматривают программирование с применением STL как генерирующее программирование (***Generic Programming***) [56] . В состав этой библиотеки входят генераторы последовательностей. Это функции, которые ориентированы на использование специального вида указателя, называемого итератором. В этом наборе генерирующих функций имеется две, которые обеспечивают генерацию перестановок. Описание этих функций следующее:

```
bool next_permutation (BidirectionalIterator first,
    BidirectionalIterator last, [ Compare ] );
bool prev_permutation (BidirectionalIterator first,
    BidirectionalIterator last, [ Compare ] );
```

Эти функции последовательно перечисляют множества, заданные двумя интераторами first и last. Функция ***next_permutation*** в прямом направлении, функция ***last_permutation*** в обратном. Основным недостатком этих функций является последовательный перебор всех перестановок. Ниже предлагается вариант развития библиотеки STL, в который вводятся шаблоны для генерации перестановок, сочетаний и размещений. Предлагаемые шаблоны основываются на алгоритмах, описанных во второй главе.

Рассмотрим первый шаблон для генерации перестановки. Класс ***T*** задает тип объектов, из которых будет состоять перестановка, причем он представляется массивом указателей. Интерфейсом класса являются следующие функции:

1. ***MPermitation(int k)*** – конструктор класса перестановок, инициирует вектора комбинаций и позиций, выделяет память под вектор указателей объектов.

2. ***void Set(int num)*** – функция определяет по номеру num перестановку.

3***. T*& operator[](int i)*** – функция переопределения квадратных скобок предназначена для установки множества объектов.

4. ***T*& operator()(int i)*** – функция переопределения круглых скобок, предназначена для чтения i-го объекта перестановки.

5. ***Next()*** – получить следующую перестановку.

6. ***Prev()*** – получить предыдущую перестановку.

7. ***Set(int num)*** – установить перестановку по номеру ***num***.

8. ***int index(int i)*** – выдает индекс ***i-го*** элемента перестановки.

9. ***int size()*** – выдает количество перестановок.

```
template <class T>
class MPermutation
{
 int *t;  //вектор комбинаций
 int *v;  //вектор позиций
 T **ts;  //множество элементов
 int n;   //размер
 int size;
 public:
  MPermutation(int k)
  int Size();
  void Next();
  void Show();
  void Insert(int e, int pos);
  void Shift(int pos);
  void Get();
  void ShowT();
  void Set(int num);
  int Num();
  T*& operator[](int i) { return ts[i]; }
  T*& operator()(int i) { return ts[t[i]]; }
  int index(int i) { return t[i]; }
  int Fact()
  ~MPermutation()
};
```

Шаблон для представления сочетаний:

```
// определение множества сочетаний из m элементов по n (n<m)
//
template <class T>
class MCombination
 {
      MINT ni;     //размер сочетания
      MINT mi;     //множество элементов
      MINT *v;     //бинарный вектор сочетаний
      T **t;       //массив указателей на элементы множества
  MINT nper;     //количество сочетаний
  MINT it;
  MINT num;    //номер сочетания
  int top;
 public:
          //конструктор m-размер множества; n-число сочетаний
  MCombination(MINT n, MINT m);
 ~MCombination();
  void Clear(); // установить начальное сочетание
  MINT Next();  // получить следующее сочетание
  MINT Size() { return nper; } //общее количество сочетаний
  //установить сочетание по заданному номеру
  void Set(MINT nu);
  MINT Num()//получить текущего номер сочетания
//оператор установки i-го элемента множества
  T*& operator[](MINT i);
//оператор чтения элемента сочетания
  T*& operator() (MINT i);
  void ClearIt(); //чтение первого элемента в сочетании
  T* NextIt();    //чтение следующего элемента в сочетании
  void Show();    //вывести сочетание
  int SizeCombination() { return ni; }
  int SizeSet() { return mi; }
  private:
   void Set(MINT, MINT,MINT);
   MINT Comb(MINT, MINT);
   long Fact(MINT);
   long NumberVar();
 };
```

вычисление множества размещений без повторений

```
template <class T>
class MArrangement
{
 long size;
 int ni; //размер размещения
```

```
int mi; //размер множества
long num;
int ll;
MCombination<T> *comb;
MPermutation<T> *perm;
public:
MArrangement(int n,int m);
long Size();
void Set(long n);
long Num();
void Next();
T*& operator[] (int i)
T*& operator() (int i) void Clear() { Set(0L); }
Show()
};
```

5.5. Классы для генерации операторов языка представления тестов

Для формирования описания вопросов на языке описания теста (см. описание языка – раздел 4.4) разработана система классов. Основой этой системы классов является класс ***GString***, который обеспечивает потоковую обработку строк символов. Члены данные класса:

ptr – указатель на строку символов, память под указатель выделяется динамически во время работы конструкторов;

len – количество символов записанных в буфере ptr;

size – размер буфера;

width – установка ширины для ввода целых чисел.

Описание класса обработки строк символов приведено ниже.

Конструкторы обеспечивают различные способы инициализации строки символов.

GString();	Выделяется строка по умолчанию
GString(char *s);	Выделяется строка размера *s* и копируется туда *s*
GString(int sz);	Выделяется строка размера *sz*
GString(GString &s);	Конструктор копирования

Потоковые операции

GString& operator<<(char *s);	Запись строки в буфер
GString& operator<<(GString &s);	Запись объекта Gstring в буфер
GString& operator<<(double k);	Запись переменной с плавающей запятой в буфер
GString& operator<<(int k);	Запись целой перемен-ной в буфер

```
class GString
{
 public:
   char *ptr;
   int len;
   int size;
   int width;
  //конструкторы
  GString();
  GString(char *s);
  GString(int sz);
  GString(GString &s);
  //операции
  GString& operator<<(char *s);
  GString& operator<<(GString &s);
  GString& operator<<(double k);
  GString& operator<<(int k);
 //функции
  void Clear();
  void Width(int k);
  void Show(char *name);
  void Show();
//деструктор
  ~GString();
};
```

Ниже перечислены классы для записи правил описания во-просов.

Класс GMath предназначен для хранения формул, записанных на языке описания математических формул:

```
class GMath: public GString
{
 public:
  GMath(char* s);
  GMath(GString *s);
  GMath(int sz);
  GMath& operator<<(char *s);
  GMath& operator<<(GString &s);
};
```

Класс Gtext предназначен для хранения текста вопроса:

```
class GText: public GString
{
 public:
 GText(char* s);
 GText(GString *s);
 GText(int sz);
 GText& operator<<(char *s);
 GText& operator<<(GString &s);
 GText& operator<<(GMath &s);
};
```

Класс GStand предназначен для оформления эталонного ответа в вопросе типа INPUT:

```
class GStand: public GString
{
 public:
  GStand(char* s);
  GStand(GString *s);
  GStand(int sz);
  GStand& operator<<(char *s);
  GStand& operator<<(GString &s);
};
```

Класс ***GquestTabl*** предназначен для организации описания оператора: ***#QuestTabl*** языка представления теста:

```
class GQuestTabl: public GString
{
 public:
  GQuestTabl(char* s);
  GQuestTabl(GString *s);
```

```
  GQuestTabl(int sz);
  GQuestTabl& operator<<(char *s);
  GQuestTabl& operator<<(GString &s);
  GQuestTabl& operator<<(GText &s);
};
```

Класс Ginput предназначен для генерации описания вопроса типа input:

```
class GInput:public GString
{
 public:
  GInput(char* s);
  GInput(GString *s);
  GInput(int sz);
  GInput& operator<<(char *s);
  GInput& operator<<(GString &s);
  GInput& operator<<(GStand &s);
  GInput& operator<<(GQuestTabl &s);
};
```

Класс GstandNumber предназначен для генерации описания правильного ответа для вопроса типа NumInput:

```
class GStandNumber: public GString
{
  public:
  GStandNumber(char* s);
  GStandNumber(GString *s);
  GStandNumber(int sz);
  GStandNumber& operator<<(char *s);
  GStandNumber& operator<<(GString &s);
};
```

Класс Ginput предназначен для генерации описания вопроса типа numinput:

```
 class GNumInput:public GString
{
 public:
  GNumInput(char* s);
  GNumInput(GString *s);
  GNumInput(int sz);
  GNumInput& operator<<(char *s);
  GNumInput& operator<<(GString &s);
  GNumInput& operator<<(GStandNumber &s);
  GNumInput& operator<<(GQuestTabl &s);
```

```
};
```

Семейство классов, предназначенных для генерации меню вопросов.

Класс для записи правильного варианта ответа:

```
class GVars: public GString
{
 public:
 GVars(char* s);
 GVars(GString *s);
 GVars(int sz);
 GVars& operator<<(char *s);
 GVars& operator<<(GString &s);
};
```

Класс для записи неправильного варианта ответа:

```
class GVarn: public GString
{
 public:
 GVarn(char* s);
 GVarn(GString *s);
 GVarn(int sz):GString(sz);
 GVarn& operator<<(char *s);
 GVarn& operator<<(GString &s);
 };
```

Класс для генерации меню вопроса:

```
class GMenu:public GString
{
 public:
 int size;
 GMenu(char* s);
 GMenu(GString *s);
 GMenu(int sz);
 GMenu& operator<<(char *s);
 GMenu& operator<<(GString &s);
 GMenu& operator<<(GStand &s);
 GMenu& operator<<(GQuestTabl &s);
 GMenu& operator<<(GVarn &s);
 GMenu& operator<<(GVars &s);
};
```

Класс для генерации темы:

```
class GThem:public GString
{
```

```
 public:
 int num;
 GString name;
 GThem(GString n);
 GThem(char* n,int sz);
 GThem(GString &n,int sz);
 GThem& operator<<(char *s);
 GThem& operator<<(GString &s);
 GThem& operator<<(GInput &s);
 GThem& operator<<(GNumInput &s);
 GThem& operator<<(GMenu &s);
};
```

Класс для генерации всего экзамена:

```
class GExam:public GString
{
 public:
  GExam(int sz);
  GExam& operator<<(char *s);
  GExam& operator<<(GString &s);
  void Append(GThem &s,int nq);
  GExam& operator<<(GThem &s);
  void Show(char *n);
  void Show();
};
```

Примеры генерации вопросов

Пример 1. Генерация меню вопроса:

```
GText text(200);
GQuestTabl quest(300);
GVars vars="Обь";
GVarn varn1="Томь";
GVarn varn2="Чулым";
GVarn varn3="Чая";
GMenu menu(1000);
text<<"Назовите основную реку Томской области?\n";
quest<<text;
menu<<quest;
menu<<varn1;
menu<<vars;
menu<<varn2;
menu<<varn3;
```

Этот фрагмент кода сгенерирует следующее описание вопроса на языке представления теста:

```
#menu #size04
#QusetTabl
#Text "Назовите основную реку Томской области? "
#end
#varn Томь #end
#vars Обь #end
#varn Чулым #end
#varn Чая #end
#end
```

Пример 2

```
GMath math(300);
GText text(500);
math<<"x/y"
text<<"Вычислите значение формулы:\n";
text<<math<<"\n";
text<<"при значениях x=20, y=10"
GQuestTabl quest(500); quest<<text;
GString srez(20); srez<<2;
GStand stand(200);
stand<<"("<<srez<<")";
GInput input(3000);
input<<quest<<stand;
```

Этот фрагмент программного кода сгенерирует описание вопроса типа Input:

```
#input
#QuestTabl
 Text "Вычислите значение формулы:
  #math x/y #end
 при значениях x=20, y=10"
#end
#stand (2) #end
#end
```

Таким образом, рассмотренная система классов позволяет генерировать описания экзаменов и контрольных работ на языке описания тестов и является важным элементом технологии создания генераторов.

5.6. Механизмы встраивания генераторов в систему проведения контрольных работ и экзаменов

Механизм встраивания генераторов в систему проведения экзаменов и контрольных работ основан на использовании динамически загружаемых библиотек (DLL) (Dynamic Load Library) [100]. Генератор реализуется как DLL, имя библиотеки является уникальным. Интерфейсными функциями генератора являются:

1. extern "C" char* __declspec(dllexport) _stdcall GenMathExam();

Эта функция производит генерацию экзамена или контрольной работы в формате системы проведения компьютерных экзаменов и контрольных работ. Функция создает буфер и записывает туда тест, адрес буфера передает вызывающей функции.

2. extern "C" char* __declspec(dllexport) _stdcall DeleteExam();

Эта функция освобождает все ресурсы выделенные для работы генератора.

3. extern "C" char* __declspec(dllexport) _stdcall GetRundom();

Эта функция выдает параметр датчика случайных чисел, который впоследствии записывается в протокол тестирования.

Ниже описана функция ***ReadGen*** системы проведения компьютерных экзаменов и контрольных работ, которая выполняет следующие действия:

1) загрузку DLL библиотеки по имени, записанном в параметре ***name***;
2) определение адресов функций интерфейса генерации;
3) генерацию теста, трансляцию во внутренний код, определение параметра датчика случайных чисел генератора.

```
void ReadGen(char *name, long &rand){
  HINSTANCE hinstLib;
  MYPROC ProcAdd;

  MYPROC ProcDel;
  MYPROC ProcRand;

  BOOL fFreeResult, fRunTimeLinkSuccess = FALSE;
  //загрузить библиотеку
  hinstLib = LoadLibrary(name);
  if (hinstLib != NULL)
```

```
    {
      //определить адреса функций
      ProcAdd = (MYPROC) GetProcAddress(hinstLib, "GenMathExam");
      ProcRand = (MYPROC) GetProcAddress(hinstLib, "GetRandom");
      ProcDel = (MYPROC) GetProcAddress(hinstLib, "DeleteExam");
      if (fRunTimeLinkSuccess = (ProcAdd != NULL))
      {
        //генерировать и транслировать
        TransQ TransObj((ProcAdd)(),Test);
        //получить параметр датчика случайных чисел
        rand=(GetRandom)();
        //освободить ресурсы генератора
       (ProcDel)();
      }
      fFreeResult = FreeLibrary(hinstLib);
    }
    if (!fRunTimeLinkSuccess)
    {
      MessageBox(NULL,"Неверное имя библиотеки",MB_OK);
      TerminateProcess(pi.hProcess,0);
      exit(1);
    }
}
```

Таким образом, для встраивания генератора в систему проведения экзаменов и контрольных работ необходимо:

1) создать DLL с уникальным именем, которая экспортирует три функции: **GenMathExam, DeleteExam, GetRundom;**

2) полученную DLL записать в базу компьютерных контрольных работ и экзаменов.

5.7. Встраивание объектов в языки интерпретирующего типа

Развитие идей объектно-ориентированного программирования, применение их в практическую плоскость привели к появлению новых технологий создания и использования программного обеспечения. Одним из выдающихся семейств таких технологий являются технологии, разработанные фирмой Microsoft, OLE, ActiveX, COM, DCOM, NET [102-105].

Важным элементом этих технологий является механизм автоматизации (Automation) [106]. Этот механизм позволяет встраивать объекты, разработанные на разных инструментальных платформах, в языки интерпретирующего типа, например в VisulBasic [107]. Однако реализация данного механизма скрыта.

Ниже описан один из возможных вариантов реализации механизма встраивания объектов в языки интерпретирующего типа. Эта реализация основана на использовании шаблонных классов STL: string, vector, map.

```
#ifndef hKRUMETODTABL
#define hKRUMETODTABL
#include <vector>
#include <stdarg.h>
#include <string>
#include <map>
using namespace std;
```

Первоначально описывается пустой класс

Все классы должны наследовать этот класс

```
class KruMethodPointer{};
```

Записывается указатель на метод класса **KruMethodPointer**

```
typedef void (KruMethodPointer::*KruMPtr)(...);
typedef  int (*KruFunc)(...);
```

Описание типа указателя: метод, установить свойство, прочитать свойство, обработчик события.

```
#define KRUMETHODTYPE  1
#define KRUPROPERTYGETTYPE 2
#define KRUPROPERTYSETTYPE  3
#define KRUEVENHANDLERTYPE 4
```

Элемент таблицы, описывающий метод класса:

```
struct KruMethod {
  int type;            //тип метода
  string format;       //описание строки формата параметоров
  KruFunc method;      //адрес метода
  //конструктор
  KruMethod(int t,char *f,KruFunc pf) { type=t; format=f; method=pf;
}
 };

//для классов
struct KruClassMethod {
```

```
 int type;            //тип функции
 string  format;      //описание формата
 KruMPtr method;      //адрес функции
 KruClassMethod(int t,char *f,KruMPtr pf) { type=t; format=f;
method=pf; }
 };
```

Описание механизма передачи параметров

Передача параметров производится через вектор, представленный в виде шаблонного класса STL vector, причем каждый параметр описывается классом KruVariant, который обеспечивает многовариантное представление. Параметры могут быть следующих типов: целый, адрес, float, double. Описание типов следующее:

```
#define PINT 1
#define PADDR  2
#define PFLOAT  3
#define PDOUBLE 4
```

Ниже дано описание класса KruVariant,
где type – целая переменная задает тип параметра;
union Param – объединение разных типов, задает значение параметра.

```
const class __declspec(dllexport) KruVariant{
 int type;
 union {
   int pi;
   void* px;
   float pf;
   double pd;
  } Param;
 public:
//конструкторы
 KruVariant () { type=0; }
 KruVariant(int a)   {  type=PINT;   Param.pi=a; }
 KruVariant(float a) {  type=PFLOAT;  Param.pf=a; }
 KruVariant(double a){  type=PDOUBLE; Param.pd=a; }
 KruVariant(void* a) {  type=PADDR;   Param.px=a; }
 KruVariant(KruVariant &p) {  type=p.type;   Param=p.Param; }
//операции преобразования
  operator int(){    //преобразование int
  return (type==PINT)? Param.pi: 0;
 }
```

```
 operator double(){ //преобразование double
  return (type==PDOUBLE)? Param.pd: 0.;
 }
 operator float(){
   return (type==PFLOAT)?Param.pf: 0.;
 }
 operator void*(){ //преобразование к адресу
   return (type==PADDR)? Param.px:NULL;
 }
Операции присвоения
int operator=(int a){ //присвоение целого
 type=PINT;
 return Param.pi=a;
 }
int operator=(float a){ //присвоение float
 type=PFLOAT;
 return Param.pf=a;
 }
int operator=(double a){ //присвоение double
 type=PDOUBLE;
 return Param.pd=a;
 }
void*  operator=(void* a){ //присвоение адреса
 type=PADDR;
 return Param.px=a;
 }
};
//задаем новый тип KruParams – список параметров
typedef vector<KruVariant> KruParams;
```

Теперь рассмотрим описание класса для представления таблицы методов. Предварительно записываются типы для представления таблицы. Таблица основана на использовании шаблонных классов STL: ***map*** и ***string***. Каждый элемент содержит имя метода и описание метода (описание ***KruClassMethod*** смотри выше). Ниже описан тип таблицы ***vtable_type***. Основные переменные класса: ***Vtable*** – таблица, описывающая методы объекта; ***p*** – итератор, предназначен для организации выполнения операций над таблицей ***Vtable***.

Основные функции:

Insert	Вставить описание метода в таблицу
Replace	Заменить табличный метод на новый
Find	Найти метод в таблице по имени

Delete	Удалить метод из таблицы
Call	Вызвать метод

```
class __declspec(dllexport) KruClassVTable{
  typedef map<string, KruClassMethod, less<string> > vtable_type;
  typedef vtable_type::value_type value_type;
  vtable_type VTable;                  //таблица методов
  vtable_type::iterator p;             //итератор

  public:
int Insert(char *name, int typ, char *format, KruMPtr pfunc);     int
Replace(char *name,int typ, char *format, KruMPtr pfunc);   int
Find(char *name);       int Delete(char *name);
int Call(char *name,KruMethodPointer *This,KruParams *p);
 };
```

Реализация методов дана в приложении 1.

Рассмотрим пример построения класса с использованием вышеописанного механизма. Создаем описание DLL:

```
int WINAPI DllEntryPoint(HINSTANCE hinst, unsigned long reason,
void* lpReserved)
{
     return 1;
}
```

Описываем класс ***TestClass***, который наследует ***KruMethodPointer***. Этот класс имеет две переменные, конструктор и три метода.

```
class TestClass: public KruMethodPointer {
 int x;
 int y;

 public:
  TestClass() { x=100; y=2000; }
  int Get(KruParams *s);
  int Put(KruParams *s);
  int Sum(KruParams *s);
};
```

Описание методов:

```
1) int TestClass::Get(KruParams *s) { (*s)[0]=y; }
2) int TestClass::Put(KruParams *s) { y=(*s)[0]; }
3) int TestClass::Sum(KruParams *s) { (*s)[0]=x+y; }
```

Записываем следующие экспортируемые функции:

1. Выдать имя класса – необязательная функция, которая выдает имя класса

```
extern "C" const char* __declspec(dllexport) _stdcall
GetClassName(){
 static char *NameClass ={“Тест”};
 return NameClass;
}
```

2. Создать класс. Функция создания объекта класса

```
extern "C" KruMethodPointer* __declspec(dllexport) _stdcall
GetClass(){
 return new TestClass;
}
```

3. Удалить класс. Функция удаления класса

```
extern "C" void __declspec(dllexport) _stdcall
DeleteClass(KruMethodPointer* t){
  delete t;
}
```

4. Выдать таблицу методов

```
extern "C" KruClassVTable* __declspec(dllexport) _stdcall
GetVTable(){
  KruClassVTable *tabl=new KruClassVTable;
  tabl-
>Insert("Взять",KRUMETHODTYPE,"x",(KruMPtr)&TestClass::Get);
  tabl->Insert("Запомнить",KRUMETHODTYPE,"x",
           (KruMPtr)&TestClass::Put);
  tabl->Insert("Увеличить",KRUMETHODTYPE,"x",
          (KruMPtr)&TestClass::Sum);
  return tabl;
}
```

5. Удалить таблицу методов

```
extern "C" void __declspec(dllexport) _stdcall
   DeleteVTable(KruClassVTable *t){
   delete t;
}
```

Тогда можно предложить следующий фрагмент программного кода, в котором создается объект класса ***TestClass*** и вызываются методы этого класса через соответствующую таблицу методов:

```
//создать список параметров
 KruParams x(2);
```

```
//получить указатель на объект класса
 KruMethodPointer *Class=GetClass();
//получить указатель на  таблицу методов
 KruClassVTable *tabl=GetVTable();
//Вызвать методы по таблице
 for(int i=0; i<20; i++){
  tabl->Call("Взять", Class,&x);
  cout<<"++ "<<int(x[0])<<endl;
  x[0]=i;
  tabl->Call("Запомнить", Class,&x);
  tabl->Call("Увеличить", Class,&x);
  cout<<int(x[0])<<endl;

 }
//Удалить таблицу
 DeleteVTable(tabl);
//Удалить объект класса
DeleteClass(KruMethodPointer* t);
```

Таким образом, можно импортировать классы в интерпретирующие системы, упакованные в DLL.
Каждая DLL должна иметь следующие интерфейсные функции:
1) ***GetClass()*** – создает объект класса;
2) ***GetClassName()*** – выдает имя класса;
3) ***GetVTable()*** – получить таблицу методов;
4) ***DeleteClass()*** – удалить объект класса;
5) ***DeleteVTable()*** – удалить таблицу методов.

Система, которая импортирует DLL с описанием класса, должна выполнить следующее:
1. Загрузить соответствующую DLL.
2. Запросить имя класса, занести в таблицу, записать в таблицу классов (***ClassTable***).
3. Запросить таблицу методов, занести имена методов в таблицу ключевых слов.
4. При объявлении объекта класса: ***Class Object***; имя объекта заносится в таблицу имен объектов, вызывается функция ***GetClass()***.
5. При вызове метода
   Object.Method(x); или
   Object.Propery=x;

По имени объекта из таблицы объектов (***ObjectTable***) определяется адрес объекта и индекс класса. По индексу в таблице классов определяется адрес таблицы методов. Затем по имени метода или свойства определяется функция в таблице методов. На основании анализа строки формата параметров функции формируется список параметров ***Params***. Затем вызывается метод:

VTable->Call(имя_метода, адрес_объекта,параметры);

Таблица классов (***ClassTable***):

Имя класса	Таблица методов	Функция создания объекта	Функция удаления таблицы	Функция удаления объекта
…				
Генератор1	**KruClassVTable***	**CraeteObj***	**DelVtable***	**DelObj***
…				

Таблица методов класса (***VTable***):

Имя	Тип	Формат	Указатель
Параметр1	**PropertySet**	**in x**	**KruMPtr(addr1)**
Параметр2	**PropertySet**	**in x**	**KruMPtr(addr2)**
Параметр3	**PropertyGet**	**in x**	**KruMPtr(addr3)**
Выполнить1	**Method**	**in x, in x**	**KruMPtr(addr4)**
Выполнить2	**Metodd**	**in x**	**KruMptr(addr5)**

Таблица объектов (***ObjectTable***):

Имя объекта	Адрес объекта	Адрес класса
…		
Шаблон1	**KruMethodPointer***	**index**
…		

5.8. Программное обеспечение генераторов ТМЦДО

В настоящее время по описанной выше технологии разработано 20 генераторов, всего 926 шаблонов, объем программного обеспечения 44627 строк. Ниже приведены два списка:

– Список генераторов для проведения экзаменов:

1. Магазинников Л.И. Высшая математика. Дифференциальное исчисление (210 шаблонов, 210 инжекторного типа, объем программного обеспечения в строках 13740).
2. Магазинников Л.И. Высшая математика. Введение в анализ (151 шаблон, 40 меню, 111 инжекторного типа, объем программного обеспечения в строках 9098).
3. Обрусник В.П. Магнитные элементы (20 шаблонов, 12 меню, 8 инжекторного типа, объем программного обеспечения в строках 1154).
4. Зинченко Е.В. Психология (31 шаблон, 30 меню, 1 инжекторный, объем программного обеспечения в строках 1232).
5. Шарапов А.В. Микропроцессорные устройства (21 шаблон, 6 меню, 15 инжекторного типа, объем программного обеспечения в строках 1104).
6. Шарапов А.В. Основы микропроцессорной техники (48 шаблонов, 6 меню, 42 инжекторного типа, объем программного обеспечения в строках 1422).
7. Шарапов А.В. Цифровая схемотехника (40 шаблонов, 9 меню, 31 инжекторного типа, объем программного обеспечения в строках 1358).
8. Шарапов А.В. Электронные цепи и МТС-1 (83 шаблона, 15 меню, 68 инжекторного типа, объем программного обеспечения в строках 846).
9. Тимченко С.В. Информатика. Часть 3 (10 шаблонов, 7 меню, 3 инжекторного типа, объем программного обеспечения в строках 1226).

– Список генераторов для проведения контрольных работ:

1. Шарапов А.В. Микропроцессорные устройства (42 шаблона, 1 меню, 41 инжекторного типа, объем программного обеспечения в строках 1096).
2. Шарапов А.В. Основы микропроцессорной техники (49 шаблонов, 6 меню, 45 инжекторного типа, объем программного обеспечения в строках 1485).
3. Шарапов А.В. Микроэлектроника. Цифровая схемотехника (40 шаблонов, 9 меню, 31 инжекторного типа, объем программного обеспечения в строках 1396).

4. Шарапов А.В. Аналоговая схемотехника. Часть 1 (57 шаблонов, 9 меню, 48 инжекторного типа, объем программного обеспечения в строках 1445).
5. Шарапов А.В. Аналоговая схемотехника. Часть 2 (25 шаблонов, 4 меню, 21 инжекторного типа, объем программного обеспечения в строках 845).
6. Лычковская Л.Е. Английский язык. Часть 1 (26 шаблонов, 6 меню, 20 инжекторного типа, объем программного обеспечения в строках 2080).
7. Лычковская Л.Е. Английский язык. Часть 2 (23 шаблона, 16 меню, 7 инжекторного типа, объем программного обеспечения в строках 1854).
8. Лычковская Л.Е. Английский язык. Часть 3 (15 шаблонов, 15 меню, объем программного обеспечения в строках 1058).
9. Лычковская Л.Е. Английский язык. Часть 4 (3 шаблона, 3 инжекторного типа, объем программного обеспечения в строках).
10. Лычковская Л.Е. Английский язык. Часть 5 (20 шаблонов, 20 меню, объем программного обеспечения в строках 1591).
11. Тимченко С.В. Информатика. Часть 3 (10 шаблонов, 7 меню, 3 инжекторного типа, объем программного обеспечения в строках 627).

Каждый генератор из приведенных списков реализован как DLL и внедрен в практику дистанционного обучения ТМЦДО. Первый опыт эксплуатации показал правильность выбранного направления. Практически каждый студент получает индивидуальный набор тестовых заданий и вопросов, независимо от того, сколько раз он запускает компьютерную контрольную работу или экзамен. При этом не имеет смысла взламывать систему проведения контрольных работ и экзаменов, поскольку нет заранее заготовленных ответов.

ПРИМЕРЫ ШАБЛОНОВ ГЕНЕРАТОРОВ

Пример 1

1. Формулировка задания

Решить систему

$$X - Y = \begin{bmatrix} a_{1,1} & a_{1,2} & . & . & a_{1,m} \\ a_{2,1} & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ a_{1,n} & . & . & . & a_{m,n} \end{bmatrix}$$

$$X + Y = \begin{bmatrix} b_{1,1} & b_{1,2} & . & . & b_{1,m} \\ b_{2,1} & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ b_{1,n} & . & . & . & b_{m,n} \end{bmatrix}$$

В ответ введите сумму i-й строки матрицы X и сумму j-й строки матрицы Y.

2. Метод генерации

1. Генерируются размеры матриц *m* и *n* правых частей системы уравнений с заданным диапазоном изменения размерностей.
2. Генерируются матрицы X и Y с заданным диапазоном изменения значений элементов матриц.
3. Вычисляются матрицы правых частей уравнений.
4. Случайно генерируются значения *i* и *j* .
5. Находятся суммы $s = \sum_{k=1}^{m} x_{i,k}$ и $p = \sum_{k=1}^{m} y_{j,k}$

3. Программа генерации с использованием системы классов

Для создания данного шаблона используется класс ***MMatrix*** для работы с матрицами:

```
void Question1() {
```

```
  MRandNum r(2,4);
  int n=r.MRandom();                    //Генерировать размер-
                                        ности матриц
  int m=(n==2)?3:2;
  MMatrix X(m,n),Y(m,n);                //Генерировать матри-
                                        цы X и Y
  X.Gen();
  Y.Gen();

  GString matr1(200);                   //Сформировать описа-
                                        ния матриц X и Y
  X.ShowGen(matr1);
  GString matr2(200);
  Y.ShowGen(matr2);
  Mmatrix A(m,n),B(m,n);
 A=X-Y;                                 //Вычислить матрицы A
                                        и B
 B=X+Y;
 GString matr3(200);                    //Сформировать описа-
                                        ния матриц A и B
 A.ShowGen(matr3);
 GString matr4(200);
 B.ShowGen(matr4);
 GMath mathu1(300);                     //Сформировать описа-
                                        ние системы
mathu1<<"X"<<"-"<<"Y"<<"="<<matr3;      //уравнений
 GMath mathu2(300);
mathu2<<"X"<<"+"<<"Y"<<"="<<matr4;
 Gmath mathu3(300);
 mathu3<<matr1;
 Gmath mathu4(300);
 mathu4<<matr2;
 int I;                                 //Сгенерировать индек-
                                        сы строк
 int K;
 MRandNum ri(m);
 I=ri.MRandom();
 K=ri.MRandom();
 int res1=0;                            //Вычислить суммы
                                        строк
 int res2=0;
 for(int j=0; j<n; j++)
res1+=X.matrix[I][j];
 for(int j=0; j<n; j++)
res2+=Y.matrix[K][j];
 GString si1(10); si1<<(I+1);           //Сформировать текст
```

```
                                                    вопроса
 GString si2(10); si2<<(K+1);
 GText text(1000);
 text<<"Решить систему\n";
 text<<mathu1<<"\n";
 text<<mathu2<<"\n";
 text<<"В ответ введите сумму элементов
матриц"
   <<si1<<"-ой строки ы X\n";
 text<<"и сумму элементов "
<<si2<<"-й строки матрицы Y,\n";
  text<<"разделив их точкой с запятой";
  GQuestTabl quest(4000); quest<<text;
  GString gx(50);                          //Сформатировать пра-
gx<<(res1)<<";"<<(res2);                   вильный ответ
  GStand stand(50);
stand<<"("<<gx<<")";
  GInput input(5000);                      //Создать вопрос типа
                                           Input
  input<<quest<<stand;                     //Занести текст вопроса
                                           и правильный ответ
  them<<input;                             //Занести вопрос в тему
}
```

4. Сгенерированный текст

```
#{#input #QuestTabl#Text "Решить систему
 #math
X-Y=[matE(3,2,minus$2,prob$2,prob$2,prob$0,prob$2, prob$1)]
 #end
 #math
X+Y=[matE(3,2,prob$8,prob$0,prob$4,minus$2,prob$8,minus$5)]
#end
В ответ введите сумму элементов 3-й строки матрицы X
и сумму элементов 1-й строки матрицы Y,
разделив их точкой с запятой."
#end
#stand (3;4)#end
#}
```

5. Экранная форма сгенерированного вопроса представлена на рис. 6.1.

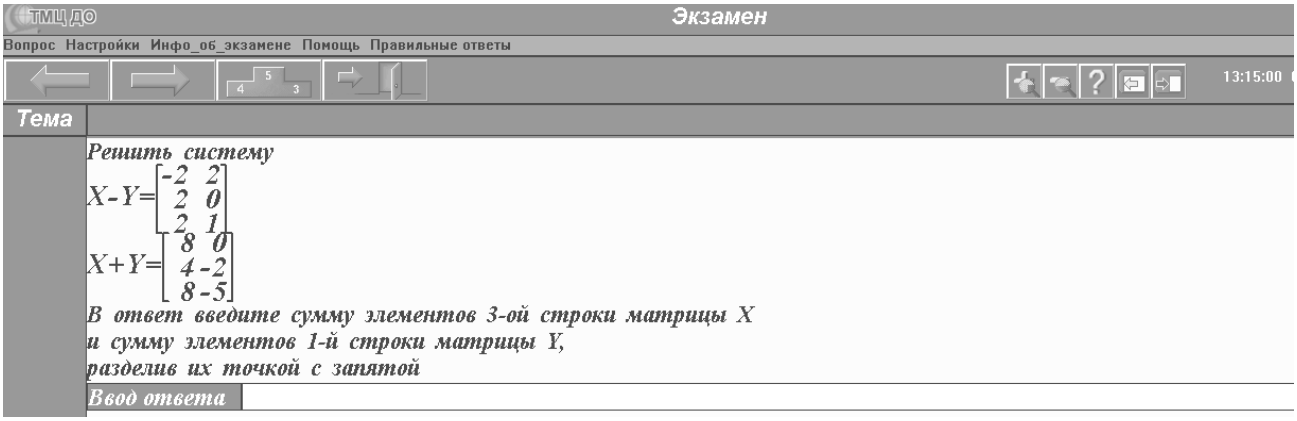


Рис. 6.1. Экранная форма вопроса

6. Подсчет числа вопросов

Число вопросов зависит от:

1) количества разнообразных матриц;
2) числа вариантов сочетаний строк.

Количество матриц можно вычислить исходя из множества значений, которые может иметь элемент матрицы:

1) предположим, что число значений равно n;
2) матрица может иметь k-размерностей;
3) генерируется всего 2 матрицы.

Тогда число вариантов матриц будет

$$V = 2 \cdot \sum_{i=1}^{k} n^{R_i} ,$$

где $R_i = m_i \cdot n_i$ – размерность матрицы.

При $n=10$, $k=2$ [2,3]=6 и [3,2]=6 формула будет

$$V = 4 \cdot 10^6 .$$

Теперь подсчитаем количество вариантов строк для случая [2,3] и [3,2]. Тогда число вариантов будет

$$2*2+3*3=13.$$

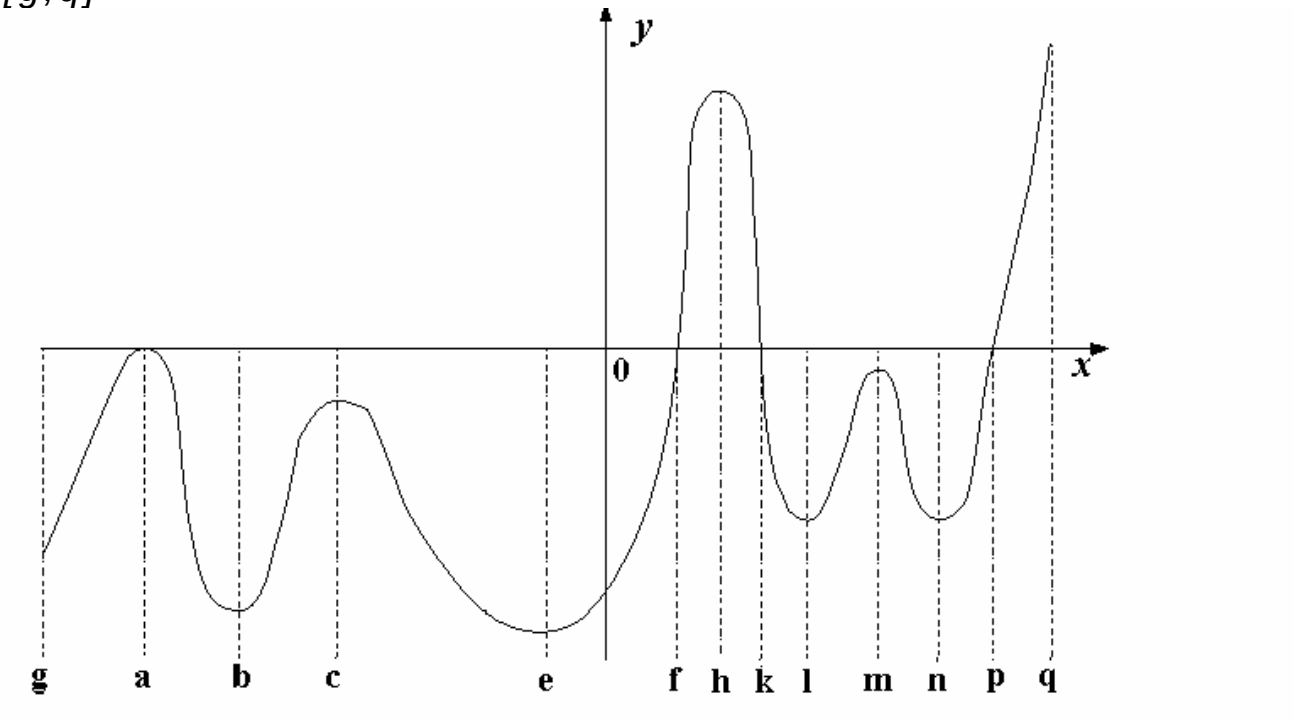
Тогда общее число вариантов вопросов будет

$$13 \cdot 4 \cdot 10^6 = 52 \cdot 10^6 \approx 5 \cdot 10^7 .$$

Пример 2

1. Формулировка задания

Дан график производной y′(x) функции f(x), определенной на [g;q]



Укажите интервалы возрастания функции в порядке следования на оси OX.
Форма записи ответа, например, (-10,-4);(-1,6).
Условие генерации: g<a<b<c<e<f<h<k<l<m<n<p<q.

2. Метод генерации

1. Генерируются параметры: *g,a,b,c,e,f,h,k,l,m,n,p,q.*
2. Проверяется условие*: g<a<b<c<e<f<h<k<l<m<n<p<q.*
3. Если условие не выполняется, то переход на шаг 1.
4. Выбирается изображение, зависящее от параметров.
5. Формируется текст описания вопроса.
6. Формируется правильный ответ.

3. Программный код

```
Question30_9(){
 double  a,b,c,d,e,g,k,l,m,n,q,p,s;
 do{   //цикл для генерации параметров : g,a,b,c,e,f,h,k,l,m,n,p,q.
   MRandNum rp(-50,-10);
```

```
  p=rp.MRandom();
  MRandNum ra(-20,-5);
  a=ra.MRandom();
  MRandNum rb(-15,-5);
  b=rb.MRandom();
  MRandNum rc(-10,0);
  c=rc.MRandom();
  MRandNum rd(-10,3);
  d=rd.MRandom();
  MRandNum re(-5,10);
  e=re.MRandom();
  MRandNum rg(-5,10);
  g=rg.MRandom();
  MRandNum rk(5,30);
  k=rk.MRandom();
 MRandNum rl(5,30);
 l=rl.MRandom();
 MRandNum rm(10,40);
 m=rm.MRandom();
 MRandNum rn(10,40);
 n=rn.MRandom();
 MRandNum rs(10,40);
 s=rs.MRandom();
 MRandNum rq(10,50);
 q=rq.MRandom();
 } while  //проверка условия
(!(p<a&&a<b&&b<c&&c<d&&d<e&&e<g&&g<k&&k<l&&l<m&&m<n&
&n<s&&s<q));
//формирование вопроса
 GString
sa(20),sb(20),sc(20),sd(20),sp(20),sq(20),sm(20),sn(20),sl(20),sk(2
0),sg(20),se(20),ss(20); sa<<a; sb<<b; sd<<d; sc<<c; sq<<q;
sp<<p; sk<<k; sl<<l; sg<<g; se<<e; sm<<m; sn<<n; ss<<s;
 GText text(1000);
 text<<" Дан график производной #math f$shtr$(x)#end функции
#math f$(x)#end, определенной на  ["<<sp<<";"<<sq<<"]:\n";
//включение изображения в зависимости от значений параметров
if (b<0&&c==0){text<<"#ris30_92.bmp#endris \n ";}
if (c<0&&d>0){text<<"#ris30_91.bmp#endris \n ";}
if (c<0&&d==0){text<<"#ris30_93.bmp#endris \n ";}
if (d<0&&e>0){text<<"#ris30_94.bmp#endris \n ";}
if (d<0&&e==0){text<<"#ris30_95.bmp#endris \n ";}
if (e<0&&g>0){text<<"#ris30_96.bmp#endris \n ";}
if (e<0&&g==0){text<<"#ris30_97.bmp#endris \n ";}
if (g<0&&k>0){text<<"#ris30_98.bmp#endris \n ";}
```

```
 text<<"g="<<sp<<", a="<<sa<<", b="<<sb<<", c="<<sc<<",
e="<<sd<<", f="<<se<<", h="<<sg<<", k="<<sk<<",
l="<<sl<<", m="<<sm<<", n="<<sn<<", p="<<ss<<",
q="<<sq<<".\n ";
text<<" Укажите интервалы возрастания функции в порядке следо-
вания на оси OX. \n";
text<<"Форма записи ответа, например, (-10,-4);(-1,6)\n";
GQuestTabl quest(4000); quest<<text;
GString gx1(50),gx2(50),gx3(50),gx4(50); gx1<<p; gx2<<e; gx3<<k;
gx4<<s;
 GStand stand(500); stand<<"key (=[ )=] ,=+ end
[("<<gx1<<"+,+"<<gx2<<");("<<gx3<<"+,+"<<gx4<<")]";
 GInput input(5000);
 input<<quest<<stand;
 them<<input;
}
```

4. Скрипт-файл для системы проведения экзаменов:

```
#input
  #QuestTabl#Text
   "Дан график производной
    #math f$shtr$(x)#end функции #math f$(x)#end,
определенной на [-18;40]:
#ris30_93.bmp#endris
 g=-18, a=-16, b=-12, c=-7, e=0, f=1, h=5, k=7, l=11, m=16,
n=21, p=22, q=40.
 Укажите интервалы возрастания функции в порядке следования их
на оси OX.
Форма записи ответа (-10,-4);(-1,6)"
#end
#stand key (=[ )=] ,=+ end [(-18+,+1);(7+,+22)]#end
```

6. Сгенерированный конкретный вопрос

На рис. 6.2 показана экранная форма конкретного вопроса, полученного на основании использования рассматриваемого генератора.

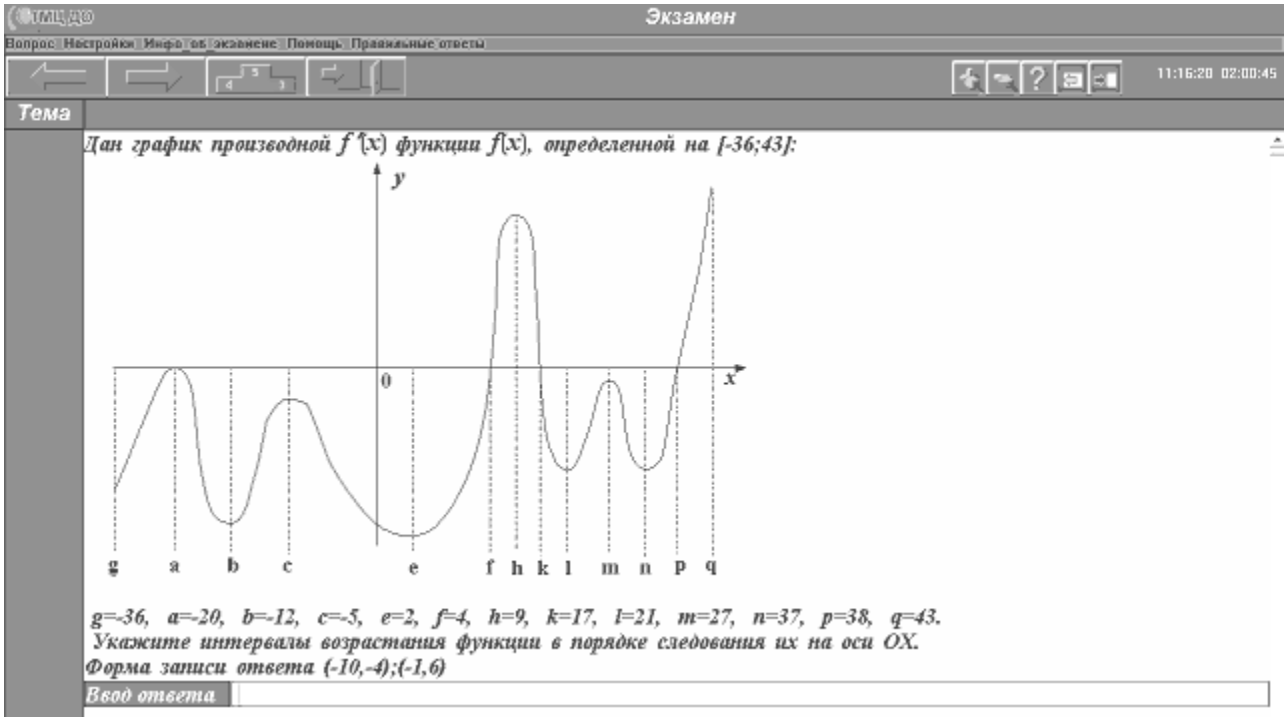


Рис. 6.2. Экранная форма конкретного вопроса

7. Подсчет числа вариантов

Примерный подсчет количества вариантов вопросов будет равен

$$V = P^n = 10^{13},$$

где P – число значений, которые принимают параметры;
n – число параметров.

Пример 3

1. Формулировка задания и метод генерации

Даны два списка устройств

RS-триггер	Двоичный сумматор
D-триггер	Дешифратор
T-триггер	Преобразователь кода
JK-триггер	Мультиплексор
Регистр памяти	Цифровой компаратор
Регистр сдвига	Пзу
Кольцевой регистр	Стабилизатор напряжения

Двоичный счетчик	Усилитель
Двоично-десятичный счетчик	Логический элемент 4И-НЕ
Озу	Сумматор по модулю два
Микропроцессор	

Случайно выбирается по 5 устройств из каждого списка. Объединяются и перемешиваются. Задание формулируется следующим образом: «В предложенном списке устройств указать те, которые реализуются на цифровых интегральных микросхемах последовательностного типа». В первом столбце таблицы записаны варианты ответов, которые являются правильными, относительно вопроса, во втором – неправильные.

2. Программная реализация

```
void Question1_1()
{
 int i, j;
 GText text(5000);
 GString Sright[11], Swrong[10], Var[10];
 //Записывается массив строк с правильными вариантами ответов
 Sright[0]<<"RS-триггер";
 Sright[1]<<"D-триггер";
 Sright[2]<<"T-триггер";
 Sright[3]<<"JK-триггер";
 Sright[4]<<"регистр памяти";
 Sright[5]<<"регистр сдвига";
 Sright[6]<<"кольцевой регистр";
 Sright[7]<<"двоичный счетчик";
 Sright[8]<<"двоично-десятичный счетчик";
 Sright[9]<<"ОЗУ";
 Sright[10]<<"микропроцессор";
 //записывается список с неправильными вариантами ответов
 Swrong[0]<<"двоичный сумматор";
 Swrong[1]<<"дешифратор";
 Swrong[2]<<"преобразователь кода";
 Swrong[3]<<"мультиплексор";
 Swrong[4]<<"цифровой компаратор";
 Swrong[5]<<"ПЗУ";
 Swrong[6]<<"стабилизатор напряжения";
 Swrong[7]<<"усилитель";
 Swrong[8]<<"логический элемент 4И-НЕ";
 Swrong[9]<<"сумматор по модулю два";
 //формулируется вопрос
```

```
 text<<"В предложенном списке устройств указать те, которые реа-
лизуются на цифровых интегральных микросхемах последователь-
ностного типа:\n";
 GQuestTabl quest(700); quest<<text;
 GMenu menu(1000);
 menu<<quest;
 //генерация верных вариантов
 MMixNum Right(0,10), Wrong(0,9), MixMenu(0,9);
//перемешаем и выберем случайно верные варианты ответов
 for(int i=0; i<5; i++)
 {
   Var[i]=Sright[Right.Get()];
 }
 //перемешаем и выберем случайно неверные варианты ответов
 for(int j=5; j<10; j++)
 {
   Var[j]=Swrong[Wrong.Get()];
 }
 //перемешаем массив из 5 вариантов ответов (верных и невер-
ных)
  for(i=0; i<10; i++)
  {
  GString s(20); s<<(i+1);
  int j=MixMenu.Get();
//формируем строку меню: vars var[0])(как верный вариант)
  if(j<5) {GVars v(50); v<<s<<") "<<Var[j]; menu<<v; }
 // все остальные строки меню будут varn
  else { GVarn v(50);  v<<s<<") "<<Var[j]; menu<<v; }
}
them<<menu;
}
```

3. Скрипт-файл для системы проведения контрольных работ

```
#num 1
#{#menu #size10#QuestTabl#Text "В предложенном списке уст-
ройств указать те, которые реализуются на цифровых интеграль-
ных микросхемах последовательностного типа:
"
#end
#varn1) сумматор по модулю два#end
#varn2) мультиплексор#end
#varn3) логический элемент 4И-НЕ#end
#vars4) Т-триггер#end
```

```
#vars5) D-триггер#end
#varn6) преобразователь кода#end
#varn7) усилитель#end
#vars8) двоично-десятичный счетчик#end
#vars9) регистр памяти#end
#vars10) ОЗУ#end
#}
```

4. Отображение полученного вопроса в системе проведения контрольных работ

На рис. 6.3 показана экранная форма конкретного вопроса, полученного на основании использования рассматриваемого генератора.

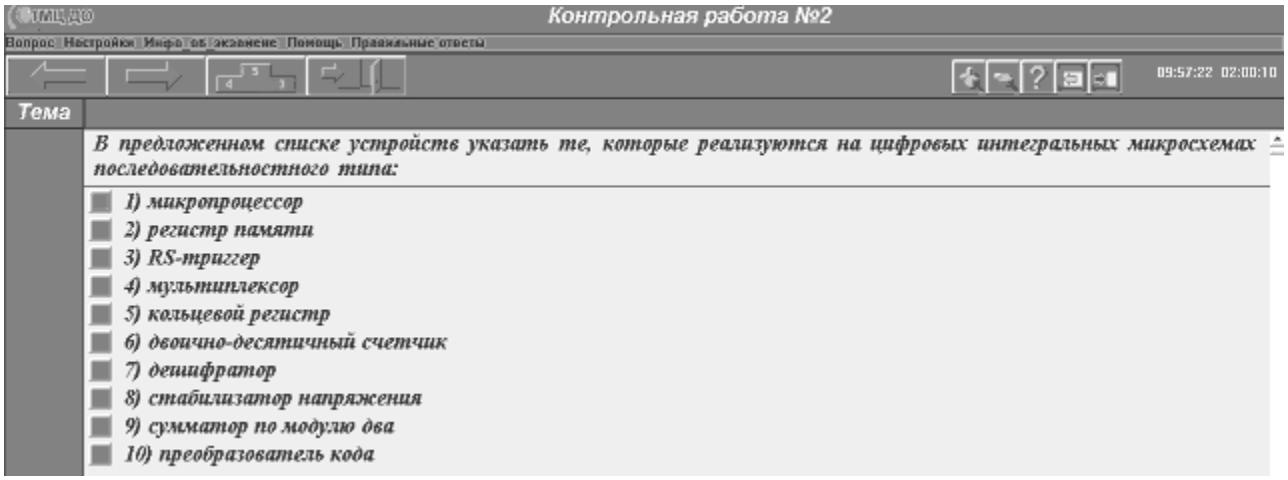


Рис. 6.3. Экранная форма конкретного вопроса

5. Подсчет числа вариантов

Подсчет числа вариантов можно осуществить по следующей формуле:

$$V = C_n^m \cdot C_k^l,$$

где C_n^m – число сочетаний из *n* элементов по *m; n* – размер списка правильных вариантов; *m* – число вариантов, выбираемых для конкретного вопроса; C_k^l – число сочетаний из *k* элементов по *l; k* – размер списка неправильных вариантов; *l* – число вариантов, выбираемых для конкретного вопроса.

Для нашего случая

$$V = C_{10}^5 \cdot C_9^5 = 252*126=31752.$$

Таким образом, данный генератор может сгенерировать 31752 различных вариантов вопроса.

Пример 4

1. Формулировка тестового задания

Это задание направлено на проверку знаний выполнения операции умножения двух матриц. Выдается список пар матриц с указанием размерностей, причем часть с правильным размерами матриц, часть с неправильным.

Укажите пары матриц, для которых произведение определено в указанном порядке:

1. $A[5\times 7]B[7\times 4]$

2. $X[7\times 3]Y[4\times 7]$

3. $C[5\times 5]D[4\times 4]$

4. $M[5\times 3]N[3\times 5]$

5. $S[5\times 6]T[7\times 5]$

6. $O[5\times 3]P[5\times 3]$

2. Метод генерации

1. Задается массив значений размерностей матрицы размерностью *k*, перемешивается и выбирается *n* чисел.
2. Генерируется полный список пар чисел как размещение из *n* по 2, размерность списка *m=n(n-1).*
3. Генерируется полный список пар матриц как размещение из *m* по 2. Общее число пар матриц будет равно *m(m-1).*
4. Из данного списка пар матриц формируется два – список правильных и список неправильных пар.
5. Из списка пар правильных выбирается I пара, из списка неправильных выбирается *q* пар.
6. Затем формируется общий список пар правильных и неправильных и перемешивается.

7. Формируется конкретный вопрос.

3. Программный код

```
QuestionExample_4()
{
 GText text(200);
 GQuestTabl quest(300);
 GMenu menu(1000);
 MArrangement <int> mx(2,4);
 MMixNum BorderValue(3,10);
 int Border[4];
 Tsk1Matrix **T=new Tsk1Matrix*[mx.Size()];
 for(int i=0; i<4; i++) Border[i]=BorderValue.Get();
 for(int i=0; i<4; i++)
  {
   mx[i]=&Border[i];
  }
 char ss[20];
 for(int i=0; i<mx.Size(); i++)
  {
   mx.Set(i);
   sprintf(ss,"%c[%dx%d]",('A'+i),*mx(0),*mx(1));
   T[i]=new Tsk1Matrix(*mx(0),*mx(1),ss);
  }
MArrangement <Tsk1Matrix> mult(2,mx.Size()); //Все множество
пар матриц
MStackNum Right(mult.Size()); //множество правильных вариантов
MStackNum False(mult.Size()); //множество неправильных вариан-
тов
for(int i=0; i<mx.Size(); i++) mult[i]=T[i];
int right=0;
for(int k=0; k<mult.Size(); k++)
  {
   mult.Set(k);
   right+=((mult(0)->ni==mult(1)->mi)?1:0);
   if(mult(0)->ni==mult(1)->mi)
    {
     Right.Push(k);
    }
    else False.Push(k);
  }
 for(int k=0; k<Right.Top(); k++)
  {
```

```
  mult.Set(Right[k]);
 }

MMixNum RightVar(0,Right.Top()-1);    //Три правильных
MMixNum FalseVar(0,False.Top()-1);    //Три неправильных
MMixNum Menu(0,5);
MStackNum Var(10);

Var.Push(Right[RightVar.Get()]);
Var.Push(Right[RightVar.Get()]);
Var.Push(Right[RightVar.Get()]);
Var.Push(False[FalseVar.Get()]);
Var.Push(False[FalseVar.Get()]);
Var.Push(False[FalseVar.Get()]);

text<<"Укажите пары матриц, для которых произведение опреде-
лено в указанном порядке\n";
 quest<<text;
 menu<<quest;
 GString ks(20);
 for(int k=0; k<Var.Top(); k++)
  {
   int j=Menu.Get();
   mult.Set(Var[j]);
   ks<<(k+1);
   if(mult(0)->ni==mult(1)->mi)
   {
    GVars vars(100);
    vars<<ks<<") "<<mult(0)->name<<" "<<mult(1)->name<<" ";
    menu<<vars;
   }
   else
    {
     GVarn varn(100);
     varn<<ks<<") "<<mult(0)->name<<" "<<mult(1)->name<<" ";
     menu<<varn;
    }
    ks.Clear();
  }
 //menu.Show();
 them<<menu;
 for(int i=0; i<mx.Size(); i++)
  {
   delete T[i];
  }
 delete []T;
```

```
}
```

4. Скрипт-файл для системы проведения контрольных работ

```
#{#menu #size06#QuestTabl#Text "Укажите пары матриц, для ко-
торых произведение определено в указанном порядке"
#end
#varn1) I[8x9] C[10x6] #end
#vars2)  I[8x9] H[9x10] #end
#varn3) E[8x6] A[9x6] #end
#vars4)  K[8x10] E[8x6] #end
#varn5) H[9x10] G[10x9] #end
#vars6)  L[10x8] E[8x6] #end
#}
```

5. Отображение полученного вопроса в системе экзаменов (рис. 6.4).

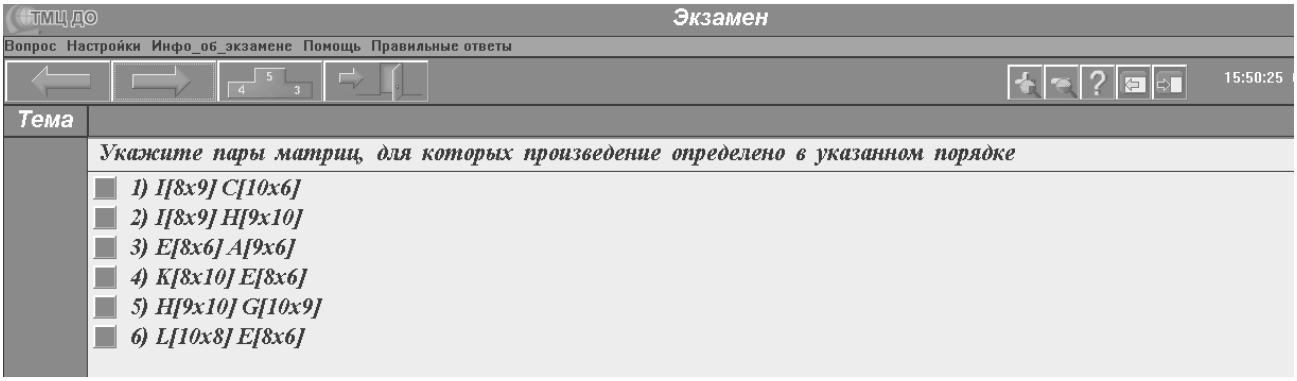


Рис. 6.4. Экранная форма конкретного вопроса

6. Подсчет числа вариантов

Подсчет числа вариантов основан на вычислении размещений:

1) число пар чисел – из n по два $m=n*(n-1);$

2) число пар матриц из m пар чисел по два $k=m*(m-1);$

3) разбиваем множество пар матриц на два, правильные (s) и неправильные (p).

Пример 5

1. Формулировка задания

Формализованные правила употребления артиклей, предлогов, обстоятельств (как частей речи английского языка) делают возможным использование методов генерации при составлении тестовых вопросов по одной из выбранных моделей, например, модели перечисления.

Модель перечисления подразумевает следующий алгоритм построения вопроса по принципу генерирования составляющих компонентов-блоков вопроса:

1) в зависимости от постановки вопроса методом мозгового штурма выделяются группы элементов-слов (количество которых может быть сколь угодно большим), которые соответствуют одному и тому же правилу (словообразования и др.);

2) случайным образом генерируется порядковый номер элемента (либо нескольких элементов) из одного множества, затем из другого;

3) выбранные элементы перемешиваются для последующего вывода на экран в виде альтернатив для выбора (вопрос типа меню); помечается один (либо несколько) верный вариант ответа.

Генерация параметров-составляющих и изменяемых элементов тестовых вопросов осуществляется с помощью классов (**Фраза** – **Phrase** в виде слова и соответствующего артикля, **Словарь** – **Dict**, Э**лемент словаря** – **WordInDic**, **Предложение** – **Sentence** как фиксированная последовательность единиц-слов, представляющая собой законченное высказывание и т.д.) и *шаблонов* (постановки вопроса и/или предлагаемых для выбора альтернатив). В шаблоне приводится описание подстановки сгенерированного(ых) параметра(ов) вопроса, последовательность заполнения структуры (предложения, списка альтернатив), фиксирование верного варианта(ов) ответа для сравнения с вводимым обучаемым с экрана в процессе тестирования.

Применительно к рассматриваемому вопросу *меню* удобно использовать структуру *Словарь*, состоящую из *Элементов сло-*

варя, в которых хранить *слово-альтернативу* ответа на вопрос (с возможными при необходимости *описаниями*).

2. Пример генерации вопроса типа меню

Отметьте глагол, окончание которого читается отлично от других.

1st group [s] – первая группа слов: works, talks, visits, speaks...
2nd group [z] – вторая группа слов: plays, goes, knows, opens...
3rd group [iz] – третья группа слов: watches, dances, washes, teaches...

Набор сгенерированных альтернатив может иметь вид: *Works, watches, washes, teaches,* из которых *works* является единственно верной.

3. Программный код

```
// правило чтения Present Simple формы глагола (окончания)
class Them1 {
 public:
  GThem them;
  Them1(char *name):them(name,10000){;}
  void Question1();
  void Gen() { Question1();}
  void Show(){ them.Show(); }
};

void Them1::Question1()
{
 GText text(2000);
 text<<" Из предложенного списка существительных выберите то, окончание множественной формы которого читается не по общему для остальных существительных правил:\n\n";
WordInDic word[3][10];
// 1st group [s]-Первая группа слов
word[0][0].Set("work",1,"works");
word[0][1].Set("talk",1,"talks");
word[0][2].Set("visit",1,"visits");
word[0][3].Set("speak",1,"speaks");
word[0][4].Set("sit",1,"sits");
word[0][5].Set("like",1,"likes");
word[0][6].Set("drink",1,"drinks");
```

```
word[0][7].Set("eat",1,"eats");
word[0][8].Set("sleep",1,"sleeps");
word[0][9].Set("put",1,"puts");
//2nd group [z]-Вторая группа слов
word[1][0].Set("play",1,"plays");
word[1][1].Set("go",1,"goes");
word[1][2].Set("know",1,"knows");
word[1][3].Set("open",1,"opens");
word[1][4].Set("add",1,"adds");
word[1][5].Set("ride",1,"rides");
word[1][6].Set("buy",1,"buys");
word[1][7].Set("do",1,"does");
word[1][8].Set("end",1,"ends");
word[1][9].Set("drive",1,"drives");
//3rd group [iz]-Третья группа слов
word[2][0].Set("watch",1,"watches");
word[2][1].Set("dance",1,"dances");
word[2][2].Set("wash",1,"washes");
word[2][3].Set("teach",1,"teaches");
word[2][4].Set("change",1,"changes");
word[2][5].Set("wish",1,"wishes");
word[2][6].Set("finish",1,"finishes");
word[2][7].Set("mix",1,"mixes");
word[2][8].Set("close",1,"closes");
word[2][9].Set("possess",1,"possesses");

 GQuestTabl quest(2000);
 GString Var[4];
 GMenu menu(5000);
 MMixNum group_num(0,2), word_num(0,9), MixMenu(0,3);
 int wordnum=0, wordn=0;
 quest<<text;
 menu<<quest;
 wordnum=group_num.Get();
 //генерируем список четырех альтернатив для выбора
 for(int j=0; j<3; j++)  {
   wordn=word_num.Get();
   Var[j]<<(char*)word[wordnum][wordn][0].c_str();
   }
 wordnum=group_num.Get();
 wordn=word_num.Get();
 Var[j]<<(char*)word[wordnum][wordn][0].c_str();
 //перемешаем случайным образом массив из 4 вариантов альтер-
натив:
 //одна верная и три неверных
  for(int i=0; i<4; i++)
```

```
  {
  GString s(20); s<<(i+1);
  int j=MixMenu.Get();
  if(j==3) {GVars v(1000); v<<s<<") "<<Var[j]; menu<<v; }
//формируем список 'меню': vars - верный вариант)
  else {GVarn v(1000);  v<<s<<") "<<Var[j]; menu<<v; }   // все остальные элементы списка 'меню' будут varn - неверные
  }
 them<<menu;
}
```

4. Скрит-файл для системы проведения контрольных работ

```
#{
 #menu #size04
  #QuestTabl
   #Text "Отметьте глагол, окончание которого читается отлично от других."
  #end
  #varn1) wahses #end
  #varn2) crashes#end
  #varn3) dances #end
  #vars4) takes  #end
#}
```

5. Экранная форма сгенерированного вопроса (рис. 6.5)

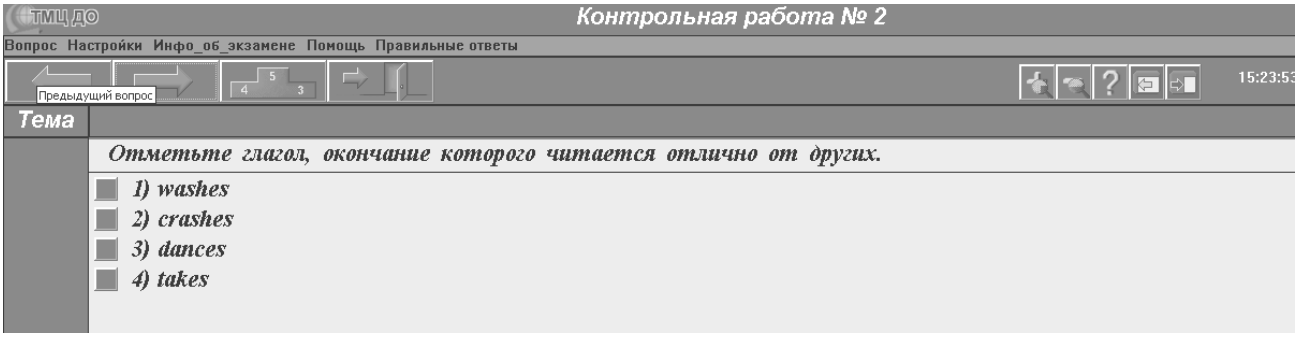


Рис. 6.5. Экранная форма конкретного вопроса

6. Подсчет числа вариантов

Имеется три множества слов. Для простоты возьмём одинаковую мощность, равную *n*. Тогда общее число вопросов будет вычисляться по следующей формуле:

$$V = A_3^2 \cdot C_n^3 \cdot C_n^1,$$

где A_3^2 – размещение из 3 по 2 – количество разных пар множеств; C_n^3 – количество сочетаний из *n* по три для формирования группы однотипных окончаний; C_n^1 – количество сочетаний из *n* по одному для формирования правильного варианта ответа.

При *n*=10, общее число вопросов будет 6*120*10=7200 вариантов. Увеличивая число *n*, можно получить более впечатляющий результат.

ЗАКЛЮЧЕНИЕ

Процесс постоянного усложнения программного обеспечения непосредственно касается компьютерных учебных программ. Генераторы становятся необходимым элементом программного обеспечения автоматизированного обучения и позволяют существенно увеличить качество процесса обучения. Генераторы обеспечивают получение тестовых заданий и вопросов при:

1) проведении различных видов тестирования уровня знаний;

2) проведении практических заданий в форме тренажа.

Важным свойством генератора является мощность генерируемого множества тестовых заданий. Здесь опыт показывает, что в КУП необходимо иметь генераторы, мощность которых обеспечивает практически каждому студенту индивидуальное тестовое задание.

Разработанный математический аппарат, основанный на алгоритмах генерации и идентификации комбинаторных объектов: перестановок, сочетаний и размещений, а также деревьев И/ИЛИ – обеспечивает фундамент для построения и исследования алгоритмов генерации вопросов и тестовых заданий.
Алгоритмы построения и перечисления вариантов деревьев И/ИЛИ позволяют построить универсальный метод генерации сложных информационных объектов.

Рассмотрено множество различных моделей и алгоритмов генерации вопросов и тестовых заданий. Показаны варианты построения генераторов, основанных на шаблонах задач и меню вопросов. Описаны универсальная структура шаблона и универсальный алгоритм работы генератора задачи по шаблону. Рассмотрены различные варианты реализации алгоритмов генерации задач. Приведены конкретные примеры шаблонов и исследованы их мощности.

На основе различных моделей предметной области: таблиц, двудольных графов и деревьев – разработаны конкретные алгоритмы генерации вопросов, исследованы их мощности и алго-

ритмы перечисления по номеру вопроса. Для каждой модели приведены конкретные примеры.

Внедрение генераторов в реальные системы контроля знаний требует учитывать особенности реализации таких систем. В Томском межвузовском центре дистанционного образования разработаны и эксплуатируются: система проведения контрольных работ и система проведения компьютерных экзаменов. В основе этих систем лежит язык описания тестов, который позволяет для формулировки вопросов использовать текст, рисунки, формулы и таблицы. Отличительной особенностью данного языка от других подобных является возможность описывать формулы, используя язык представления формул, разработанный в лаборатории инструментальных систем моделирования и обучения ТУСУРа. Показана возможность преобразования скрипт-программ тестов в формат языка HTML.

Кратко рассмотрена технология проведения компьютерного экзамена и контрольной работы, показаны возможности систем тестирования ТМЦДО. Приведена структура системы проведения контрольных работ.

Технология создания компьютерных тестов является важным элементом организации инфраструктуры процесса обучения. В ТМЦДО применяется оригинальная технология создания компьютерных тестов и генераторов, которая включает:

- прием задания;
- анализ множества вопросов на предмет реализации с помощью инструментальной системы проектирования КУП;
- доработку инструментальной системы;
- планирование работ;
- реализацию на языке инструментальной системы;
- отладку КУП средствами инструментальной системы;
- внутреннее тестирование;
- внешнее тестирование;
- формирование программы проведения контрольной работы или экзамена на основе итоговой программы проверки;
- рецензирование.

В приложении 2 приведены экзаменационные программы, выполненные по описанной выше технологии и которые эксплуатируются в ТМЦДО.

Показаны отличия в трудоемкости создания и отладки генераторов от обычных тестов с фиксированным банком вопросов. Особый интерес в связи с этим вызывают формальные модели генераторов и исследования, связанные с качеством вопросов, получаемых в ходе генерации. Однако эти исследования в данной работе не рассматриваются.

Программное обеспечение в конечном итоге выполняет функции по генерации тестовых заданий. Все программное обеспечение делится на уровни: базовый, средний и генераторы тестовых заданий и вопросов. В базовый уровень входят датчики случайных чисел и программы генерации комбинаторных объектов. В средний уровень входят классы для генерации описаний на языке представления тестов.

Рассмотрены механизмы встраивания генераторов в систему проведения контрольных работ и экзаменов. Предлагается использовать динамические библиотеки (DLL). Для этого разработан стандартный интерфейс и реализован в системах проведения компьютерных экзаменов и контрольных работ.

В ТМЦДО уже используется 20 генераторов, 926 шаблонов, объем программного обеспечения составляет 44627 строк исходного текста. В примерах реализации шаблонов показано число вариантов вопросов и задач, которые может генерировать шаблон. Общее число вопросов может быть порядка одного миллиарда. Эта цифра гарантирует индивидуальный набор вопросов для каждого студента ТМЦДО (из расчета количества студентов – десять тысяч).

Итак, разработка и внедрение генераторов являются перспективным направлением совершенствования программного обеспечения автоматизированного обучения, которое нацелено на повышение качества образовательного процесса.

ЛИТЕРАТУРА

1. *Бондарь В.А., Кобзев А.В., Кручинин В.В.* Стратегия построения программного обеспечения автоматизированного обучения в Томском межвузовском центре дистанционного образования // Труды IV Международной научно-практической конференции «Высшее образование: качество и интернационализация». Томск: Изд. ТПУ, 2000. С. 67.

2. *Дубнищева Т.Я., Мицель А.А., Веретенников М.В.* Новая версия электронного учебника по дисциплине «Концепция современного естествознания»//Открытое и дистанционное образование. 2002. №4.

3. *Томиленко В.А., Кручинин В.В., Борисов С.И., Долматов А.В., Дмитриева Н.М., Ситникова Е.А.* Реализация компьютерного учебника по математике для дистанционного образования//Тезисы докладов научно-методической конференции «Современное образование: качество и новые технологии». Томск: Томский гос. ун-т систем управления и радиоэлектроники, 2000. С. 88-89.

4. *Кобзев А.В., Бондарь В.А., Кручинин В.В., Ситникова Е.А.* Использование и разработка компьютерных контрольных работ для организации мониторинга учебного процесса в Томском центре дистанционного образования//Материалы научно-практического семинара: учебно-методическое обеспечение открытого инженерного образования/Под ред. Тихомирова В.П., Моисеева В.Б. Пенза, 2001. С. 90.

5. *Дмитриева Н.М., Кручинин В.В., Ситникова Е.А.* Технология разработки компьютерных учебных работ и экзаменаторов в Томском межвузовском центре дистанционного образования// Тезисы докладов семинара «Электронные учебники и учебно-методические разработки в открытом образовании». М.: МЭСИ, 2000. С. 69-70.

6. *Воронин А.И., Исакова О.Ю., Кручинин В.В.* Проблемы создания и сопровождения базы компьютерных учебных программ в Томском межвузовском центре дистанционного образования//Организация дистанционного обучения в Томском межву-

зовском центре дистанционного образования // Тез докл. региональной конф. «Современное образование: Система и практика обеспечения качества». Томск: ТУСУР, 2002. С. 103.

7. *Исакова О.Ю., Кручинин В.В., Миллер А.В.* Технология мониторинга знаний в Томском межвузовском центре дистанционного образования // Труды научно-практической конф. «Информационные недра Кузбасса». Кемерово, 2003. С. 180-181.

8. System Of Realization Of Common Information Space In Education Process/I. Panfilov, D. Karminskaya, V. Bonlar, V. Kruchinin //2-nd WBLE Conference-2001, 24-26 October. Lund University. P. 121-126.

9. *Носуленко А.В.* Проект создания корпоративной информационной системы «Лоцман.edu» ТМЦДО. Результаты 2002, планы на 2003 год // Материалы отчетной конференции Томского межвузовского центра дистанционного образования. Томск: Изд-во ТУСУР, 2002. С. 21-26.

10. *Кобзев А.В., Бондарь В.А., Воронин А.И., Кручинин В.В., Миллер А.В., Уваров А.Ф., Хлопотникова Н.И., Шарапов А.В.* Организация дистанционного обучения в Томском межвузовском центре дистанционного образования//Тез докл. региональной конф. «Современное образование: Система и практика обеспечения качества». Томск: Изд-во ТУСУРа, 2002. С. 98-101.

11. Переход от кейсовой технологии дистанционного обучения к виртуальному университету / Кручинин и др. // Современное образование: Интеграция учебы, науки производства: Материалы региональной научно-методической конф. Томск: изд-во ТУСУРа, 2003. С. 79-80.

12. *Кручинин В.В.* Проблемы и пути решения защиты компьютерных учебных программ//Доклады третьей научно-практической конф. «Современные средства и системы автоматизации – гарантия высокой эффективности производства». Томск, 2002. С. 328-330.

13. *Исакова О.Ю., Кручинин В.В.* Автоматизация синтеза вопросов в компьютерных учебных программах//Доклады третьей научно-практической конф. «Современные средства и системы автоматизации – гарантия высокой эффективности производства». Томск, 2002. С. 302-305.

14. *Кручинин В.В.* Разработка компьютерных учебных программ. Томск: изд-во Томск. ун-та. 1998. 211 с.

15. *Башмаков А.И., Башмаков И.А.* Разработка компьютерных и обучающих систем. М.: Информационно-издательский дом «Филинъ», 2003. 616 с.

16. *Кручинин В.В., Магазинников Л.И.* О повышении качества контролирующих материалов // Материалы научно-методической конференции. Томск: Изд-во ТУСУР, 2002. 69 с.

17. *Кашкарева Н.В., Кручинин В.В., Лычковская Л.Е.* Генератор тестовых заданий по дисциплине «Английский язык»//Современное образование: Интеграция учебы, науки производства: Материалы региональной научно-методической конф. Томск: Изд-во ТУСУРа, 2003. С. 49-50.

18. *Егоркина Ю.В., Кручинин В.В., Шарапов А.В.* Пакет генераторов тестовых заданий по циклу «Цифровые микропроцессорные устройства»// Современное образование: Интеграция учебы, науки производства: Материалы региональной научно-методической конф. Томск: Изд-во ТУСУРа, 2003. С. 86-87.

19. *Перегудов Ф.И., Тарасенко Ф.П.* Основы системного анализа. Томск: Изд-во НТЛ, 1997. 396 с.

20. Словарь иностранных слов. М.: Рус. яз., 1989. 662 с.

21. *Первозванский А.А.* Курс теории автоматического управления: Учеб. пособие. М.: Наука, 1986.

22. *Игнатов В.А.* Теория информации и передачи сигналов. М.: Радио и связь, 1991. 279 с.

23. *Дмитриев В.И.* Прикладная теория информации. М.: Высшая школа, 1989. 319 с.

24. *Белоглазов И.Н., Тарасенко В.П.* Корреляционно-экстремальные системы. М.: Сов. радио, 1974. 392 с.

25. *Фу К., Гонсалес Р., Ли К.* Робототехника. М.: Мир, 1989.

26. *Амосов Н.М., Байрон Т.Н., Гольцев А.Д.* и др. Нейрокомпьютеры и интеллектуальные роботы. Киев: Наук. думка, 1991. 271 с.

27. *Горбань А.Н.* Обучение нейронных сетей. М.: С.П. «ПараГраф», 1990. 56 с.

28. Методы Монте-Карло в статистической физике / К. Биндер, Д. Сиперли, Ж.-П. Ансен и др. М.: Мир, 1982. 400 с.

29. *Прицкер А.* Введение в имитационное моделирование и язык СЛАМ II. М.: Мир, 1997. 646 с.

30. *Буймов А.Г.* Корреляционно-экстремальная обработка изображений. Томск: Изд-во Том. ун-та, 1987. 132 с.

31. http://www.csa.ru/~star/cad/3d.htm

32. *Кнут Д.* Искусство программирования. т 2.

33. *Баас Р., Фервай М., Гюнтер Х.* Delphi 4: полное руководство: Пер. с нем. Киев: BHV, 1998. 800 с.

34. http://www.abxsoft.com/pcyacc.htm

35. http://www.rvb.ru/soft/catalogue/catalogue.html#TextGenerators

36. *Альтшуллер Г.С.* Найти идею. Новосибирск, 1986.

37. *Половинкин А.И.* Основы инженерного творчества. М., 1988.

38. http://www.creax.com/tools/software.html

39. *Гульден Я.*, Джексон Д. Перечислительная комбинаторика. М.: Наука 1990. 504 с.

40. *Стенли Р.* Перечислительная комбинаторика. М.: Мир, 1990. 440 с.

41. *Akl S.G*. A comparison of combination generation methods, ACM Trans. of Math. Software, 7(1981). p. 42-45.

42. *Akl S.G*. Adaptive and optimal parallel algorithms for enumerating permutations and combinations, The Computer Journal, 30(1987). p. 433-436.

43. *Slagle J.R.* A Heuristic Program that Solves Symbolic Integration Problems in Freshmen Calculus In E. Feigenbaum and J. Feldman, editors, Computer and Thought, pages 192-203, McGraw-Hill, New York, 1963.

44. *Nilsson N.* Principles of Artifical Intelligence. Spring Verlag, 1983.

45. *Ефимов Е.И.* Решатели интеллектуальных задач. М.: Наука, 1982. 320 с.

45. *Братко И.* Программирование на языке Пролог для искусственного интеллекта. М.: Мир, 1990-560с.

47. *Хант Э.* Искусственный интеллект. М.: Мир, 1978. 558 с.

48. *Рейуорд–Смит В. Дж.* Теория формальных языков. Вводный курс. М.: Радио и связь, 1988. 129 с.

49. *Ахо А., Ульман Дж.* Теория синтаксического анализа, перевода и компиляции. Т.1. Синтаксический анализ. М.: Мир, 1978. 535 с.

50. *Ландо С.К.* Комбинаторика. М.: изд-во независимого москов. ун-та, 1994. 78 с.

51. *Erkki Makinen* Ranking and unranking left szilard languages. University of Tampere, Departament of Computer science, Series of publications A A-1997-2, January, 1997. 10 p.

52. *Федотов М.И.* Программно-методический комплекс эффективной поддержки педагогического тестирования http://www.nsu.ru/archive/ conf/nit/96/sect1/node20.html

53. *Амзараков М.Б.* Автоматическая генерация вариантов педагогического теста http://ito.bitpro.ru/1999/II/6/6140.html

54. *Левинская М.А*. Автоматизированная генерация заданий по математике для контроля знаний учащихся//Educational Technology & Society 5(4) 2002. ISSN 1436-4522 –c214-221.

55. *Успенский В.А., Семенов А.Л.* Теория алгоритмов: основные открытия и приложения. М.: Наука, 1987. 288 с.

56. *David R. Musser* Generic Programming. www.cs.rpi.edu /~musser/ gp/index.html

57. *Буч Г., Рамбо Д., Джекобсон А.* Язык UML. Руководство пользователя. М.: ДМК, 2000. 432 с.

58. *Гаврилова Т.А., Хорошевский В.Ф.* «Базы знаний интеллектуальных систем». СПб.: Питер, 2000. 384 с.

59. *Шикунов С.А.* Метод построения учебного гипертекста по ключевым словосочетаниям http://vio.fio.ru/vio_10/cd_site/ articles/art_1_7.htm

60. *Мейер Д.* Теория реляционных баз данных. М.: Мир, 1987.

61. *Шпинар З.В.* История жизни на земле. Прага: Артия, 1977. 228 с.

62. *Евстигнеев В.А.* Применение теории графов в программировании. М.: Наука, 1985. 352 с.

63. http://kuking.net/2_86.htm

64. Жизнь животных. Т.5. Земноводные. Пресмыкающиеся. М.: Просвещение, 1985. 399 с.

65. Организация всестороннего контроля знаний студентов в дистанционной технологии обучения/ Кручинин В.В. и др. //Новые информационные технологии в университетском образовании. Тезисы докладов международной научно-метод. конф. Новосибирск, СГУТиИ, 2003. С. 72-73.

66. *Воронин А.И., Кручинин В.В., Миллер А.В., Сербин Э.Ф., Шарапов А.В.* Учебно-методическое и программное обеспечение дистанционной технологии обучения: Учебно-методическое по-

собие. Томск: Томский межвузовский центр дистанционного образования, 2003. 60 с.

67. *Михайлычев Е.А.* Дидактическая тестология. М.: Народное образование, 2001. 432 с.

68. *Майоров А.Н.* Теория и практика создания тестов для системы образования (как выбирать, создавать и использовать тесты для целей образования). М.: Народное образование, 2000.- 352с.

69. *Анастази Л.* Психологическое тестирование: В 2 кн. М., 1982.

70. *Панфилов С.А.* Контроль знаний на ЭВМ. Учебное пособие. Саранск: Морд. ун-т, 1989. 76 с.

71. *Свиридов А.П.* Основы статистической теории обучения и контроля знаний. М.: Высш. шк., 1981. 262 с.

72. *Кандрашина Е.Ю., Литвинцева Л.В., Поспелов Д.А.* Представление знаний о времени и пространстве в интеллектуальных системах. М.: Наука, 1989. 328 с.

73. Тезисы докладов Всероссийской научно-методической конференции «Развитие тестовых технологий в России». М., 2002. 320 с.

74. *Вирт Н.* Алгоритмы + структуры данных = программы: Пер. с англ. М.: Мир, 1985. 406 с.

75. *Секунов Н.Ю.* Самоучитель Visual C++. СПб.: БХВ-Петербург, 2001. 960 с.

76. *Рейсдорф К.* Borland C++ Builder 3. Освой самостоятельно. М.: ЗАО «Издательство БИНОМ», 1999. 736 с.

77. *Дик О., Глен Ф.* Популярные Web-броузеры. Энциклопедия пользователя. Киев: ДиаСофт, 1998. 464 с.

78. *Матросов А.В., Сергеев А.О., Чаунин М.П.* HTML 4.0. СПб.: БХВ-Петербург, 1999. 672 с.

79. http://www.webuniversity.ru/tor/review.html

80. http://www.imsglobal.org

81. http://www.imsproject.org/question/qtiv1p2
/imsqti_oviewv1p2.htm/#1399104

82. http://www.xdlsoft.com/

83. *Митюнин В.А.* Обзор средств публикации и просмотра математических документов в сети Интернет
http://tex.msu.ru/mitjunin/mml.htm

84. *Хэлворсон М., Янг М.* Эффективная работа в Microsoft Office 2000. СПб.: Питер, 2001. 1232 с.

85. *Тайц А.* Эффективная работа с Photoshop 6. СПб.: Питер, 2001. 731 с.

86. *Борн Г.* Форматы данных: Пер. с нем. Киев: BHV, 1995. 472 с.

87. *Орлов С.* Технологии разработки программного обеспечения. СПб.: Питер, 2003. 430 с.

88. *Липаев В.В , Штрик А.А.* Технология сборочного программирования. М.: Радио и связь, 1992.

89. *Калянов Г.* Case-технологии: консалтинг в автоматизации бизнес-процессов. СПб.: Питер, 2002. 320 с.

90. *Буч Г., Рамбо Дж., Якобсон А*. Унифицированный процесс разработки программного обеспечения. СПб.: Питер, 2002. 496 с.

91. *Трофимов С.* Case-технологии. Практическая работа в Rational Rose. СПб.: Питер, 2002. 288 с.

92. *Бобровский С.* Delphi 5: учебный курс. СПб.: Питер, 2000. 640 с.

93. ISO/IEC 14882:1998. Programming languages -- C++. Languages: Edition: XN. Pages: 732.

94. http://www.stlport.org/

95. *Pierre L'Ecuyer* Good Parameter Sets for Combined Multiple Recursive Random Number Generators: Operations Research, no. 1, 1999. Vol. 47. p. 159-164.

96. *George Marsaglia and Arif Zaman* A New Class of Random Number Generators. Annals of Applied Probability, no. 3. vol. 3. p. 462-480.

97. *M. Matsumoto and T. Nishimura, Mersenne Twister*: A 623-dimensionally Equidistributed Uniform Pseudorandomnumber Generator: ACM TOMACS, no. 1. vol. 8. pp. 3-30. 17 December. 1998.

98. *Stephen K. Park and Keith W. Miller*, Random Number Generators: Good Ones Are Hard to Find.: CACM, no. 10. vol. 31. p. 1192-1201. October. 1988.

99 *Мейерс С.* Эффективное использование STL. Библиотека программиста**.** СПб.: Питер, 2003. 224 с.

100. *Рихтер Дж.* Windows для профессионалов: создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows. СПб.: Питер, 2001. 752 с.

101. *Чеппел Д.* Технологии ActiveX и OLE /Microsoft Press. Understanding ActiveX and OLE/ ISBN 5-7502-0029-9, 1997. 320 с.

102. *Дейл Роджерсон*. Основы COM. 2 изд-е. М.: Русская редакция, 2000.

103. *Дональд Бокс.* Сущность технологии COM. СПб.: Питер, 2001.

104. *Вильямс А.* Системное программирование в Windows 2000 для профессионалов. СПб.: Питер, 2001. 624 с.

105. *Шеффилд Дж*. Программирование на Microsoft Visual C++.NET/пер с англ. М.: Русская редакция, 2003.-928 с.

106 *Джекобсон Р.* Microsoft Office 2000: автоматизация и Интернет-возможности –М.:Русская редакция, 2000. 352 с.

107. *Андерсон Т.* Visual Basic: шаг за шагом. М.: Бином, 1998. 224 с.

108. *Савинов А.А.* Применение иерархических многомерных пространств для описания структуры предметной области http:// www.ais.fraunhofer.de/~savinov/publicat/ami98_1.html

109. IMS Question & Test Interoperability: ASI XML Binding Specification, C.Smythe, E.Shepherd, L.Brewer and S.Lay, Final Specification, Version 1.2, IMS, February, 2002.

Приложение 1. Описание методов класса ***KruVTable***

```
#include "KruMethodNext.h"
///
int KruVTable::Insert(char *name, int t,  char *format, KruFunc
pfunc){
 KruMethod x(t,format,pfunc);
 VTable.insert(value_type(string(name),x));
}
/////
int KruVTable::Find(char *name){
 vtable_type::iterator p = VTable.find(string(name));
  if (p != VTable.end()) return 1;
  return 0;
}
//////
int KruVTable::Call(char *name,...){
 Variant Params[SIZEVARIANTPARAMS];   //массив параметров
 int top=0;           //номер параметра
 va_list ap;          //для извлечения из стека
                  //используются макросы stdarg
 p = VTable.find(string(name)); //найти метод
 if (p == VTable.end()) return 0;
 va_start(ap,name); //выделяем параметры из стека
 string::iterator s=p->second.format.begin();
 while(s!=p->second.format.end()) {  //выделяем параметры в
соответствии с форматом
  switch(*s){
   case 'i': //для целого параметра
      Params[top].Param.pi = va_arg(ap,int);
      Params[top++].type=PINT;
         break;
   case 'x': //для адреса
      Params[top].Param.px = va_arg(ap,void*);
      Params[top++].type=PADDR;
         break;
   case 'f': //для float
      Params[top].Param.pf = va_arg(ap,float);
      Params[top++].type=PFLOAT;
         break;
   case 'd': //для double
      Params[top].Param.pd = va_arg(ap,double);
      Params[top++].type=PDOUBLE;
         break;
  }
  s++;
```

```
 }
 unsigned long offset=0;
 unsigned long off=4;          //сдвигаем адрес возврата
 for(int i=top-1; i>=0; i--){   //заносим в стек параметры с конца
списка
   _ESP-=off;                   //параметров
   switch(Params[i].type){
 case PINT:{             //заносим целое значение
   *((int *)_ESP)=Params[i].Param.pi;;
    offset+=sizeof(int);
    off=sizeof(int);
     }
    break;
 case PADDR:{            //заносим адресс
    *((void **)_ESP)=Params[i].Param.px;;
    offset+=sizeof(void *);
    off=sizeof(void *);
    }
    break;
 case PFLOAT: {          //заносим плавающий формат
   *((float*)_ESP)=Params[i].Param.pf;;
    offset+=sizeof(float);
    off=sizeof(float);
    }
   break;
 case PDOUBLE: {         //заносим double
   *((double*)_ESP)=Params[i].Param.pd;;
    offset+=sizeof(double);
    off=sizeof(double);
    }
  break;
 }
}
 p->second.method(); //выполнить метод
 _ESP+=offset;
 return 1;
}
////////////////////////////////////////////////////
int KruVTable::Call(char *name,KruParams *params){
 p = VTable.find(string(name)); //найти метод
 if (p == VTable.end()) return 0;
p->second.method(params); //выполнить метод
}
int KruVTable::Call(char *name,void *This,KruParams *params){
 p = VTable.find(string(name)); //найти метод
 if (p == VTable.end()) return 0;
```

```
 p->second.method(This,params); //выполнить метод
}

//////////////////////class
int KruClassVTable::Insert(char *name, int t,  char *format, KruMPtr
pfunc){
  KruClassMethod x(t,format,pfunc);
  VTable.insert(value_type(string(name),x));
}
/////
int KruClassVTable::Find(char *name){
  vtable_type::iterator p = VTable.find(string(name));
   if (p != VTable.end()) return 1;
   return 0;
}
//////////////////////
int KruClassVTable::Call(char *name,KruMethodPointer
*This,KruParams *params){
  p = VTable.find(string(name)); //найти метод
  if (p == VTable.end()) return 0;
 (This->*p->second.method)(params); //выполнить метод
}
```

Приложение 2. Каталог компьютерных экзаменов ТМЦДО на 10.11.2003

№	Название	Автор	Специальности	Количество вопросов
1	Автоматизация проектирования программных средств –2 (моделирование систем –2)	Черкашин М.В.	220500, 220300, 200100, 071700, 075300, 200307, 200400, 220500, 220507, 351000,	63
2	Автоматизированные информационные технологии и аппаратура	Ульянов В.Н.	201100, 201400, 201500	100
3	Автоматизированные комплексы распределенного управления	Рождественский Д.А.	210100	100
4	Алгоритмические языки и технология программирование –1	Кручинин В.В.	220200	100
5	Алгоритмические языки и технология программирование –2	Кручинин В.В.	220200	97
6	Анализ финансово-хозяйственной деятельности предприятия	Хмельницкая Е.Б.	060100, 060200	100
7	Аналоговая и цифровая схемотехника	Шибаев А.А.	220500, 220507	121
8	Английский язык-1	Лычковская Л.Е.	013100, 060400, 060800, 061000, 071700, 075300, 075500, 200100, 200307, 200400, 200700, 200800, 201100, 201400, 201500, 210100, 220200, 220300, 220400, 220500, 220507, 350500, 351000, 351400, 552800	384
9	Английский язык-2	Лычковская Л.Е.	013100, 060400, 060800, 061000, 071700, 075300, 075500, 200100, 200307, 200400, 200700, 200800, 201100, 201400, 201500, 210100, 220200, 220300, 220400, 220500, 220507, 350500, 351000, 351400,	130

			552800	
10	Английский язык-3	Лычковская Л.Е.	013100, 060400, 060800, 061000, 071700, 075300, 075500, 200100, 200307, 200400, 200700, 200800, 201100, 201400, 201500, 210100, 220200, 220300, 220400, 220500, 220507, 350500, 351000, 351400, 552800	145
11	Архитектура вычислительных систем и сетей ЭВМ (Вычислительные системы, сети и телекоммуникации)	Поникоровский С.В.	060800, 20400, 351400, 351000	99
12	Базы данных	Красина Ф.А.	060400, 060800, 552800	85
13	Базы данных	Муравьев А.И.	200400, 220500, 220507, 351000, 200100, 200307, 060400, 060800	86
14	Базы данных (Базы данных в САПР)	Карминская Т.Д.	210100, 220300	70
15	Банковское дело		060800, 351000, 351400	84
16	Безопасность жизнедеятельности	Ципилева Т.А.	031000, 031300, 060400, 060800, 200100, 200307, 200400, 200700, 200800, 201100, 201400, 201500, 210100, 220200, 220300, 220400, 220500, 220507, 351400	92
17	Бизнес – планирование	Байкалова А.И.	351000	100
18	Биология –2	Карташев А.Г.	13100	100
19	Бухгалтерский учет	Григорьева М.В.	060800, 351400	100
20	Бухгалтерский учет	Крискевич С.Н.	351000	100
21	Бухгалтерский учет		60400	150
22	Бюджетная система Российской Федерации	Тухватулина Л.А.	60400	42

23	Введение в вычислительную технику	Давыдова Е. М.	220500, 075500, 220507	106
24	Введение в психологическую деятельность	ТГПУ	31000	98
25	Введение в специальность (социальная работа)	Лавровский Н.А.	350500	24
26	Возрастная анатомия, физиология и гигиена	Хоч И.С.	31000	99
27	Высшая математика –1	Магазинников Л.И.	013100, 060400, 060800, 061000, 071700, 075300, 075500, 200100, 200307, 200400, 200700, 200800, 201100, 201400, 201500, 210100, 220200, 220300, 220400, 220500, 220507, 351000, 351400, 552800	121
28	Высшая математика –2	Магазинников Л.И.	013100, 071700, 075300, 075500, 200700, 200800, 201100, 201400, 201500, 210100, 220200, 220300, 220400, 220500, 220507, 351000, 351400, 552800, 200100, 200307, 200400	155
29	высшая математика -4	Магазинников Л.И.	013100, 060400, 060800, 071700, 075300, 075500, 200100, 200307, 200400, 200700, 200800, 201100, 201400, 201500, 210100, 220200, 220300, 220400, 220500, 220507, 351400, 552800	160
30	Высшая математика -5	Магазинников Л.И.	071700, 075300, 075500, 200100, 200307, 200400, 200700, 200800, 201100, 201400, 201500, 210100, 220200, 220300, 220400, 220500, 220507, 351400, 552800	360
31	Вычислительная математика – 2	Мицель А.А.	220400	100

32	Вычислительная математика – 3	Ельцов А.А.	060400, 060800, 013100, 071700, 075300, 075500, 200100, 200307, 200400, 200700, 200800, 201100, 201400, 201500, 210100, 220200, 220300, 220400, 220500, 220507, 351000, 351400, 552800	107
33	Вычислительная техника –3 (Цифровые устройства и микропроцессоры-2)	Кормилин В.А.	201400	110
34	Вычислительные методы	Бабак Л.И.	220300	118
35	Геохимия и геофизика атмосферы	Воскресенский В.В.	13100	100
36	Государственное регулирование экономики	Горбатог З.В.	060800, 061000	99
37	Делопроизводство и корреспонденция	Красина Ф.А.	60800	80
38	Деньги, кредит, банки	Тухватулина Л.А.	60400	91
39	Диагностика аудиовизуальной аппаратуры	Кирпиченко Ю.Р.	201400, 201500	100
40	Дискретная математика	Пермякова Н.В.	0610006 351000	138
41	Дискретная математика –1	Зюзьков В.М.	220507, 351400	110
42	Дискретная математика –1		210100, 220300, 220400	100
43	Документоведение	Исакова А.И.	75300	155
44	Измерительная техника и датчики	Сидоров Ю.К., Эрастов В.Е.	200400	110
45	Инженерная и компьютерная графика	Беляева Е.П.	071700, 200100, 200700, 200800, 201400, 201500, 220500, 220507, 201100, 075300, 200307, 200400, 210100, 220300	50

46	Инновационное предпринимательство	Семиглазов В.А.	351000	98
47	Интерактивные системы	Силич М.П.	61000	130
48	Интерфейсы АСОИУ –2	Силич М.П.	220200	130
49	Информатика	Миньков С.Л.	351400	99
50	Информатика	Никитин К.В.	031000, 031300, 220500	100
51	Информатика	Потахова И.В.	060400, 061000, 220400, 350500, 552800, 060800	50
52	Информатика –1	Одиноков В.В.	210100, 220300	200
53	Информатика –1 (Информатика и математическое моделирование –1)	Муравьев А.И.	060400, 060800, 071700, 075300, 075500, 200307, 200400, 200700, 200800, 201100, 201400, 201500, 220200, 220500, 220507, 351000, 200100	93
54	Информатика –2 (Информатика и математическое моделирование –2)	Муравьев А.И.	060400, 060800, 071700, 075300, 075500, 200307, 200400, 200700, 200800, 201100, 201400, 201500, 220200, 220500, 220507, 351000, 200100	69
55	Информатика -3	Четвергов К.В.	200700,200800, 201100, 201400, 201500,	50
56	Информационное обеспечение САПР (Информационное обеспечение систем управления)	Карминская Т.Д.	220300, 210100	101
57	Информационные технологии в экономике	Исакова А.И.	075300, 351000, 351400	217
58	Исследование систем управления –2	Рыбалова Е.А.	61000	102
59	История социальной работы в России	Казакевич Л.И., Зиновьева В.И.	350500	200
60	История экономики	Силютина В.А.	60800	100

61	История экономических учений	Алферова Л.А.	060800, 061000, 351400	168
62	Квантовые и оптоэлектронные приборы и устройства	Шангин А.С.	200307	102
63	Комплексирование бытовой радиоэлектронной аппаратуры	Озеркин Д.В.	201400, 201500,	100
64	Компьютерная графика	Буймов Б.А.	060800, 220200, 552800	100
65	Компьютерное моделирование	Лузина Л.И.	200307, 220400	100
66	Конструирование и технология электронных устройств	Бацула А.П.	200700, 201500, 200400	100
67	Концепции современного естествознания	Мицель А.А.	031000, 031300, 350500, 351400, 060400, 0610	100
68	Культура поведения и деловой этикет	Гныря Е.С.	350500	189
69	Культурология		013100, 060400, 060800, 061000, 071700, 075300, 075500, 200100, 200307, 200400, 200700, 200800, 201100, 201400, 201500, 210100, 220200, 220300, 220400, 220500, 220507, 351000, 351400, 552800	100
70	Лизинг (Управление проектами)	Горбатог З.В.	351000, 060800	82
71	Логистика	Тухватулина Л.А.	60800	98
72	Магнитные элементы электронных устройств	Обрусник В.П.	200400	93
73	Маркетинг (Экономика и менеджмент)	Алферова Л.А.	060400, 060800, 061000 200100, 200400, 351000, 351400, 200307, 200800, 220500, 220507	99
74	Математическая логика и теория алгоритмов	Шелупанов А.А.	75500	106
75	Материаловедение и материалы ЭВС	Солдатова Л.Ю.	200800, 220500, 220507	100
76	Материалы электронной техники и методы их анализа –1	Давыдов В.Н.	200307	100

77	Материалы электронной техники и методы их анализа –2	Давыдов В.Н.	200307	100
78	Машинная графика и геометрическое моделирование	Поляков А.Ю.	220300	100
79	Метрология	Эрастов В.Е.	060800, 201500, 200307, 200400, 200800, 220507, 210100, 201400, 200700, 201100, 200100	98
80	Микропроцессорные устройства систем управления –1	Рождественский Д.А.	210100	100
81	Микроэкономика	ТЭПТ	060100, 060200	54
82	Мировая экономика	Тухватулина Л.А.	060400, 351000, 351400	46
83	Моделирование систем -1	Шевченко Н.Ю.	220200	101
84	Моделирование систем -2	Шевченко Н.Ю.	220200	100
85	Моделирование систем управления (методы анализа и расчета электронных схем)	Бондарь В.А., Воронин А.И.	200400, 210100	173
86	Муниципальный менеджмент	Кудряшова Л.В.	61000	104
87	Налоги и налогообложение	Красина, Ф.А. Алферова Л.А.	351000	202
88	Налоги с физических лиц	Колыхаева М.В.	60400	100
89	Налоговая система	Алферова Л.А	60800	140
90	Налогообложение предприятий	Алферова Л.А.	60400	135
91	Науки о Земле		13100	100
92	Обработка экспериментальных данных на ЭВМ	Катаев М.Ю.	220400	96
93	Общая экология –2	Смирнов Г.В., Зиновьев Г.Г.	13100	136

94	Объектно-ориентированное программирование	Катаев М.Ю.	220300, 220400	44
95	Операционные системы	Раводин О.М.	75500	103
96	Оптические системы передачи информации	Шангин А.С., Шангина Л.И.	200700	148
97	Оптоэлектронные приборы и устройства СВЧ	Соколова Ж.М., Кущ Г.Г., Шангина Л.И.	201400, 201500	222
98	Организация баз данных	Сенченко П.В.	220200, 061000	100
99	Организация и функционирование ЭВМ	Поникоровский С.В.	220400	100
100	Организация производства		60800	70
101	Организм и среда –2		13100	100
102	Основы автоматизированного проектирования радиоэлектронной аппаратуры	Кологривов В.А.	200700	115
103	Основы алгоритмизации и языки программирования	Сафьянова Е.Н.	351400	100
104	Основы аудита	Тухватулина Л.А.	060400, 351000	100
105	Основы бизнеса и предпринимательства	Камышев Э.Н.	200400, 351000, 351400,	120
106	Основы внешнеэкономической деятельности	Чигоряев К.Н.	60800	85
107	Основы налогообложения	Красина Ф.А.	060800, 060400	99
108	Основы права	ТЭПТ	060100, 060200	50
109	Основы преобразовательной техники	Семенов В.Д.	200400	95
110	Основы социальной медицины	Макарова К.В.	350500	140
111	Основы телевидения	Казанцев Г.Д.	200700, 201400, 201500, 201100	100
112	Основы теории управления	Кориков А.М.	220400	100
113	Основы теории цепей -1	Коновалов Б.И	200100, 200307, 200400, 200700, 201400, 200800, 201500, 071700, 075300, 201100	74

114	Основы теории цепей –2 (Теория электрических цепей -2)	Обрусник В.П.	200100, 200307, 200400, 200800	100
115	Основы теории цепей –2 (Теория электрических цепей -2)	Тельпуховская Л.И., Каминский В.Л.	200700, 201400, 201500, 071700, 075300, 201100	120
116	Основы экономической теории	ТЭПТ	060100, 060200	45
117	Основы электроники –1	Малютин Н.Д., Вершинин И.М.	13100	100
118	Основы электронного бизнеса	Семиглазов В.А.	351000	82
119	Отечественная история	Кирдяшкин И.В.	071700, 075300, 013100, 060800, 200700, 210100, 220200, 220300, 220507, 350500, 351000, 351400, 200307, 200400, 200800, 201100, 201400, 201500, 220400, 552800, 060400, 061000, 077500, 200100	202
120	Отечественная история -1 (История мировых цивилизаций)	Снегирева А.А.	031000, 031300	100
121	Отладочные средства микропроцессорных систем	Власов А.И.	200400	98
122	Персональные компьютерные программно – аппаратные средства (Эксплуатация и развитие компьютерных сетей и систем) –1	Обрусник В.П.	200307, 200800	55
123	Персональные компьютерные программно – аппаратные средства (Эксплуатация и развитие компьютерных сетей и систем) –2	Обрусник В.П.	200307	79
124	Перспективные технологические процессы в производстве СБИС	Данилина Т.И.	200100	120

125	Планирование на предприятии	Афонасова М.А.	60800	56
126	Политология	Лавровский Н.А.	060400, 060800, 061000, 200100, 200307, 200400, 200700, 200800, 201100, 201400, 201500, 210100, 220200, 220300, 220400, 220500, 220507, 350500, 351000, 351400	94
127	Правоведение	Камышев Э.Н.	031000, 031300, 060400, 060800, 061000, 200100, 200307, 200400, 200700, 200800, 201100, 201400, 201500, 210100, 220200, 220300, 220400, 220500, 220507, 350500, 351000, 351400	100
128	Правовое обеспечение	Кирилова В.В.	60800	100
129	Правовое регулирование хозяйственной деятельности	ТЭПТ	60100	45
130	Приборы и устройства СВЧ –1	Соколова Ж.М., Кущ Г.Г., Шангина Л.И.	200307	218
131	Прикладная механика		220300, 220500, 220507, 200800	100
132	Прикладные системы искусственного интеллекта (искусственный интеллект и экспертные системы)	Зюзьков В.М.	220300, 210100	100
133	Программирование –1	Сафьянова Е.Н.	220400	100
134	Программирование и основы алгоритмизации (программирование) –1	Зюзьков В.М.	210100, 220300, 552800	100
135	Программирование и основы алгоритмизации (программирование) –2	Зюзьков В.М.	210100, 220300, 552800	100
136	Профессионально – этические основы социальной работы	Рябова Г.Б.	350500	97
137	Психология –1 (основы общей психологии)	Рощина Н.А.	031000, 350500	62

138	Психология –2	Дубинина И.А.	350500	100
139	Психология и педагогика	Жуков В.К.	200307, 220500, 060400, 060800, 061000, 200100, 200400, 200700, 200800, 201100, 201400, 201500, 210100, 220200, 220300, 220400, 200507, 351000, 351400	100
140	Радиовещание и радиоаккустика (Радиовещание и связь, Системы радиовещания)	Мелихов С.В.	201100, 200700, 201500, 201400	210
141	Радиоматериалы и радиокомпоненты	Нефедцев Е.В.	201500, 200700, 071700, 075300, 201400	100
142	Разработка и применение пакетов прикладных программ	Миньков С.Л.	351400	80
143	Разработка САПР -1	Губин И.Г.	220300	100
144	Разработка САПР -2	Губин И.Г.	220300	100
145	Регионоведение	Адуева Т.В.	61000	126
146	Система государственного управления	Камышев Э.Н.	61000	182
147	Системное программное обеспечение (объектно – ориентированное программирование)	Катаев М.Ю.	210100	45
148	Системный анализ и методы научно-технического творчества	Озеркин Д.В.	200800	100
149	Системотехника, вычислительные комплексы и сети ЭВС –1	Прищепа Л.С.	220500, 220507	77
150	Системотехника, вычислительные комплексы и сети ЭВС –2	Прищепа Л.С.	220500, 220507	55
151	Социальная экология	Карташев А.Г.	350500	100
152	Социальный менеджмент		61000	63

153	Социология	Рябова Г.Б.	220200, 220300, 031000, 031300, 060400, 060800, 061000, 200100, 200307, 200400, 200700, 200800, 201100, 201400, 201500, 210100, 220400, 220500, 220507, 351000, 351400	99
154	Социология и политология	ТЭПТ	060100, 060200	98
155	Социология управления	Захарова Л.Л.	61000	101
156	Спецглавы математики	Синчинова Л.И.	220200	70
157	Статистика	Лузина Л.И.	351400	100
158	Статистика	Сидоренко М.Г.	060400, 060800, 351000	160
159	Стратегический менеджмент	Крискевич И.И.	351000	101
160	Страхование	Тухватулина Л.А.	60400	95
161	Схемотехника –1 (цифровая схемотехника)	Шарапов А.В.	220400, 220200, 220300, 200307, 201100	89
162	Схемотехника –2 (основы микропроцессорной техники)	Шарапов А.В.	200307, 200400, 220300, 200400	96
163	Схемотехника аналоговых электронных устройств	Красько А.С.	200800, 200700, 201100, 201400, 201500	100
164	Твердотельные приборы и устройства (электроника и микроэлектроника)	Легостаев Н.С.	200307	38
165	Телекоммуникационные системы и компьютерные сети (основы построения телекоммуникационных систем) –1	Пуговкин А.В.	220300, 201100, 210100	122
166	Теоретическая механика	Люкшин Б.А.	210100, 220300	100
167	Теоретические основы конструирования и надежности ЭВС	Илюшкин В.А.	220300, 220507, 220500	100
168	Теоретические основы управления	Рыбалова Е.А.	220200	203

169	Теория автоматического управления	Лебедев Ю.М.	210100, 220200, 200400, 220300	100
170	Теория бухгалтерского учета	ТЭПТ	60100	30
171	Теория организации		61000	100
172	Теория экономических информационных систем	Исакова А.И.	351400	130
173	Техническая электродинамика	Корогодов В.С.	200800	160
174	Техническое обеспечение и внешние устройства ЭВС –1	Прищепа Л.С.	220500, 220507,	100
175	Технологические процессы микроэлектроники	Романовский М.Н.	200800	100
176	Технология и автоматизация производства ЭП и устройств –2	Орликов Л.Н.	200307	100
177	Технология программирования	Буймов Б.А.	60800	100
178	Управление персоналом	Красина Ф.А.	060800, 061000, 351000, 351400	99
179	Управление ресурсами (Финансовый менеджмент)	Красина Ф.А.	60800	100
180	Устройства СВЧ и антенны	Шангина Л.И.	31000, 031300	97
181	Устройства управления бытовой радиоэлектронной аппаратурой	Кормилин В.А.	201400, 201500	100
182	Устройства формирования сигнала (Радиопередающие устройства)	Бордус А.Д., Ильин А.Г.	200700, 201100, 201400, 201500	100
183	Устройчтва СВЧ и антенны	Гошин Г.Г,, Замотринский В.А., Шангина Л.И.	201100, 200700	197
184	Физика –1	Орловская Л.В.	013100, 060800, 071700, 075300, 075500, 200100, 200307, 200400, 200700, 200800, 201100, 201400, 201500, 210100, 220200, 220300, 220400, 220500, 220507, 552800	185

185	Физика –2	Легостаев Н.С.	013100, 071700, 075300, 075500, 200100, 200307, 200400, 200700, 200800, 201100, 201400, 201500, 210100, 220200, 220300, 220400, 220500, 220507, 552800	92
186	Физика -3	Зеленский В.И.	071700, 075300, 075500, 200100, 200307, 200400, 200700, 200800, 201100, 201400, 201500, 210100, 220200, 220300, 220400, 220500, 220507, 552800	119
187	Физика твердого тела –1	Смирнов С.В.	200100	100
188	Физика твердого тела –2	Смирнов С.В.	200100	99
189	Физические методы исследования объектов окружающей среды	Смирнов Г.В.	13100	99
190	Физические основы микроэлектроники	Несмелов Н.С.	220500, 220507	100
191	Физические основы электронной техники –2	Давыдов В.Н.	200307	100
192	Философия	Московченко А.Д.	013100, 060400, 060800, 061000, 071700, 075300, 075500, 200100, 200307, 200400, 200700, 200800, 201100, 201400, 201500, 210100, 220200, 220300, 220400, 220500, 220507, 350500, 351000, 351400, 552800	100
193	Финансовый менеджмент	Красина Ф.А.	60400	100
194	Финансы	Черская Р.В.	60400	37
195	Финансы и кредит	Крискевич Е.И.	351000	100
196	Финансы и кредит	Терентьева М.И.	60100	92

197	Финансы предприятий	Мезенцева Н.С.	60400	100
198	Финансы предприятия	Крискевич С.Н., Крискевич Е.И.	351000	100
199	Финансы, денежное обращение и кредит	Кудряшова Л.В.	61000	421
200	Финансы, денежное обращение и кредит	Мезенцева Н.С.	060800, 351400	99
201	Функциональное и логическое программирование	Зюзьков В.М.	220400	101
202	Химия	Якунина Г.М.	060800, 200100, 200307, 200400, 200800, 201100, 220200, 220500, 220507, 552800, 013100	100
203	Химия –2	Смирнов Г.В., Якунина Г.М.	13100	100
204	Цифровые устройства и микропроцессоры –2	Кормилин В.А.	200700, 201500	110
205	ЭВМ и вычислительные системы	Обрусник В.П.	210100	100
206	ЭВМ и вычислительные системы	Одиноков В.В.	210100	102
207	ЭВМ и периферийные устройства –1	Прищепа Л.С.	220300	110
208	ЭВМ и периферийные устройства –2	Губин И.Г.	220300	100
209	ЭВМ и периферийные устройства –3	Губин И.Г.	220300	100
210	Экология	Смирнов Г.В.	060800, 061000, 071700, 075300, 075500, 200100, 200307, 200400, 200700, 200800, 201100, 201400, 201500, 210100, 220200, 220300, 220400, 220500, 220507, 351400	138
211	Эконометрика	Лузина Л.И.	060400, 220400, 351400	100

212	Экономика	Алферова Л.А., Мезенцева Н.С.	031000, 031300, 200100, 200307, 200400, 200700, 200800, 201100, 201400, 201500, 210100, 220200, 220300, 220400, 220500, 220507, 350500, 351000	94
213	Экономика и социология труда	Силютина В.А.	060800, 351400	100
214	Экономика недвижимости		60800	100
215	Экономика отрасли	Глотова И.А.	60800	150
216	Экономика предприятия		060400, 060800	100
217	Экономико – математические методы (мат. Модели в экономике)	Сидоренко М.Г.	060400, 0608006 351000, 351400	100
218	Экономико – правовые основы рынка программных продуктов	Минькова Н.П.	351400, 220400	75
219	Экономическая безопасность	Донцов Г.Ю.	351000	60
220	Экономическая теория		351400, 060400, 060800, 061000	98
221	Электродинамика и распространение радиоволн	Соколова Ж.М., Мандель А.Е.	071700, 200700, 075300, 201100, 201400, 201500	128
222	Электродинамика и техника СВЧ	СоколоваЖ.М., Шангина Л.И.	200307	100
223	Электроника –1	Денисов Н.П.	210100	100
224	Электроника –1 (Оптоэлектронные средства передачи и обработки информации, Вакуумные и плазменные приборы и устройства)	Злобина А.Ф.	200307, 200400	100
225	Электроника –2	Денисов Н.П.	210100	100
226	Электроника и микроэлектроника –1 (Физические основы микроэлектроники)	Ицкович В.М.	200800, 071700, 075300, 200700, 201400, 201500, 201100	81

227	Электроника и микроэлектроника –2 (Физические основы микроэлектроники)	Ицкович В.М.	200800, 200700, 071700, 075300, 201400 , 201500, 201100	127
228	Электроника и схемотехника	Денисов Н.П.	75500	100
229	Электронные приборы СВЧ и квантовые приборы	Шангина Л.И.	200700, 201100	130
230	Электропитание аудиовизуальной аппаратуры (Электропитание и элементы электромеханики)	Зайченко Т.Н.	201400, 201500	100
231	Электротехника и электроника –1 (теоретические основы электротехники)	Кобрина Н.В., Фикс Н.П.	210100, 220300	100
232	Электротехника и электроника –1 (теоретические основы электротехники)	Кобрина Н.В., Фикс Н.П.	210100, 220300	100
233	Элементы и устройства систем управления	Гарганеев А.Г.	210100	100
234	Этика и культура управления	Невраева И.В.	61000	108
235	Языки программирования (ООП)	Катаев М.Ю.	75500	45
			итого	25361

Оглавление