№ 46 (335) 8–15 декабря 2001

ИНФОРМАТИКА

Еженедельная газета Объединения педагогических изданий «ПЕРВОЕ СЕНТЯБРЯ»
ПОДПИСКА: (095) 249-47-58

XI международная конференция-выставка "Информационные технологии в образовании" ("ИТО-2001")

С 5 по 9 ноября 2001 года в Московском городском лицее № 1511 под патронатом Федерации Интернет Образования прошла XI международная конференция-выставка "Информационные технологии в образовании" ("ИТО-2001")

Информация подготовлена по материалам пресс-релиза, распространенного организаторами конференции после ее завершения

Очередная конференция "ИТО" была проведена при поддержке Министерства образования РФ, Фонда поддержки российского учительства, Института ЮНЕСКО по информационным технологиям в образовании, Института проблем информатики Российской академии наук, Московского комитета образования, Финансовой академии при Правительстве РФ, Московского государственного инженерно-физического института, Московского центра Интернет-образования и научно-производственного предприятия "БИТ про".

Генеральным спонсором выступила компания "Формоза". Спонсорами — Intel, "Хронобус", "1С", TrueSystem, Step Logic, "Инис-Софт". Информационную поддержку оказали — издательство Hard'n'Soft; журналы: "Информатика и образование", "Лидеры образования", "Мир Школы", "Семья и школа", "Учитель года", "Элитное образование", "Юный техник"; газеты: "Информатика", "Учительская газета"; CNews.ru. Технические спонсоры — "Кирилл и Мефодий", "Фантазия". Проекционное оборудование предоставлено компаниями "Смистар" и Activision.

Для участия в конференции зарегистрировались более 860 человек. Из них приехали на конференцию для очного участия 500 человек из 140 населенных пунктов России, Украины, Казахстана, Беларуси, США, Германии.

Подробная информация о конференции — тезисы представленных докладов, списки участников, расписание всех мероприятий и т.д. — опубликована в Интернете по адресу: **http://ito.edu.ru**. По проблемам, затронутым на конференции, открыт форум на сайте Федерации Интернет Образования по адресу: **http://www.fio.ru/ito-forum**.

В ходе конференции были проанкетированы около тысячи специалистов в области информационных технологий. На основе их ответов были определены лучшие продукты в области образования в 16-ти номинациях.

На с. 2—3 приводятся данные номинации и процентное распределение голосов посетителей между ними без учета ответов "не знаю".

Продолжение читайте на с. 2


Министерство образования Российской Федерации
Фонд поддержки российского учительства
Институт ЮНЕСКО по информационным технологиям в образовании
Институт проблем информатики Российской Академии наук
Московский комитет образования
Научно-производственное предприятие "БИТ про"

Диплом

XI Международной конференции-выставки "Информационные технологии в образовании"
в Интернете Федерация Интернет Образования

НАГРАЖДАЕТСЯ УЧАСТНИК ВЫСТАВКИ

Объединение педагогических изданий "Первое сентября"

за лучший образовательный WEB-сайт

http://www.1september.ru

по результатам опроса участников и гостей конференции

Директор выставки Федерации С.А. Филиппов

Председатель конкурсной комиссии С.В. Мизгина

Председатель оргкомитета конференции С.Г. Григорьева

Москва, 5-9 ноября 2001 г.

Лучшее электронное издание по математике

	ПРОДУКТ	ФИРМА
23%	**Виртуальная школа КМ. Уроки геометрии. Уроки алгебры**	**Кирилл и Мефодий**
16%	Живая геометрия 3.1	ИНТ
16%	Курс математики Боревского	МедиаХауз
16%	Открытая математика 1.0. Планиметрия	Физикон
13%	Математика 7—11	КУДИЦ
8%	Математика. Начальная школа	Инис-Софт
8%	TeachPro Математика (7—11-е классы)	ММТ и ДО
	Лучшее электронное издание по естественно-научным дисциплинам	
22%	**1С:Репетитор. Естественные науки**	**1С**
18%	Живая физика	ИНТ
17%	Открытая физика	Физикон
10%	Курс физики Боревского	МедиаХауз
10%	Виртуальная школа КМ. Уроки физики. Уроки химии. Уроки биологии	Кирилл и Мефодий
9%	Шерлок Холмс. Дело о радиации	ИБРАЭ РАН
6%	TeachPro Физика (7—11-е классы)	ММТ и ДО
6%	Redshift 4	Новый Диск
2%	Органическая химия	МарГТУ
	Лучшее электронное издание по гуманитарным дисциплинам	
37%	**История России: XX век**	**Клио Софт**
19%	Атлас древнего мира	Новый Диск
15%	Виртуальная школа КМ. Репетитор по истории	Кирилл и Мефодий
14%	История древнего мира. Загадки Сфинкса	МедиаХауз
10%	Основы моделирования	Инис-Софт
5%	География 8-й класс. Природа и население	РМЦ
	Лучшее электронное издание по английскому языку	
28%	**Профессор Хиггинс. Английский без акцента!**	**ИстраСофт**
20%	ABBYY Lingvo 7.0	ABBYY Software House
15%	Английский на каждый день	Кирилл и Мефодий
14%	English Gold 2000	ММТ и ДО
14%	REWARD InterN@tive	Новый Диск
9%	Английский. Быстрый старт	МедиаХауз
	Лучшее электронное издание по русскому языку	
34%	**1С:Репетитор. Тесты по пунктуации**	**1С**
24%	Грамотей	Эрикос
15%	Виртуальная школа КМ. Уроки русского языка	Кирилл и Мефодий
10%	Русский язык	МедиаХауз
10%	Фраза	Новый Диск
7%	Русский язык. Средняя школа. Часть 1	Инис-Софт
	Лучшее электронное издание по использованию офисных продуктов	
29%	**Самоучители: Word 2000, Excel 2000, Windows 98**	**Новый Диск**
23%	MS Office 2000 Professional	Общество "ЗНАНИЕ" России
20%	1С:Учебное предприятие	1С
12%	Практический курс. Изучаем Windows 2000	Кирилл и Мефодий
9%	Основы работы в Windows	Инис-Софт
7%	TeachPro PC User98	ММТ и ДО
	Лучшая среда для конструирования уроков, система тестирования	
14%	**Батисфера**	**ИНФОРМПРОЕКТ**
14%	**Открытая образовательная платформа "Формоза"**	**Формоза**
13,5%	Грамотей	Эрикос
13,5%	ЛогоМиры	ИНТ
10%	ABBYY FormReader 4	ABBYY Software House
10%	Активный Экран	TDS Promethean Deutschland GmbH
10%	Виртуальная школа КМ	Кирилл и Мефодий
8%	Генератор тестов и уроков	Инис-Софт
7%	ПоЗнание 2.0	Общество "ЗНАНИЕ" России
	Лучшая система дистанционного образования	
19%	**Открытый Колледж**	**Физикон**
18%	Соло на клавиатуре 7.0	Шахиджанян В.В., ABBYY Software House
15%	1С:Репетитор-он-Лайн	1С

13%	Виртуальная школа КМ	Кирилл и Мефодий
12%	СДО "Прометей"	Институт виртуальных технологий в образовании
12%	REWARD InterN@tive	Новый Диск
6%	"Семейный наставник" и "Школьный наставник"	Инис-Софт
5%	http://education.kudits.ru/homeandschool	КУДИЦ, МИПКРО
	Лучшая развивающая программа для детей до 10 лет	
29%	**Детская мастерская**	**Кирилл и Мефодий**
25%	ПервоЛого	ИНТ
14%	Русский язык. Начальная школа	Инис-Софт
13%	Математика на планете счетоводов	МедиаХауз
12%	Правильный английский без скучных правил	Новый Диск
7%	Остров арифметики	ИстраСофт
	Лучшая электронная энциклопедия	
80%	**Большая энциклопедия Кирилла и Мефодия-2001**	**Кирилл и Мефодий**
14%	Энциклопедия истории России	Новый Диск
6%	Кругосвет	Россия-он-Лайн
	Лучшая мультимедиа-книга	
41%	**Библейские сюжеты в коллекции Эрмитажа**	**Инфостудия ЭКОН**
31%	Храм Христа Спасителя	Кирилл и Мефодий
17%	Русские народные промыслы (береста, лаковая миниатюра)	ИБРАЭ РАН
11%	English Reading Club	Новый Диск
	Лучшая программа автоматизации делопроизводства учебного заведения	
39%	**ХроноГраф 2.0**	**Хронобус**
24%	База данных школьного администратора	Кирилл и Мефодий
20%	Система автоматизации "БЭСТ"	Интеллект-Сервис
9%	ПараГраф: Учебное заведение XXI	Инис-Софт
8%	АВЕРС-Бухгалтерия	Аверс
	Лучшее ПО общего назначения для использования в учебном заведении	
28%	**Медиатека Кирилла и Мефодия**	**Кирилл и Мефодий**
27%	ABBYY FINE READER 5.0	ABBYY Software House
14%	Батисфера	ИНФОРМПРОЕКТ
10%	"Школьная система автоматизированного проектирования" профессора А.А. Богуславского	Аскон-М
10%	Учебный мониторинг	Инис-Софт
6%	Дизайнер курсов	Институт виртуальных технологий в образовании
5%	Как решить проблему	Новый Диск
	Лучший учебник по информатике	
40%	**Учебно-методический комплект Н.Д. Угриновича "Информатика и информационные технологии"**	**Лаборатория базовых знаний**
29%	Комплект учебников по информатике для средней школы, расширенное и дополненное издание, под редакцией проф. Н.В. Макаровой	Питер
19%	Макарова Н.В. и др. "Информатика", учебник и практикум	Финансы и статистика
8%	Основы алгоритмизации (учебные пособия + CD-ROM)	Инис-Софт
4%	Семенов, Рудченко, Щеглова. "Информатика 1—4"	ИНТ
	Лучший образовательный web-сайт	
22%	**http://www.1september.ru**	**Объединение педагогических изданий "Первое сентября"**
18%	http://vschool.ru	Кирилл и Мефодий
11%	http://www.school.edu.ru	Портал российского образования
11%	Сайт "КОМПАС в образовании"	Аскон-М
10%	http://repetitor.1c.ru/online	1С
10%	http://www.college.ru	Физикон
9%	http://lingvo.yandex.ru/	Совместный проект компаний Yandex и ABBYY Software House
5%	http://education.kudits.ru/homeandschool	КУДИЦ
4%	http://www.toefl.ru	Новый Диск
	Лучшее аппаратное обеспечение образовательного процесса	
30%	**Проектор NEC VT 45**	**Activision**
30%	**Активный Экран**	**TDS Promethean Deutschland GmbH**
28%	Компьютеры Teen	Формоза
11%	WinClass	TrueSystem

ИТО-2001: традиции и новинки выставки

Д.Ю. Усенков,
Москва

Традиционная выставка аппаратных и программных средств учебного назначения, сопровождающая ежегодную конференцию "Информационные технологии в образовании", довольно консервативна: и по составу участников, и по представляемым продуктам. В этом кратком обзоре мы представляем ряд новых интересных экспонатов.

Выступает А.В. Могилев, г. Воронеж.

Junior — Linux для неспециалиста

О преимуществах операционных систем Linux уже сказано и написано немало. Это и существенно большая по сравнению с Microsoft Windows (по крайней мере прежних версий 95/98) надежность в работе, и изначальная ориентация на работу в сетях, и, что немаловажно, открытость этих систем (что для пользователей означает возможность получить дистрибутив практически бесплатно). Однако до недавнего времени Unix-системы оставались прерогативой специалистов: прежде всего из-за отсутствия простых в использовании файловых и оконных оболочек ("традиционный" Unix-интерфейс строится на текстовых сообщениях и командах аналогично MS-DOS), из-за недостаточно широкого набора прикладного программного обеспечения и отчасти из-за сложностей при установке и конфигурировании.

Версия ALT Linux Junior (версия 1.1), представленная на выставке компанией ALT Linux, ориентирована на широкий круг пользователей. Для них фирма предлагает простой в установке и в освоении дистрибутив на одном CD-диске, специально сконфигурированный для использования на домашних, офисных и школьных компьютерах. Процесс установки ОС, по заверению представителей фирмы, не сложнее такового при инсталляции Windows, а поддержка широкого спектра периферийного оборудования облегчает автоматическое конфигурирование системы (перечень моделей настольных и переносных компьютеров, на которых проводилось тестирование Junior, в предоставленных фирмой материалах не приводится, но, по словам ее представителей, достаточно широк). В результате сразу после установки пользователь получает полностью настроенную и готовую к работе систему с солидным набором прикладных программ, для использования которых практически не требуется каких-либо специальных знаний системного администратора.

В комплект Junior входят, в частности, офисная система OpenOffice, поддерживающая форматы файлов Microsoft Office и работающая с кириллическими шрифтами, средства воспроизведения мультимедиа (драйверы DVD, VideoCD, плейеры форматов MPEG4, MP3 и пр.), браузеры и почтовые клиенты, ICQ и даже набор различных игр. Требования же к компьютеру сравнимы с предъявляемыми Windows 98: процессор Pentium и выше, ОЗУ 32 Мб (желательно 64 Мб). Стоимость диска с дистрибутивом составляет порядка 100—130 руб., причем это оплата главным образом за комплектацию дистрибутива и изготовление диска — критерий бесплатности самого программного обеспечения по-прежнему остается в силе и здесь, так что при наличии высокоскоростного доступа в Интернет отдельные "составные части" этого дистрибутива можно будет свободно скачивать по своему выбору (представители фирмы обещают сделать доступными для всех желающих даже их исходные тексты).

Более подробную информацию о Junior и других продуктах фирмы можно найти на сайте по адресу: **http://www.altlinux.ru**. Здесь же при желании вы сможете найти информацию о том, где приобрести дистрибутив или скачать желаемые файлы.

Конференции "ИТО" стали постоянным местом встречи специалистов из различных регионов России. "Круглый стол" проводит Е.К. Хеннер, г. Пермь.

"Живое видео" в Интернете — своими руками

Сегодня все более популярным становится трансляция в реальном времени видеоизображений через Интернет: пользователям предоставляется возможность увидеть, что происходит в том или ином месте, на которое нацелена видеокамера, — будь то улица другого города, зал музея или аудитория, в которой проходит какое-либо мероприятие. Последнее особенно привлекает внимание в свете современных перспектив дистанционного и открытого образования — видеотрансляция через Интернет представляет собой одну из наиболее удобных технологий организации видеоконференций, открытых уроков (как лекционных занятий, так и семинарских), демонстрации хода экспериментов и дистанционного мониторинга и пр., а также для проведения "виртуальных экскурсий" по достопримечательностям других городов и стран.

Однако создать собственную web-страницу с "живым видео" достаточно сложно. Прежде всего необходимо иметь собственный сервер и все необходимое программное обеспечение. А видеокамера подключается к этому серверу (для чего также требуется специальное ПО и в некоторых случаях — дополнительное периферийное оборудование). Но собственный сервер с постоянным доступом в сеть — дело дорогое и не по карману большинству образовательных учреждений в нашей стране. Как быть?

Специалисты компании HyperMethod, известной многим пользователям благодаря одноименной инструментальной системе, предложили интересное и достаточно простое решение проблемы. Представленная ими программа LiveCam Pro (версия 2.0; демо-версия доступна на сайте **http://www.hypermethod.ru**) позволяет организовать Интернет-вещание с видеокамеры, подключенной к вашему локальному компьютеру (не серверу!), тогда как соответствующая web-страница может быть размещена на любом сервере, вплоть до общедоступных бесплатных типа **chat.ru** или **narod.ru**. На web-странице необходимо разместить лишь один Java-апплет (сгенерированный программой LiveCam Pro), а изображение с вашей видеокамеры в виде последовательности отдельных кадров — "слайдов" будет фактически поступать с вашего локального компьютера и далее транслироваться всем посетителям данной странички как бы "сквозь сервер". При этом даже не обязательно иметь постоянное подключение к Интернету: вы можете установить "график вещания", а программа LiveCam Pro автоматически, в соответствии с заданным графиком, будет выполнять дозвон к вашему провайдеру и производить передачу видеосигнала через модем. Кроме того, программа LiveCam Pro позволяет аналогичным способом транслировать и текущее изображение на экране дисплея, что позволяет при работе уже в локальной сети осуществлять демонстрацию работы преподавателя с каким-либо программным продуктом всему классу.

Вообще же LiveCam Pro — это нечто гораздо большее, чем просто отдельный программный пакет. Это не что иное, как способ (или, во всяком случае, хотя бы идея) реализации некоторых возможностей технологии "клиент — сервер" без необходимости размещения на сервере соответствующих программных модулей (CGI, ISAPI, Perl и пр.). Можно предположить, что в ближайшее время появятся разработки, позволяющие аналогично организовать на web-странице регистрацию пользователей, интерактивные тесты, голосования, работу с базами данных, почтовой рассылкой и пр. с размещением необходимого программного обеспечения на локальном компьютере, а это, в свою очередь, позволило бы существенно расширить спектр возможностей, доступных авторам web-сайтов, размещенных на бесплатных серверах.

Открытый урок проводит один из авторов популярных учебников по информатике И.Г. Семакин.

"Физикон": пополнение прибыло

Такие программные продукты, как "Открытая физика" или "Открытая математика", созданные компанией "Физикон", хорошо известны многим читателям. Их "изюминка" — это входящие в состав учебных курсов интерактивные модели (на базе Java-апплетов и Flash-анимаций), демонстрирующие те или иные явления и процессы. При этом пользователь может изменять значения целого ряда их параметров, проводя таким образом "виртуальные эксперименты", а в ряде случаев и самостоятельно конструировать из имеющихся типовых элементов произвольные

У Н.В. Макаровой, как всегда, немало и сторонников, и оппонентов.

схемы и затем исследовать их работу.

На выставке "ИТО-2001" компания "Физикон" представила новые учебные курсы из этого семейства: "Открытая астрономия" и "Открытая химия" (их демоверсии можно найти на сайте **http://www.physicon.ru**).

Курс "Открытая астрономия" предназначен для изучения таких тем, как "Звездное небо", "Свет и вещество", "Небесная механика", "Солнечная система", "Солнце", "Звезды", "Галактики" и "Вселенная". Он содержит в себе иллюстрированный теоретический материал (750 фотографий, графиков, карт и схем), справочные таблицы, задачи и контрольные вопросы для учащихся, а самое главное — почти 60 различных интерактивных моделей и целый интерактивный Планетарий. С их помощью учащиеся могут пронаблюдать солнечное затмение и восходы на Меркурии, осмотреть со всех сторон нашу Галактику и даже совершить экскурсию в прошлое, например, наблюдать формирование Луны. Планетарий позволяет увидеть карту звездного неба для любой точки планеты в любое время и содержит, кроме планет Солнечной системы, более 9000 звезд, 100 малых планет и более 200 комет, туманностей и галактик. Дополнительно в комплекте имеются методическое руководство для учителя и комплект карт звездного неба для средней полосы России.

Курс "Открытая химия" содержит в себе, помимо теоретического материала и иллюстраций, более 100 схем и графиков, 58 интерактивных моделей для наиболее интересных химических явлений и законов, позволяющих учащемуся увидеть схемы строения молекул и образования различных типов химических связей, понять механизмы протекания химических реакций и провести "виртуальные эксперименты". Интерактивная таблица Менделеева, также имеющаяся в составе учебного курса, содержит информацию обо всех химических элементах и примерах их использования, а трехмерный визуализатор химических формул позволяет наблюдать структуру более 600 различных органических молекул, произвольно вращая их и рассматривая со всех сторон.

Огромный интерес вызвал "круглый стол", посвященный вопросам, связанным с введением Единого государственного экзамена. Его проводил один из ведущих российских специалистов в области тестирования А.Г. Шмелев.

Доклады, вызвавшие наибольший интерес у участников конференции, были отмечены дипломами. Один из дипломов вручен хорошему другу дедушки Фёрстова Ю.А. Первину.

И наконец, напомним читателям о проекте "Открытый колледж" (**http://www.college.ru**), разрабатываемом компанией "Физикон" и направленном на интеграцию выпускаемых на компакт-дисках учебных курсов с индивидуальным обучением через Интернет. Система дистанционного обучения "Открытый колледж" предоставляет преподавателю системы администрирования, генерации тестовых заданий, мониторинга результатов учебной деятельности, позволяющие реализовать требуемые online- и offline-режимы обучения. Сюда также входят более 3000 готовых тестов по математике, физике, химии, биологии и экономике, а также обзоры различных Интернет-ресурсов по предметам школьной программы, методические материалы для преподавателей и форум для обмена опытом использования в школе различных компьютерных программ и даже сетевая учебная игра для школьников, где тесты для самопроверки формируются индивидуально в зависимости от запросов ученика, ведется постоянный мониторинг их успехов и определяется десятка лучших учащихся. Кстати, для пользователей, не имеющих выхода в Интернет, выпущена специальная версия "Открытого колледжа" на CD-диске.

"Юный техник" — online!

Хорошо знакомый многим журнал "Юный техник" с началом нового тысячелетия наконец-то обрел новое лицо в сети Интернет. Кроме выпускаемых этим издательством "бумажных" журналов ("ЮТ", "Левша" и предназначенного главным образом для младших школьников "А почему"), теперь вниманию всех желающих предлагается ежеквартальный сетевой дайджест "Спутник ЮТ", публикуемый на сайте издательства по адресу: **http://junetech.chat.ru**. В каждом его выпуске читатели найдут ряд интересных статей, уже опубликованных в журналах "ЮТ", "Левша" и "А почему", а также некоторые еще не опубликованные материалы, дополняющие эти статьи. Среди них — обзоры новинок техники и новейших технологий (например, достижений термоядерного синтеза или генной инженерии), рассказы о различных природных явлениях и экспериментах, которые можно провести на школьном уроке физики, советы умельцам, а также ответы на некоторые детские "почему?", которые, впрочем, нередко оказываются интересными и взрослым. Любители же бумажного моделирования смогут переписать готовые листы выкроек, чтобы распечатать их на цветном принтере. И наконец, те, кто увлекается фантастикой, найдут в каждом выпуске "Спутника ЮТ" хороший фантастический рассказ (или даже повесть), а также и более серьезные статьи о горизонтах неизведанного.

К сожалению, рассказать обо всех интересных экспонатах в одном кратком обзоре совершенно нереально. Так что снова порекомендуем читателям самим посещать конференции-выставки "Информационные технологии в образовании" (что можно сделать совершенно бесплатно). Поверьте, на это не жалко потратить время!

Олимпиады по информатике. Пути к вершине

Лекции читает Е.В. Андреева

Лекция 5. Олимпиадные задачи и сортировка

Данную тему, так же, как и прошлую, следует рассматривать с двух сторон. Во-первых, при решении различных олимпиадных задач данные довольно часто требуется упорядочить по некоторому признаку (то есть отсортировать). При этом, если специально не оговорено иное, мы будем считать, что массив требуется отсортировать в порядке неубывания значений его элементов (для различных элементов — в порядке возрастания). Во-вторых, задача сама по себе может требовать построения оптимального в смысле определенных требований или нестандартного алгоритма сортировки. Помимо этого, изюминка олимпиадной задачи может состоять в формализации критерия, по которому следует сортировать данные.

Использование "стандартных" алгоритмов сортировки одномерных массивов

Во всех примерах, относящихся к одномерным массивам, будем использовать следующий тип данных:

```
type list=array[1..N] of <скалярный тип>;
```

Конкретный тип элемента в большинстве описываемых алгоритмов может быть как любым числовым, так и символьным или даже строковым. Последний тип скалярным не является, но к элементам этого типа применимы все операции сравнения, с помощью которых обычно (но не всегда !!!) и производится сортировка.

Пусть в задаче данные необходимо отсортировать один раз. Тогда выбирать следует тот из известных вам алгоритмов сортировки, на реализацию которого лично у вас уйдет меньше всего времени и отлаживать который тоже не придется. В таком случае ребята обычно реализуют так называемую "пузырьковую" сортировку, или сортировку прямым выбором. Конечно, размерность массива при этом должна быть не намного более 1000 элементов, в противном случае время работы программы может оказаться неоправданно большим, так как количество только операций сравнения у обоих упомянутых алгоритмов равно $N(N-1)/2$.

План публикаций лекций курса "Олимпиады по информатике. Пути к вершине" на "Страницах повышения квалификации".

Номер лекции	Номер газеты
1	38/2001
2	40/2001
3	42/2001
4	44/2001
5	46/2001
6	48/2001
7	6/2002
8	8/2002
9	10/2002
10	12/2002
11	14/2002
12	16/2002

Уважаемые читатели, пожалуйста, обратите внимание на то, что лекции 7—12 будут опубликованы в первом полугодии 2002 г.

Если же размерность массива велика или сортировку требуется проводить на каждом шаге некоторого цикла, то проще всего использовать алгоритм так называемой "быстрой" сортировки [1—4]. Этот алгоритм на практике выполняется за не более $N\log_2 N$ операций сравнения и зачастую еще меньшее число операций присваивания, хотя теоретическая оценка для худшего случая является квадратичной по N. В языке программирования Си он реализован в виде функции `qsort`, входящей в библиотеку `stdlib`. В полной поставке среды программирования Турбо Паскаль данный алгоритм можно найти в файле EXAMPLES\DOS\QSORT.PAS. Нередко приведенный в указанном файле текст процедуры сортировки не требуется изменять вообще, но иногда незначительные переделки необходимы, поэтому полезно разобраться в особенностях данной реализации алгоритма сортировки. Рассмотрим возможную модификацию этой процедуры, выполненную для задачи *Кировской открытой командной олимпиады по программированию 2001 г.* "Предусмотрительное жюри".

> В командных соревнованиях по программированию временем решения задачи считается время в минутах от начала тура до момента посылки правильного решения этой задачи на проверку. Жюри старается расположить задачи в таком порядке, чтобы для команды, решившей по порядку все задачи, общее время решения было максимальным, если ориентировочное время, которое требуется потратить на решение каждой из задач, известно. В самом деле, если на тур предлагаются две задачи, и на решение первой команда тратит 10 минут, а на решение второй — 20, то время решения будет равно 40 минутам (первую задачу команда сдает на 10-й минуте тура, вторую — на 30-й, $10 + 30 = 40$). Если же задачи расположить в обратном порядке, то время будет равно 50 (задачи будут сданы на 20-й и 30-й минутах). Помогите членам жюри расположить задачи в желаемом порядке.

Очевидно, что задачи следует предлагать в порядке невозрастания ожидаемых временных затрат на каждую из них. То есть требуется произвести сортировку временных характеристик имеющихся задач и переставить согласно полученному порядку исходные номера задач. Проще всего делать это одновременно, незначительно модифицируя стандартную процедуру (изменения, внесенные в реализацию алгоритма сортировки, выделены серым цветом):

```
const nn = …{максимальное количество задач};
type rr = record t, k : integer end;
     list = array[1..nn] of rr;
var time : list;
    i, n : integer;
procedure QuickSort(var a : List;
                        Lo, Hi : integer);
```

```
procedure Sort(l, r : integer);
var
  i, j, x : integer;
  y : rr;
begin
  i := l; j := r; x := a[(l + r) div 2].t;
  repeat
    while a[i].t > x do i := i + 1;
    while x > a[j].t do j := j - 1;
    if i <= j then
    begin
      y := a[i]; a[i] := a[j]; a[j] := y;
      i := i + 1; j := j - 1
    end
  until i > j;
  if l < j then Sort(l, j);
  if i < r then Sort(i, r)
end;
begin {QuickSort};
  Sort(Lo, Hi)
end;
begin
{решение задачи с помощью модифицированной
 процедуры QuickSort}
  read(n);
  for i := 1 to n do
  begin
    read(time[i].t);
    time[i].k := i
  end;
  QuickSort(time,1,n);
{сортировали по временам, а печатаем индексы}
  for i := 1 to n do
    write(time[i].k:8)
end.
```

В заключение данного раздела приведем таблицу, в которой для известных универсальных алгоритмов сортировки приведены порядки для количества выполняемых тем или иным алгоритмом в худшем случае операций сравнения и присваивания.

Таким образом, наилучшую теоретическую оценку имеют два последних из перечисленных в таблице алгоритмов, однако в практическом программировании для сортировки данных обычно используется QuickSort, как в силу высокой производительности (особенно выигрывает данный алгоритм по числу реально выполняемых присваиваний), так и в силу простой реализации. Тем не менее в олимпиадных задачах данные могут быть представлены так, что отсортировать за отведенное время их можно будет лишь с помощью пирамидальной сортировки. Ее еще называют сортировкой кучей или деревом [1—5]. Особенно полезным при решении некоторых задач, даже напрямую с сортировкой не связанных, может оказаться понимание и умение реализовать операцию "проталкивания" элемента по упорядоченному дереву, с помощью которой и осуществляется данная сортировка. Дополнительную информацию о различных алгоритмах сортировки можно получить в [6].

Название сортировки	Количество сравнений	Количество присваиваний
Простой обмен (пузырьковая)	$O(N^2)$	$O(N^2)$
Прямой выбор	$O(N^2)$	$O(N)$
Простая вставка	$O(N^2)$	$O(N^2)$
Бинарная (двоичная) вставка	$O(N\log N)$	$O(N^2)$
Быстрая (*QuickSort*)	$O(N^2)$ (на практике $O(N\log N)$)	$O(N^2)$ (на практике $O(N\log N)$)
Слияниями	$O(N\log N)$	$O(N\log N)$
Пирамидальная (*HeapSort*)	$O(N\log N)$	$O(N\log N)$

Оптимальная сортировка

Пусть нам требуется построить алгоритм сортировки, оптимальный по количеству операций присваивания или сравнения для худшего случая. Из приведенной таблицы следует, что алгоритм прямого выбора является линейным по количеству выполняемых операций присваивания (их количество для худшего случая равно $3(N - 1)$) и, следовательно, асимптотически оптимален. Более того, при использовании дополнительного массива для записи результата этот алгоритм можно реализовать ровно за N перемещений элементов, что является точной нижней оценкой для количества присваиваний в худшем случае, а именно — если все N элементов первоначально находились не на своих местах, мы не можем менее чем за N перемещений расставить их в нужном порядке. Данное свойство алгоритма прямого выбора можно использовать и в практическом программировании, когда при сортировке некоторых объектов операция присваивания (перемещения) оказывается значительно более трудоемкой, чем операция сравнения. Например, при сортировке столбцов двухмерного массива по значениям первых элементов столбцов, или при сортировке записей по значению одного из полей, или при сортировке настоящих контейнеров, которые требуется переставить с помощью подъемного крана по возрастанию стоимости находящихся в них грузов. Справедливости ради заметим, что возникающую в последнем случае техническую проблему при компьютерной сортировке можно решать и по-другому: сортировать не сами объекты, а указатели на них.

На олимпиаде задача построения оптимального по количеству перемещений алгоритма сортировки может быть поставлена и иначе. Например, попробуйте найти оптимальное решение задачи "Парковка", предлагавшейся на международной олимпиаде по информатике в 2000 году (см. [7]). Учтите, что для каждой из конкретных входных неупорядоченных последовательностей оно может быть своим. Определите, для любых ли значений N из условия задачи это можно сделать в общем случае. Универсальное решение этой задачи, рассмотренное в [7], является эвристическим и не всегда минимально, хотя и удовлетворяет условию.

Перейдем теперь к рассмотрению алгоритма сортировки, оптимального по количеству сравнений элементов между собой. Если изначально нам не известны никакие отношения между элементами, то для N элементов результатом их сортировки может оказаться любая из $N!$ перестановок элементов. Тогда из теории информации следует, что нижняя граница числа сравнений, необхо-

димых в худшем случае для определения нужной нам перестановки, равна $\lceil \log_2 N! \rceil$. При больших N это число растет так же, как и $N\lceil \log_2 N \rceil$ — теоретическая оценка сверху для ряда универсальных алгоритмов (см. таблицу выше). Например, алгоритм вставки на первом шаге сравнивает элементы a_1 и a_2, на втором — в упорядоченную последовательность из двух элементов вставляется элемент a_3 и т.д. Если поиск места для вставки очередного элемента в уже упорядоченную последовательность сделать бинарным, то число сравнений в худшем случае составит $\lceil \log_2 2 \rceil + \lceil \log_2 3 \rceil + \ldots + \lceil \log_2 N \rceil$ и будет отличаться от $\lceil \log_2 N! \rceil$ менее чем на N. Составим таблицу, в которой для небольших N приведены сравнительные значения результатов вычисления по трем формулам:

N	2	3	4	5	6	7	8	9	10	11	12
$\lceil \log 2 N! \rceil$	1	3	5	7	10	13	16	19	22	26	29
Бин. вставка	1	3	5	8	11	14	17	21	25	29	33
$N\lceil \log 2 N \rceil$	2	6	8	15	18	21	24	36	40	44	48

Несложно показать, что и другие универсальные алгоритмы сортировки, имеющие ту же асимптотическую оценку на количество сравнений, не совпадают по количеству сравнений с точной нижней оценкой. В задачниках по программированию встречается упражнение, требующее отсортировать 5 произвольных элементов не более чем за 7 сравнений (убедитесь, что любой универсальный алгоритм сделает это, в худшем случае используя 8 сравнений, и попробуйте самостоятельно построить алгоритм для решения этой задачи, так как это действительно возможно сделать за 7 сравнений).

В [8] показано, как для известного фиксированного значения N за абсолютно минимальное число сравнений отсортировать конкретную входную последовательность из N элементов. Для этого можно использовать так называемый метод центров тяжестей. Пусть некоторое количество сравнений уже было проведено, то есть наше множество элементов частично упорядочено. Назовем линейными продолжениями такого множества те перестановки из N элементов, которые не противоречат уже имеющемуся частичному упорядочению. Тогда $a(i)$ — среднее место i-го элемента из входного массива a в упорядоченном массиве, по имеющейся на настоящий момент информации можно подсчитать так:

$$a(i) = \frac{1}{k(a)} \sum_{\substack{\text{Линейные} \\ \text{продолжения}}} p(i),$$

$p(i)$ — место i-го элемента в линейном продолжении,
$k(a)$ — общее число линейных продолжений элементов массива a.

Два различных элемента i и j не сравнимы тогда и только тогда, когда $b(i, j) = |a(i) - a(j)| < 1$. Более того, в очередном сравнении должны участвовать элементы i и j, значение $b(i, j)$ у которых минимально и $i < j$. После этого количество линейных продолжений уменьшается, средние места элементов в них пересчитываются и выбирается новая пара элементов для сравнения. Отсортированный массив соответствует единственному оставшемуся линейному продолжению, места всех элементов в котором определены, и $b(i, j) \geq 1$, если $i \neq j$.

Покажем работу этого алгоритма на следующем примере. Пусть для 5 элементов уже известно, что $a_1 \leq a_2 \leq a_3$ и $a_4 \leq a_5$. Перечислим все 10 линейных продолжений этого множества и подсчитаем среднее место каждого из 5 элементов в них:

4	5	1	2	3	$a(1) = (6 \cdot 1 + 3 \cdot 2 + 1 \cdot 3)/10 = 1{,}5$
4	1	5	2	3	$a(2) = (3 \cdot 2 + 4 \cdot 3 + 3 \cdot 4)/10 = 3$
4	1	2	5	3	$a(3) = (1 \cdot 3 + 3 \cdot 4 + 6 \cdot 5)/10 = 4{,}5$
4	1	2	3	5	$a(4) = (4 \cdot 1 + 3 \cdot 2 + 2 \cdot 3 + 1 \cdot 4)/10 = 2$
1	4	5	2	3	$a(5) = (1 \cdot 2 + 2 \cdot 3 + 3 \cdot 4 + 4 \cdot 5)/10 = 4$
1	4	2	3	5	Минимальные $b(i, j)$: $b(1,4) = b(3,5) = 0{,}5$
1	4	2	3	5	То есть сравнивать нужно 1-й и 4-й или
1	2	4	5	3	3-й и 5-й элементы, затем повторить про-
1	2	4	3	5	цедуру подсчета средних мест в оставших-
1	2	3	4	5	ся линейных продолжениях.

Теперь можно показать, в каком порядке оптимальный по количеству сравнений алгоритм должен сравнивать элементы изначально никак не упорядоченного массива. До начала сортировки допустимыми линейными продолжениями являются все $N!$ перестановок элементов массива, все элементы в которых равноправны. Так, для 5 элементов все $a(i) = 3$, а $b(i, j) = 0$. Тогда сначала мы сравниваем два любых элемента, а потом — два любых из оставшихся, так как значение b для них после первого сравнения останется нулевым, то есть минимально возможным. Подобным образом выполняются сравнения на первом шаге, например, алгоритма сортировки слиянием. Пусть теперь известно, что $a_1 \leq a_2$ и $a_4 \leq a_3$. Очевидно, что $b(1, 4) = b(2, 3) = 0$, причем сами значения a для упомянутых элементов можно и не подсчитывать. Таким образом, на третьем шаге мы должны сравнить, например, максимальные элементы в упорядоченных парах. Пусть окажется, что $a_2 \leq a_3$. Тогда по оставшимся 15 линейным продолжениям легко подсчитать значения a: $a(1) = 1{,}6$; $a(2) = 3{,}2$; $a(3) = 4{,}8$; $a(4) = 2{,}4$; $a(5) = 3$. То есть на четвертом шаге, отличном от универсальных сортировок, сравнению подлежат элементы a_2 и a_5. Вне зависимости от результатов этого сравнения за оставшиеся 3 сравнения массив будет упорядочен.

Открытым остается вопрос, всегда ли количество сравнений в таком оптимальном алгоритме совпадает с точной нижней оценкой для всех алгоритмов сортировки. К сожалению, это не так. Уже при $N = 12$ оптимальному алгоритму потребуется сделать 30 сравнений вместо 29 ожидаемых. Дальнейшее изучение этого вопроса затруднено в силу чрезвычайной трудоемкости алгоритма, основанного на методе центров тяжестей.

В заключение данного раздела покажем, как может быть сформулирована олимпиадная задача, при решении которой требуется построить алгоритм оптимальный или асимптотически оптимальный по количеству сравнений.

> Входным параметром в задаче является число N. После его считывания ваша программа должна выдавать номера элементов, сравнить которые ей требуется на очередном шаге, и считывать результат этого сравнения. Когда будет произведено достаточное для вашего алгоритма число сравнений, программа должна выдать индексы элементов в массиве в порядке неубывания значений соответствующих элементов.

Вместо диалогового режима работы может быть предложена специальная библиотечная функция из созданного специально для этой задачи модуля, путем обращения к которой можно будет получать информацию о соотношении значений у пары элементов.

Сортировка данных специального вида

Выше было показано, как следует использовать так называемые универсальные алгоритмы сортировки, пригодные для любого вида данных, подлежащих сравнению. Однако зачастую информация о множестве значений, которые могут принимать элементы массива, может в корне изменить подход к сортировке. Пусть, например, требуется отсортировать 100 000 *цифр*, вводимых, например, из стандартного потока ввода (это типичная задача для районных или городских олимпиад по информатике). Как уже говорилось в лекции 3, максимальный размер массива даже однобайтовых элементов не может превышать 64 килобайтов. Кроме того, время работы любого из универсальных "эффективных" (то есть выполняющих порядка $N\log_2 N$ действий) для 100 000 элементов скорее всего будет превышать допустимое по условию время решения задачи. Поэтому обратим внимание на то, что элементами массива, подлежащего сортировке, являются не произвольные числа, а цифры, то есть целые числа от 0 до 9. В этом случае алгоритм сортировки фактически заключается в подсчете количества каждой из цифр и записи результата согласно полученным результатам подсчета. Реализовать этот алгоритм можно так:

```
var d : array[0..9] of longint;
    {массив для подсчета}
    a : byte;{переменная для ввода цифр}
    n, i, j : longint;
begin
  read(n);{n — количество цифр}
  fillchar(d,sizeof(d),0);
  for i := 1 to n do
  begin
    read(a);
    d[a] := d[a] + 1
  end;
  for j := 0 to 9 do {печатаем результат}
    for i := 1 to d[j] do
      write(j:2)
end.
```

Заметим, что эта программа работает верно и в случае, когда какой-либо из цифр во вводимой последовательности нет совсем. Подобный алгоритм называют сортировкой подсчетом, или "карманной" сортировкой [1, 5]. Его трудоемкость составляет $O(N + m)$ (а именно, $2N + m$ операций), где m — количество различных значений элементов в массиве. Очевидно, что если $m \le N$ и мы можем отвести память для подсчета количества каждого из m возможных значений для элементов массива, то алгоритм окажется линейным относительно N. Если же $m > N$, то алгоритм может как выигрывать, так и проигрывать по сравнению с универсальными эффективными алгоритмами. Например, для массива из 1000 положительных чисел типа **integer** сортировка подсчетом выполняется за $32\,767 \cdot 2 + 1000 = 66\,534$ операции, а, например, сортировка слиянием — за $2 \cdot 1000 \cdot \lceil \log_2 1000 \rceil = 20\,000$ операций. Но уже для 10 000 таких же чисел сортировка подсчетом выполняется за 75 534 операции, а слиянием — за 280 000.

В качестве упражнения, иллюстрирующего ту же самую идею, что и сортировка подсчетом, реализуйте программу, определяющую, какая буква встречается в тексте чаще всего. Напоминаем, что в Турбо/Паскале индексами массива могут служить и символы, в данном случае это можно использовать при описании дополнительного массива. Не забудьте также, что одной и той же букве алфавита соответствуют 2 различных символа, что следует учесть при подсчете (а для русских букв это не настолько просто, как для латинских).

Идею сортировки подсчетом продолжает "цифровая" сортировка, объектами которой могут служить произвольные числа и даже строки. Такой алгоритм ранее использовался для сортировки перфокарт [5]. В перфокартах цифры кодировались одиночными дырками в строках 0—9 соответствующей колонки. Сортировочной машине указывали столбец для сортировки, и она раскладывала перфокарты на 10 стопок в зависимости от того, какая из дырок была пробита. Как же с помощью этой машины можно было отсортировать колоду перфокарт с многозначными цифрами? Как ни странно, процедуру сортировки начинали не со старшего разряда, а с младшего. Полученные в результате первой сортировки 10 стопок вновь складывали в одну колоду (начиная с нулей в младшем разряде и заканчивая девятками) и сортировали уже по разряду десятков и т.д. Если числа являлись k-значными, то после k операций поразрядной сортировки колода оказывалась упорядоченной. Продемонстрируем это на примере трехзначных чисел (каждый столбец, начиная со второго, показывает результат сортировки исходного столбца по очередному разряду):

329	720	720	329
457	355	329	355
657	436	436	436
839	457	839	457
436	657	355	657
720	329	457	720
355	839	657	839

Когда числа сортируются по какому-либо разряду, важно, чтобы применяемый при этом алгоритм сортировки был устойчивым, то есть числа, у которых в текущем разряде стоят одни и те же цифры, сохраняли то же взаимное расположение, какое было между ними перед сортировкой по этому разряду. Устойчивым является, например, модифицированный алгоритм сортировки подсчетом, в котором в элементе вспомогательного массива `d[x]` будет храниться количество элементов в массиве, не превосходящих `x`. Покажем, как изменится при этом программа сортировки массива цифр, расположенных в переменной `a` типа `list` (результат поместим в массив `b` того же типа):

```
fillchar(d,sizeof(d),0);
for i := 1 to n do
  d[a[i]] := d[a[i]] + 1;
for j := 1 to 9 do
  d[j] := d[j] + d[j-1];
{d[j] — количество элементов, не превосходящих j}
for i := n downto 1 do
  begin
    b[d[a[i]]] := a[i];
    d[a[i]] := d[a[i]]-1
  end
```

В самом деле, в отсортированном массиве `a[i]` заведомо может стоять на месте `d[a[i]]`, ведь именно столько элементов в массиве не превосходят `a[i]`. Затем `d[a[i]]` уменьшается на единицу, и другие элементы массива, равные `a[i]`, будут записаны левее. Взаимный порядок между

равными элементами при этом не нарушится, так как при записи результата массив просматривается с конца.

Оценим время работы алгоритма цифровой сортировки с поразрядным использованием сортировки подсчетом. Для N чисел с k знаками (или для строк длины k), в записи которых участвуют m различных чисел (символов), количество операций имеет порядок $O(kN + km)$. Если k фиксированно и $m \le N$, то общее количество действий имеет порядок $O(N)$. Как и ранее, коэффициент в таком линейном алгоритме следует сравнивать со значением $\log_2 N$ для определения области его эффективного применения. Однако цифровая сортировка совершенно незаменима для упорядочения массивов данных, имеющих несколько различных полей. Например, при сортировке дат, заданных значением года, месяца и числа и т.п.

На идее, схожей с цифровой сортировкой, основано решение задачи "Редактор" I Всероссийской командной олимпиады по программированию [9].

Обратные сортировке задачи

Пусть заданы конкретная реализация алгоритма сортировки и значение N. Требуется восстановить некоторые его характеристики. Подобная задача предлагалась на открытой командной олимпиаде по программированию в Санкт-Петербурге в 1996 году (см. текст задачи в рамке).

Приведем рекурсивную процедуру, решающую эту задачу:

```
procedure Fill (var A : list; u, v : integer;
                start, step : integer);
begin
  if u = v then A [u] := start
  else begin
    Fill (A, u, (u + v) div 2, start, step*2);
   Fill (A, (u + v) div 2 + 1, v, start + step, step*2)
  end
end;
```

Массив A будет заполнен в результате следующего обращения к такой процедуре: `Fill(A, 1, N, 1, 1)`. Кроме того, можно потребовать определить количество сравнений, выполняемых алгоритмом в худшем случае, причем в такой задаче ограничения на величину N уже практически отсутствуют, поэтому решить ее путем подстановки заполненного массива A в исходную процедуру сортировки и включения в нее счетчика сравнений невозможно. Попробуйте написать программу, подсчитывающую количество сравнений, выполняемых приведенной выше процедурой сортировки массива слиянием, по введенному значению N. Саму процедуру сортировки, а также рекурсию в программе использовать не нужно. Подобные задачи можно поставить и для других алгоритмов сортировки. Постройте пример массива, на котором стандартная процедура QuickSort будет выполнять максимальное количество сравнений. Эта задача имеет и практическое значение. По построенным для различных значений N примерам можно понять, для каких именно массивов быстрая сортировка не является эффективной и действительно ли в этом случае количество сравнений имеет порядок $O(N^2)$.

Литература

1. *Ахо А.А., Хопкрофт Д.Э., Ульман Д.Д.* Структуры данных и алгоритмы. М.: Вильямс, 2000.

2. *Вирт Н.* Алгоритмы и структуры данных. М.: Мир, 1989.

3. *Окулов С.М.* Основы программирования. "Информатика" № 23, 2001.

4. *Окулов С.М.* Сортировка и поиск. "Информатика" № 33, 35, 2000.

5. *Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы. Построение и анализ. М.: МЦНМО, 2000.

6. *Кнут Д.* Искусство программирования. Том 3: Поиск и сортировка. М.: Вильямс, 2000.

7. *Окулов С.М., Шулятников Д.С.* Разбор задач международной олимпиады 2000 года. "Информатика" № 12, 2001.

8. *Хачиян Л.Г.* Проблемы оптимальных алгоритмов в выпуклом программировании, декомпозиции и сортировке. В кн.: Компьютер и задачи выбора. М.: Наука, 1989.

9. *Станкевич А.С.* Решение задач I Всероссийской командной олимпиады по программированию. "Информатика" № 12, 2001.

Ниже приведен алгоритм MSort, который сортирует слиянием заданный массив A из N целых чисел.

Требуется написать программу, которая для заданного N строит пример массива A, на котором достигается максимальное число сравнений. Все элементы массива A должны быть различными и находиться в диапазоне от 1 до N.

Пример исходных данных: 3

Результат: 2 1 3

```
procedure MSort (N : integer; var A : list);
var Help : list;
  procedure Make (u, v : integer);
  var i, j, m, k : integer;
  begin
    if u < v then
     begin
       m := (u + v) div 2;
       Make (u, m);
       Make (m + 1, v);
       i := u; j := m + 1; k := u;
       while (i <= m) and (j <= v) do
       begin
         if A [i] < A [j] then begin Help[k] := A[i]; i := i + 1 end
                          else begin Help[k] := A[j]; j := j + 1 end;
           k := k + 1
       end;
       while i <= m do begin Help[k] := A[i]; i := i + 1; k := k + 1 end;
       while j <= v do begin Help[k] := A[j]; j := j + 1; k := k + 1 end;
       for i := u to v do A[i] := Help[i]
     end
  end
begin {MSort}
   Make (1, N)
end;
```

Лабораторные работы по алгоритмизации

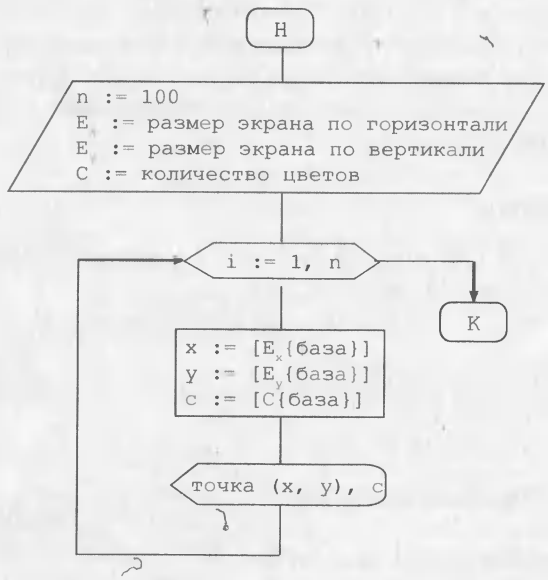
Д.П. Береговой,
Краснодарский край, станица Каневская

Окончание. См. № 44, 45/2001

Лабораторная работа № 5
Простейшие приемы построения графических изображений

Цель работы. Освоить простейшие приемы построения графических изображений с помощью ЭВМ.

Техническое задание 5.1а. Разработать алгоритм вывода на экран дисплея 100 точек (пикселей) случайных цветов в случайных местах экрана.



Алгоритм 5.1а. Эта очень простая задача позволит освоить основные приемы работы с графикой на ЭВМ. Прежде всего необходимо задать параметры графического объекта (или *примитива*). Поскольку в нашем случае это точка, то необходима пара чисел, определяющих положение точки относительно верхнего левого угла экрана; а также *атрибуты* примитива: для точки это ее цвет.

Поскольку все параметры задаются случайным образом, очевидно, что нужно воспользоваться генератором случайных чисел (см. алгоритм и программу 3.1г). Здесь, однако, имеется одна тонкость. Требуется учесть то, что размеры экрана и количество цветов, которые можно использовать, ограничены.

Для этого служат переменные E_x, E_y и C, которые определяют размер экрана по горизонтали и вертикали, а также номер цвета точки соответственно. И еще: поскольку выводить нужно заранее заданное количество точек, целесообразно воспользоваться циклом с параметром.

Программа 5.1а. Реализация вывода графики на языке Бейсик имеет ряд особенностей. Во-первых, необходимо переключиться в режим работы с графикой. Для этого имеется оператор SCREEN (в переводе на русский — экран). Во-вторых, нужно задать режим графического дисплея. Режимы графического дисплея нумеруются с 7-го по 13-й и определяют размер (разрешение) экрана и количество возможных цветов. В приведенной программе применен 12-й режим (он задается переменной Mode, что соответствует размеру дисплея 640×480 точек и палитре из 16 цветов. Не забудьте, что нумерация точек и цветов начинается с *нуля*, что и отражено в начальных установках переменных E_x, E_y и C.

В качестве начального значения для генератора случайных чисел здесь использованы показания системных часов компьютера. Они вызываются функцией TIMER, которая возвращает количество секунд, прошедших с полуночи.

```
RANDOMIZE TIMER
DEFINT A-Z
CLS
n = 100
EndX = 639
EndY = 479
C = 15
Mode  = 12
SCREEN Mode
FOR i = 1 TO n
  x = CINT(EndX * RND(TIMER))
  y = CINT(EndY * RND(TIMER))
  ColorPoint = CINT(C * RND(TIMER))
  PSET (x, y), ColorPoint
NEXT i
END
```

Техническое задание 5.1б. Даны координаты левого верхнего и правого нижнего углов прямоугольника:

$$x_{лв} = 100,\ y_{лв} = 100;$$
$$x_{пн} = 350,\ y_{пн} = 250.$$

Произвести параллельный перенос прямоугольника на 150 пикселей вправо и вниз.

Алгоритм 5.1б. Любые преобразования графических объектов (в данном случае это примитив в виде прямоугольника) требуют пересчета его старых координат в новые. Так что без математики тут не обойтись. Иногда она достаточно проста, как в данном случае, а иногда и не очень. Ну и, конечно, для совершения такого рода перестроений на экране нужно знать элементарную геометрию. В частности, вспомните, что параллельный перенос любого объекта куда бы то ни было обозначает прибавление к координатам всех его точек одной и той

* Конечно, у вас эти значения могут быть иными.

же величины: в нашем случае к координате по оси x нужно прибавить 150 пикселей; по оси y — столько же.

Обозначим величину переноса вправо dx, а величину переноса вниз — dy. Не забывайте, что на экране дисплея ось OX расположена обычно, но ось OY направлена от левого верхнего угла экрана вертикально вниз!

Имеем следующую математическую модель параллельного переноса (индексом $н$ обозначены новые координаты):

$$x_{лвн} = x_{лв} + dx,$$
$$x_{пнн} = x_{пн} + dx,$$
$$y_{лвн} = y_{лв} + dy,$$
$$y_{пнн} = y_{пн} + dy.$$

Таким образом, алгоритм решения задачи будет состоять из следующих блоков:

- ввод исходных данных;
- пересчет координат;
- построение прямоугольников.

Как видите, ничего необычного нет.

Программа 5.16. В программе реализован перенос исходного прямоугольника (его координаты заданы переменными Xlu, Ylu и Xrd, Yrd, кроме того, он закрашен темно-бирюзовым цветом) на заданные величины (переменные Dx, Dy). Новый прямоугольник закрашен светло-бирюзовым цветом. Кроме того, желтым показаны линии переноса базовых координат прямоугольника. Напомним, что переход в графический режим осуществляется с помощью оператора SCREEN

```
DEFINT A-Z
CLS
Xlu = 100: Ylu = 100
Xrd = 350: Yrd = 250
Dx = 150: Dy = 150
XluNew = Xlu + Dx: YluNew = Ylu + Dy
XrdNew = Xrd + Dx: YrdNew = Yrd + Dy
SCREEN 12
LINE (Xlu, Ylu)-(Xrd, Yrd), 3, BF
LINE (XluNew, YluNew)-(XrdNew, YrdNew), 11, BF
LINE (Xlu, Ylu)-(XluNew, YluNew), 14
LINE (Xrd, Yrd)-(XrdNew, YrdNew), 14
END
```

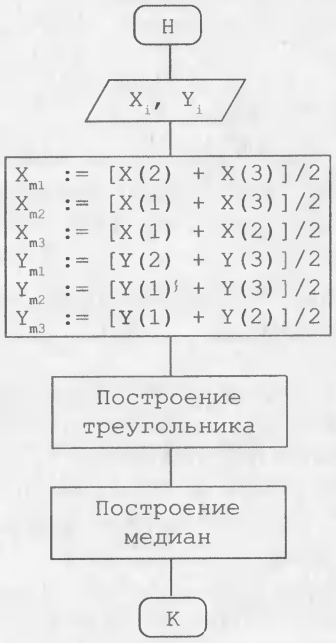
Техническое задание 5.1в. Вершины треугольника заданы тремя парами чисел. Разработать алгоритм построения его медиан. Результат вывести на дисплей в графическом виде.

Напоминание. Медианой называется отрезок, соединяющий вершину треугольника с серединой противоположной стороны.

Алгоритм 5.1в: Напоминание должно навести на мысль о том, что для построения отрезка прямой нужно иметь по крайней мере две пары координат, указывающих его начало и конец. С координатами начала этих отрезков проблем нет, так как вершины треугольника заданы. А вот координаты концов необходимо вычислять. Хотя сами вычисления достаточно примитивны — нужно определить среднее арифметическое двух чисел, но весьма поучительны, так как иллюстрируют общее правило: перед тем как что-то нарисовать на экране, нужно рассчитать координаты точек объекта.

Итак, если обозначить через x_i, y_i координаты вершин треугольника, а через x_{mi}, y_{mi} — координаты медиан, то модель расчета координат точек медиан будет выглядеть так:

$$x_{m1} = \frac{1}{2}(x_2 + x_3);\ y_{m1} = \frac{1}{2}(y_2 + y_3),$$
$$x_{m2} = \frac{1}{2}(x_1 + x_3);\ y_{m2} = \frac{1}{2}(y_1 + y_3),$$
$$x_{m3} = \frac{1}{2}(x_1 + x_2);\ y_{m3} = \frac{1}{2}(y_1 + y_2).$$



Алгоритм, таким образом, не будет отличаться особенной сложностью, и это является следствием простоты расчетов.

Программа 5.1в. В приведенной ниже программе использованы следующие приемы. Во-первых, координаты вершин треугольника генерируются случайным образом. Это дает возможность увидеть, как работает модель построения медиан, и, кроме того, помогает убедиться в том, что медианы пересекаются в одной точке.

Во-вторых, в программу введен новый оператор WINDOW, который служит для определения размеров окна, в которое будет выводиться графика. В предложенном варианте используется окно, в котором определяется "нормальная" система координат, то есть шкала оси ординат возрастает снизу вверх. Этот эффект достигается указанием координат окна вывода. При этом точка дисплея с координатами (0, 0) переносится в точку с координатами (0, 479). Однако имейте в виду, что использование оператора WINDOW в два раза замедляет процесс вывода графики на экран дисплея, так что использовать его нужно только в случае крайней необходимости.

```
RANDOMIZE TIMER
CLS
DIM i AS INTEGER
DIM X(3) AS INTEGER, Y(3) AS INTEGER
```

```
FOR i = 1 TO 3
  X(i) = CINT(500 * RND(TIMER))
  Y(i) = CINT(300 * RND(TIMER))
NEXT i
Xm1 = (X(2) + X(3)) / 2
Ym1 = (Y(2) + Y(3)) / 2
Xm2 = (X(1) + X(3)) / 2
Ym2 = (Y(1) + Y(3)) / 2
Xm3 = (X(1) + X(2)) / 2
Ym3 = (Y(1) + Y(2)) / 2
SCREEN 12
WINDOW (0, 479)-(639, 0)
LINE (X(1), Y(1))-(X(2), Y(2)), 14
LINE (X(2), Y(2))-(X(3), Y(3)), 14
LINE (X(3), Y(3))-(X(1), Y(1)), 14
LINE (X(1), Y(1))-(Xm1, Ym1), 9
CIRCLE (Xm1, Ym1), 3, 9
LINE (X(2), Y(2))-(Xm2, Ym2), 9
CIRCLE (Xm2, Ym2), 3, 9
LINE (X(3), Y(3))-(Xm3, Ym3), 9
CIRCLE (Xm3, Ym3), 3, 9
END
```

Для наглядности середины сторон дополнительно обведены окружностями.

Техническое задание 5.2а. Разработать алгоритм вывода на экран дисплея 10 окружностей случайных радиусов (от 0 до 10 пикселей) с центрами, расположенными в случайных местах экрана. Цвета окружностей также задавайте случайно, выбирая их из первой восьмерки цветов.

Рекомендации: Здесь можно воспользоваться алгоритмом и программой 5.1а. Однако потребуется еще и сгенерировать радиус окружности. Кстати, попробуйте использовать параметр, задающий сжатие окружностей, и посмотреть, что из этого выйдет.

Техническое задание 5.2б. Задан прямоугольник координатами верхнего левого и нижнего правого углов:

$$x_{лв} = 100,\ y_{лв} = 100;$$
$$x_{пн} = 350,\ y_{пн} = 250.$$

Построить на экране дисплея прямоугольники в 2 раза больше и в 3 раза меньше заданного. Левые верхние углы исходного и полученных прямоугольников должны совпадать. Для каждого прямоугольника использовать свою окраску границы.

Рекомендации: Чтобы решить поставленную задачу, нужно для начала рассчитать длину диагонали исходного прямоугольника, а затем — длины диагоналей новых.

Техническое задание 5.2в. Построить в центре экрана дисплея следующий набор графических элементов:

- квадрат со стороной 100 пикселей;
- диагонали этого квадрата;
- окружность, описывающую квадрат;
- окружность, вписанную в квадрат.

Рекомендации: Достаточно правильно построить квадрат, тогда координаты всех остальных объектов легко определяются. Попробуйте сначала изобразить требуемые объекты на бумаге, и вы сами в этом убедитесь.

Техническое задание 5.2г. Разработайте алгоритм построения на экране дисплея координатной сетки, состоящей из 10 отрезков по вертикали и 10 отрезков по горизонтали. Оси сетки должны быть выделены цветом, а граничные линии — располагаться на краях экрана дисплея. Центр координатной сетки должен совпадать с центром экрана.

Рекомендации: Понятно, что для построения такой сетки потребуется использовать циклический алгоритм, так как выполняется одна и та же операция: рисование отрезков. При этом лишь изменяются координаты их концов. Необходимо организовать два цикла: для горизонтальных и вертикальных линий отдельно.

Не забудьте также, что экранные координаты не совпадают с общепринятыми. Например, $Bx = 0$, $By = 0$ — координаты начала системы координат *дисплея* — это левый верхний угол, а $Ex = 639$, $Ey = 463$ — координаты правого нижнего угла. Соответственно центр системы координат для вашей задачи определится так:

$$Cx = (Bx + Ex) \setminus 2,\ Cy = (By + Ey) \setminus 2.$$

Обратите внимание, что здесь использовано целочисленное деление на 2, поскольку в выражении стоит обратный слэш (\).

В качестве параметров циклов лучше всего использовать номера линий и строить их, пользуясь следующими формулами:

- для изменения x-й координаты вертикалей:

$$Cx \pm i \cdot Dx$$

- для изменения y-й координаты горизонталей:

$$Cy \pm i \cdot Dy.$$

Здесь Dx, Dy — величины шагов по горизонтали и вертикали соответственно;
$i = 0, 1, 2, 3, 4$ — номер линии.

Подумайте, каковы должны быть величины шагов Dx и Dy?

Контрольные вопросы

1. Дайте определение понятию "машинная (компьютерная) графика".
2. Что такое графический дисплей и чем он характеризуется?
3. Что определяет режим работы графического дисплея?
4. Укажите особенности координатной сетки графического дисплея.
5. Что такое графический примитив?
6. Перечислите основные виды графических примитивов.
7. Опишите параметры основных графических примитивов.
8. Перечислите технологические приемы построения изображений в ЭВМ.

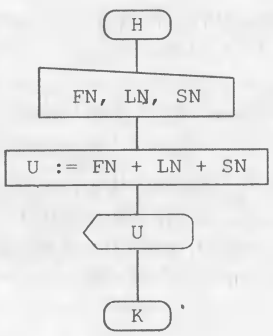
Лабораторная работа № 6
Основные приемы обработки символьных данных

Цель работы. Освоение основных приемов обработки символьных данных с помощью ЭВМ.

Техническое задание 6.1а. Введите ваши фамилию, имя и отчество в три различные переменные. Организуйте присвоение фамилии, имени и отчества одной переменой. Результат вывести на дисплей.

Алгоритм 6.1а. Для решения поставленной задачи потребуется простейший линейный алгоритм. Но это как раз тот случай, когда "маленький шаг одного человека" может привести к большим последствиям. В различных задачах мы уже использовали символьные данные (см., например, технические задания 1.1а, 1.1в, 1.2а ...), но в них все ограничивалось вводом-выводом какого-то заранее заданного текста. Все муки творчества сводились к написанию правильного синтаксиса предложения в программе.

Сейчас же нам требуется не только ввести, но и обработать символьную информацию. Для объединения символов служит операция конкатенации. В переводе на русский язык — сцепление, что, конечно, запоминается легче. Блок-схема алгоритма приведена ниже. В нем использованы следующие переменные: в FN, LN, SN вводятся фамилия, имя, отчество соответственно. Переменная U содержит результат их конкатенации. Не забудьте, что все переменные символьные!



Программа 6.1а. Вы, наверное, помните, что в языке Бейсик для группового определения типа переменных можно воспользоваться служебным словом DEF. В приведенной программе это использовано для того, чтобы объявить символьными все переменные, чьи имена начинаются на букву S. Кроме того, имена символьных переменных начинаются с префикса str (от англ. *string* — символьная строка).

```
DEFSTR S
CLS
INPUT "Введите фамилию: ", strFirstName
INPUT "Введите имя: ", strLastName
INPUT "Введите отчество: ", strSecondName
strUnion = strFirstName + strLastName +
           strSecondName
PRINT strUnion
END
```

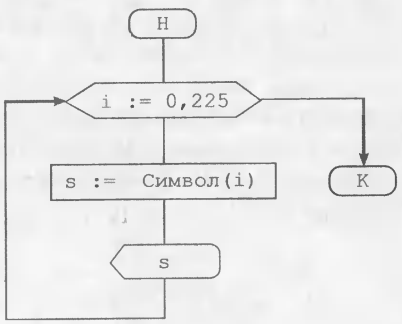
После выполнения программы у вас должен возникнуть по крайней мере один вопрос: "почему все введенные значения "прилеплены" друг к другу без пробелов и что с этим делать?"

Ответ: А у вас не возникает вопроса, почему между словами не появился значок "+" или, скажем, запятая? Важно понять, что пробел — точно такой же символ, как и все остальные, и нет никаких причин добавлять лишние символы при операции сцепления. Поэтому, чтобы выводимое выражение было более читабельным, вставьте необходимые пробелы, например, вот так:

```
strUnion = strFirstName + " " +
           strLastName +  " " + strSecondName
```

Техническое задание 6.1б. Вывести на экран дисплея символьный набор ЭВМ.

Алгоритм 6.1б. Чтобы выполнить это задание, нужно вспомнить, что каждый символ имеет свой ASCII-код, который выражается в виде двоичного (шестнадцатеричного) или десятичного числа. Причем эти коды лежат в диапазоне $[0, 255_{10}]$. Значит, нужно организовать цикл, в котором перебираются эти значения и соответствующие им символы выводятся на дисплей. Преобразование кода в символ производится с помощью некоей функции. В алгоритме мы ее просто обозначим *Символ*(<*код*>).



Программа 6.1б. В языке Бейсик для преобразования кода в его символьное представление имеется функция CHR$(<*код*>), от англ. *character* — буква. Обратите внимание, что функция возвращает символ, поэтому в ее имени присутствует символ "$".

```
DEFINT I
DEFSTR S
CLS
FOR i = 0 TO 255
  strSymbol = CHR$(i)
  PRINT strSymbol;
NEXT i
END
```

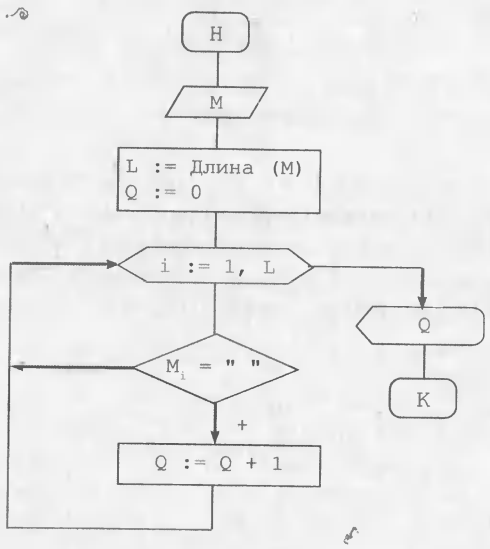
Символы выводятся в строку. Заметьте, что в процессе выполнения программы компьютер должен "бибикнуть". Таким образом он отреагирует на вывод символа

CHR$(7), так как код 7_{10} соответствует звуковому сигналу. Полюбуйтесь на набор символов вашей ЭВМ и вспомните, что первые 32 символа (коды $00—31_{10}$) — это управляющие (непечатаемые) символы. Они не предназначены для вывода на экран, однако при попытке это сделать можно получить интересные эффекты. Кроме того, целый набор символов отсутствует на клавиатуре (например, √), и иначе, как через функцию CHR$(), на дисплей их не вывести.

Техническое задание 6.1в. Дан набор символов: "Информатика — это область знаний, изучающая природу создания, хранения и передачи информации в естественных и технических системах". Определить количество пробелов в нем.

Алгоритм 6.1в. Данная задача является как бы обратной к сформулированной в техническом задании 6.1а. Там нам нужно было из набора символов (или слов, что с точки зрения алгоритма все равно) сконструировать одно сообщение, то есть произвести *синтез* выражения. Здесь же нужно выяснить: есть ли внутри сообщения требуемый символ, то есть произвести *анализ* имеющегося выражения.

Первое, что приходит в голову: нужно просматривать каждый символ сообщения и проверять, пробел это или нет. И если результат проверки "истина", то увеличить значение некоей переменной на 1. То есть налицо цикл с разветвлением (см. алгоритм 3.1г). Что нужно добавить? Во-первых, перед организацией цикла надо знать длину сообщения *L* и, во-вторых, ввести переменную, в которой будет подсчитываться количество пробелов, назовем ее *Q*. При обнаружении пробела в наборе символов ее значение будет изменяться на единицу, поэтому такие переменные называют *счетчиками*. Блок-схема соответствующего алгоритма приведена ниже. Запомните этот прием, он встречается достаточно часто. (Под M_i подразумевается текущий символ сообщения.)



Программа 6.1в. Успех реализации алгоритма зависит от того, насколько хорошо вы знаете имеющийся в языке (в нашем случае это Бейсик) набор функций, предназначенных для обработки символьных данных. В рассматриваемой задаче нам понадобятся две наиболее часто используемые функции:

- определения длины сообщения — LEN(<*сообщение*>);
- "вырезание" группы символов — MID$(<*сообщение*>, <*текущий номер символа*>, <*количество символов*>).

Заметьте, что функция LEN() возвращает число, поэтому не имеет суффикса, а функция MID$() возвращает символы, поэтому снабжена суффиксом $.

```
DEFINT I
DEFSTR S
CLS
CONST strMessage = "Информатика — это
                    область знаний, изучающая
                    природу создания,
                    хранения и передачи
                    информации в естественных
                    и технических системах."
intLength = LEN(strMessage)
intQ = 0
FOR i = 1 TO intLength
  strLetter = MID$(strMessage, i, 1)
  IF strLetter = " " THEN intQ = intQ + 1
NEXT i
PRINT "В выражении найдено "; intQ; " пробелов."
END
```

Техническое задание 6.2а: С клавиатуры вводятся два произвольных символа. Определить порядок их следования в ASCII-таблице.

Рекомендации. Для решения данной задачи нужно вспомнить, что сравнение символов, так же как и чисел, производится с помощью операций отношения. Собственно, сравниваются коды символов, которые в ASCII-таблице расположены в алфавитном порядке. Вывод результата сравнения можно оформить в виде, например, такого сообщения: "Символ <*x*> расположен раньше символа <*y*>". Поэкспериментируйте с различными символами (цифры, спецсимволы, буквы латиницы и кириллицы).

Техническое задание 6.2б. Дан произвольный набор символов. Вывести на экран коды, соответствующие каждому символу сообщения.

Рекомендации: Для разложения строки на символы можно использовать тот же прием, что и в алгоритме 6.1в, то есть в цикле с помощью функции MID$() вырезать последовательно по одному символу, а затем преобразовывать каждый символ в его код с помощью функции ASC(<*символ*>).

Контрольные вопросы

1. Укажите особенности символьного типа данных.
2. Что такое стандарт ASCII?
3. Перечислите основные группы символьных кодов.
4. Что является результатом операции конкатенации?
5. Как происходит сравнение символов (строк символов)?
6. Перечислите основные группы символьных функций.

Приложение 1

Задачи для самостоятельного решения

1. Разработать алгоритм для вычисления значения выражения:

$$R = \frac{\sqrt{2x^2 - x + 1}}{\lg(x^3 + x - 1)} + \frac{1}{x}.$$

2. Разработать алгоритм вычисления среднего арифметического пяти чисел, задаваемых с клавиатуры.

3. С помощью показаний системного таймера определить, сколько прошло полных минут и часов от полуночи.

4. По произвольно заданным парам координат точек (x_1, y_1) и (x_2, y_2) определить расстояние между точками на плоскости.

5. Разработать алгоритм вычисления суммы кубов двузначных натуральных чисел. Результат вывести на экран дисплея.

6. Разработать алгоритм вычисления суммы квадратов двузначных натуральных чисел, кратных 3. Результат вывести на экран дисплея.

7. Разработать алгоритм генерации последовательности целых случайных чисел в заданном диапазоне $[a; b]$.

8. Дан одномерный числовой массив из p элементов. Рассчитать сумму его положительных элементов. Результат вывести на экран дисплея.

9. Дан одномерный числовой массив из k элементов. Разработать алгоритм для подсчета суммы четных элементов массива. Результат вывести на экран дисплея.

10. Задана последовательность из m чисел. Разработать алгоритм для подсчета количества нулей, положительных, отрицательных чисел.

11. Заданы n пар чисел x и y. Разработать алгоритм присвоения переменной r значения 1, если $x < y$; 0, если $x = y$; и -1, если $x > y$. В алгоритме должен производиться подсчет соответствующих пар. После каждого сравнения переменную r выводите на экран дисплея.

12. Разработать алгоритм приближенного вычисления числа π. Результат должен быть выведен на экран дисплея. Воспользоваться формулой:

$$\frac{\pi^2}{6} = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \ldots + \frac{1}{15^2} = \sum_{i=1}^{15} \frac{1}{i^2}.$$

13. Заданы два одномерных массива A и B, содержащие по q чисел. Разработать алгоритм формирования массива C, содержащего $2q$ чисел, в который последовательно включены элементы массивов A и B.

14. Заданы два одномерных массива A и B, содержащих n чисел. Разработать алгоритм для формирования одномерного массива C, в котором четные элементы взяты из массива A, а нечетные — из массива B, причем $C(1) = B(1), C(2) = A(1), C(3) = B(2), C(4) = A(2)$ и т.д.

15. Задана функция: $y = \sqrt{\frac{f(x) - 1}{f(x) + 1}}$. Разработать алгоритм определения значения функции при $f(x) = x + 2$, $f(x) = x^2 - 2$, $f(x) = \ln(x + 1)$.

16. Разработать алгоритм решения квадратного уравнения $ax^2 + bx + c = 0$ для произвольных коэффициентов a, b и c.

17. Разработать алгоритм, который по введенным фамилии, имени и отчеству выводит фамилию и инициалы. Например, вводится: Иванов Иван Иванович; выводится: Иванов И. И.

18. Разработать алгоритм подсчета слов в введенном предложении.

19. Определить, является ли введенный набор символов палиндромом. Палиндром — это слово, одинаково читающееся слева направо и справа налево.

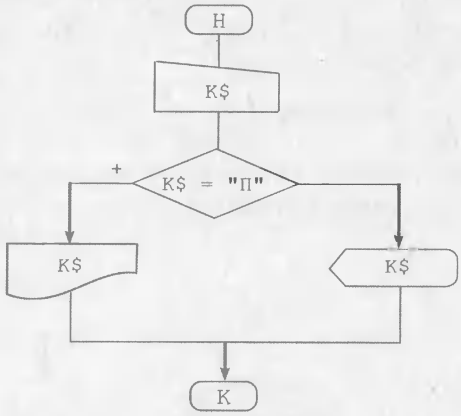
20. Вводится произвольное четырехзначное число. Разработать алгоритм вычисления суммы его цифр.

Приложение 2

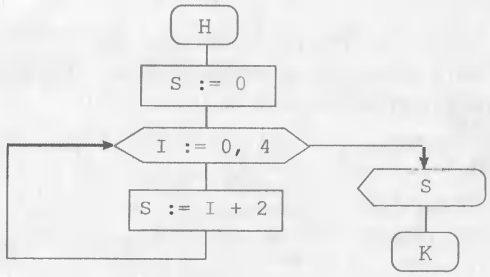
Контрольная работа. Основные алгоритмические структуры

Вариант 1

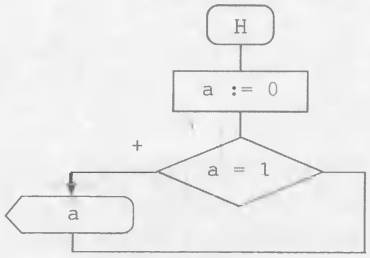
1. Дайте словесное описание блок-схемы алгоритма. Опишите его характеристики.



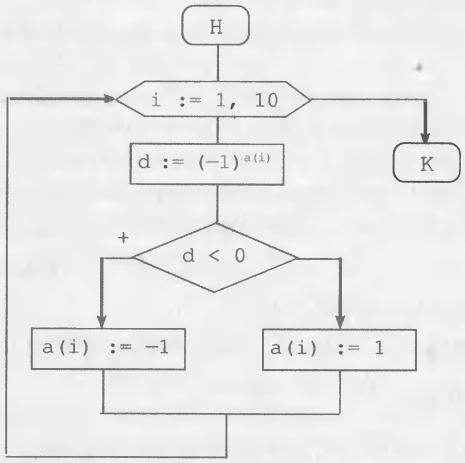
2. Укажите, что будет выведено на дисплей в результате выполнения приведенного ниже алгоритма.



3. Укажите ошибки, допущенные в блок-схеме.



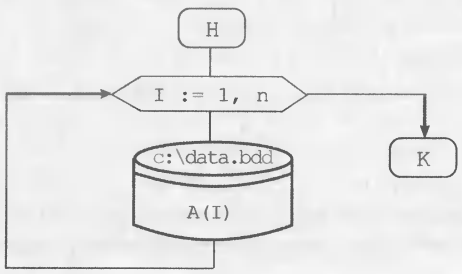
4. Имеется одномерный целочисленный массив из десяти элементов. Ниже дана блок-схема алгоритма, который заменяет четные *значения* элементов на 1, а нечетные — на —1. Модифицируйте алгоритм таким образом, чтобы четные *элементы* исходного массива были заменены на —1, а нечетные — на 1.



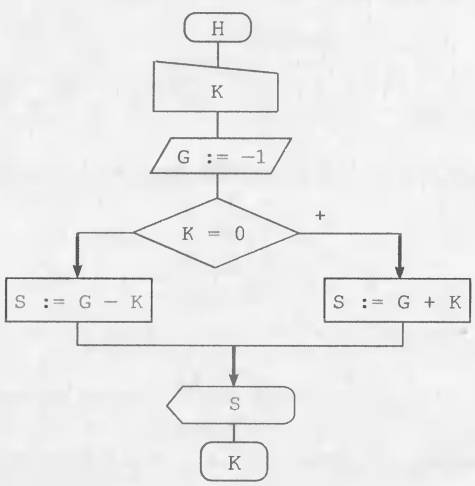
5. Дан двухмерный числовой массив. Разработайте алгоритм, заменяющий нулевые значения его элементов на 1, а всех остальных — на 0.

Вариант 2

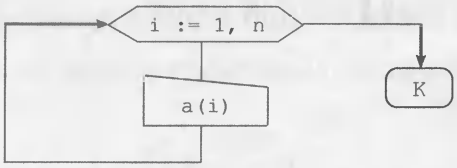
1. Дайте словесное описание блок-схемы алгоритма. Опишите его характеристики.



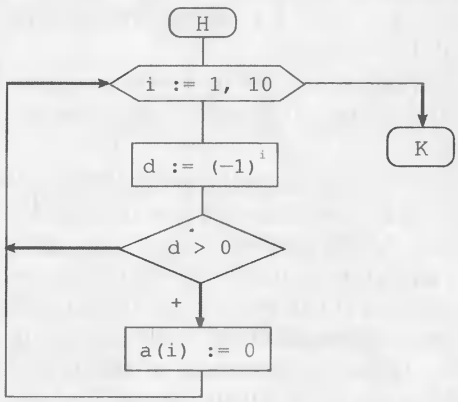
2. Укажите, что будет выведено на дисплей в результате выполнения приведенного ниже алгоритма, если было введено значение $k = 1$.



3. Укажите ошибки, допущенные в блок-схеме.



4. Имеется одномерный целочисленный массив, содержащий 10 элементов. Алгоритм, блок-схема которого приведена ниже, производит замену всех четных *элементов* массива на 0. Модифицируйте алгоритм таким образом, чтобы он заменял все нечетные *значения* элементов массива на 0.



5. Дан двухмерный числовой массив. Разработайте блок-схему алгоритма (и программу) его транспонирования (при транспонировании матрицы строки и столбцы меняются местами). Приведем пример транспонирования матрицы:

$$\begin{pmatrix} 0 & 1 & -1 \\ 1 & -1 & 0 \end{pmatrix} \rightarrow \begin{pmatrix} 0 & 1 \\ 1 & -1 \\ -1 & 0 \end{pmatrix}$$

Приложение 3

Отчет о выполнении лабораторной работы № __
"Название работы"

Цель работы. Она указана в пособии.

Техническое задание. Также приведено в пособии.

Алгоритм. Изобразите здесь блок-схему алгоритма решаемой задачи. Не забудьте, что аккуратность — одна из главных составляющих успеха. И обязательно опишите, для чего предназначена та или иная переменная.

Программа. Здесь должен быть листинг программы, если преподавателем не указано иное.

Протокол работы программы. Внимание! Все, что выводится на экран дисплея в процессе работы алгоритма: запросы значений переменных; сообщения; результаты обработки; служебные или аварийные сообщения — то есть абсолютно все, — записывается в этот протокол.

Компилятор? Это довольно просто!

Е.А. Еремин,
г. Пермь

Продолжение. Начало в № 40, 43, 45/2001

3.6. Стандартная процедура ввода read

Для ввода данных в Паскале имеются встроенные процедуры `read` и `readln`. Чтобы ввод с клавиатуры осуществлялся так же, как из дискового файла, в языке определен специальный **входной файл `input`**, в который попадает все, что набирается с клавиатуры. Процедуры чтения извлекают информацию именно оттуда, а не с клавиатуры.

Входной файл **`input`** автоматически открывается при старте программы. Он является текстовым, так что информация в нем представляет собой строки текста, разделенные специальным **признаком конца строки `eoln`** (*End Of LiNe*). Конкретный код **`eoln`** в рамках Паскаля не определен, но фактическим стандартом является сочетание кодов 13_{10} (ВК) и 10_{10} (ПС), уже описанное ранее в п. 2.2.

Пусть, например, после компиляции и запуска полученной программы пользователь набрал следующее:

`12 <пробел> 34` Enter

`56 <пробел> OK` Enter

Тогда в текстовый файл **`input`** попадет следующая информация (верхняя строка — в символьном виде, а две нижние — коды этих символов в шестнадцатеричной и десятичной системах счисления):

Символ	1	2		3	4	ВК	ПС	5	6		O	K	ВК	ПС
16 с/с	31	32	20	33	34	0D	0A	35	36	20	4F	4B	0D	0A
10 с/с	49	50	32	51	52	13	10	53	54	32	79	75	13	10

Рис. **3.6. Пример содержимого входного файла `input`**

Посмотрим, как эта информация будет прочитана. Наберем на Турбо Паскале короткую программу, чтение в которой осуществляется процедурой `read`:

```
program  wwod;
var  c, d : integer;
begin read(c); read(d);
      writeln(c); writeln(d)
end.
```

Запустим ее.

При первом вызове процедура `read` извлечет из входного потока символы "1", "2" и пробел, который свидетельствует об окончании первого числа. Следуя заложенному в нее алгоритму, процедура преобразует прочитанные цифровые символы в целое число 12, и результат (число $000C_{16}$) запишется в переменную `c`.

При повторении вызова аналогичным образом будет извлечено число 34 и записано в `d`. Вся вторая введенная строка окажется невостребованной.

Совсем другая картина получится, если вместо `read` использовать `readln`. Согласно своему назначению процедура `readln` не просто читает данные, но и находит конец строки, в которой они хранятся. Таким способом происходит ***подготовка к вводу из следующей строки***.

Вернемся к нашему примеру. При первом вызове процедура `readln` прочитает первые символы до пробела, а затем в поисках конца строки пропустит "3" и "4", а также сами коды **`eoln`** 13 и 10. Следующий ввод начнется с цифры 5. Поэтому повторный вызов `readln` примет число 56 и занесет его в `d`, что и подтвердит выведенное на экран значение этой переменной. Обратите внимание, что наличие текста "OK" вместо числа не было воспринято как ошибка только потому, что эти символы были просто "проброшены" при поиске конца строки, но не вошли в анализируемый текст числа.

Еще более тонко работают процедуры `read` и `readln` с типом **`char`**. Аккуратно используя вариант `read`, можно даже прочитать из буфера сами коды конца строки. Мы здесь не будем этим заниматься.

В компиляторе "Компас", чтобы не усложнять ситуацию, для ввода целых чисел ***всегда*** используется `readln`, а для типа **`char`** — `read`. Оба вызова транслируются в обращение к подпрограммам ПЗУ, первая из которых имеет адрес `4108`, а вторая — `40FA`. В последнем случае результат возвращается в `R0`, поэтому приходится добавить команду переписи его в `R1`:

```
9C0D
40FA
0101
```

Примечание. "Компас" воспринимает только один аргумент в `read` и `readln`: список параметров через запятую не разрешается.

Данные типа **`boolean`**, как и в Турбо Паскале, вообще не могут быть введены с клавиатуры.

Подчеркнем, что процедуры `read` и `readln` в "Компасе" не являются точными аналогами соответствующих процедур Турбо Паскаля. Поэтому особенности и тонкости ввода лучше изучать в "настоящем" Паскале.

Σ

Наиболее важные положения об организации в программе на Паскале ввода информации:

- Ввод информации в компьютер может осуществляться с различных устройств; наиболее часто для этой цели используют набор с клавиатуры и чтение данных с диска. Для того чтобы программирование ввода выглядело одинаково для любого из них, Н.Вирт предложил считать, что существует специальный стандартный входной файл **`input`**, через который попадает в программу поток данных с клавиатуры.
- Входной файл **`input`** автоматически открывается при старте любой программы. Он является текстовым, поэтому информация в нем представляет собой строки текста, разделенные специаль-

ным признаком конца строки **eoln**. Фактическим стандартом на разделитель строк текста во всех языках, в том числе и в Паскале, служит сочетание кодов 13 и 10 (ВК и ПС).

• Для чтения данных из стандартного файла **input** служат две встроенные в Паскаль стандартные процедуры `read` и `readln`. Процедура `read` считывает указанные ей переменные из входного файла и преобразует их из текстового представления во внутреннее. `Readln` дополнительно к этому пропускает неиспользуемую информацию до конца строки (вплоть до **eoln** включительно) и тем самым подготавливает последующий ввод с новой строки.

• Не все типы данных в Паскале могут вводиться с клавиатуры. Например, это невозможно сделать для **boolean**, перечисления, множества и некоторых других типов данных.

• Демонстрационный компилятор "Компас" не отражает всех нюансов ввода/вывода с помощью процедур `read` и `readln`: целочисленные данные в нем всегда читаются через `readln`, а символьные — через `read`.

3.7. Стандартная процедура вывода write

Вывод результатов на экран дисплея в Паскале осуществляется через стандартный выходной файл **output** с помощью двух процедур: `write` и `writeln`. Единственное различие между ними состоит в том, что в последнем случае происходит переход на следующую строку. Для этого в исполняемый код добавляется обращение к подпрограмме `40EC`, которая обеспечивает это действие.

Аргументом процедуры вывода может быть либо текст — `write('Text')`, либо имя переменной, значение которой мы хотим вывести, — `write(z)`.

Примечание. "Компас" воспринимает только один аргумент в `write` и `writeln`, список параметров через запятую не разрешается.

Оба указанных выше случая реализуются компилятором по-разному.

Для вывода на экран текста вызывается специальная подпрограмма с адресом `40DC`, которая в качестве аргумента использует расположенный после команды вызова подпрограммы текст. В байте, непосредственно предшествующем тексту, задается его длина. Например, для оператора `writeln('Text')` это выглядит следующим образом:

Код	Комментарии
9C0D 40DC	обращение к ПЗУ
5404 7865 0074	число 4 и "T" "ex" "t"
9C0D 40EC	переход на другую строку

Примечание. Число 4 означает длину текста — нулевой (пятый) байт не используется. Обратите внимание, что первым всегда выводится младший байт, который соответствует двум последним шестнадцатеричным цифрам.

Вывод значений переменных зависит от типа выводимых данных. Во всех случаях данные берутся из регистра `R1`. Для **char** и **boolean** просто вызываются соответствующие подпрограммы вывода в ПЗУ, расположенные по адресам `40BA` и `40C4`. Для целых чисел к вызову подпрограммы `4068` добавляется задание в `R3` начала рабочей области в ОЗУ, которая используется при переводе из двоичной системы счисления в десятичную:

Код	Комментарии
01D3 <*адрес буфера*>	задать адрес рабочей области
9C0D 4068	вывод целого числа

Адрес рабочей области является параметром компилятора (см. *рис.* 1.1 и его описание в п. 1.4).

Таким образом, мы видим, что компиляция операторов ввода и вывода в "Компасе" фактически сводится к подготовке и вызову соответствующих подпрограмм ПЗУ.

Σ

Основная идеология вывода в Паскале во многом похожа на идеологию ввода.

• Имеется стандартный выходной файл **output**, в который отправляется вся выводимая на экран информация.

• Вывод результатов осуществляется с помощью стандартных процедур `write` и `writeln`. В последнем случае после вывода указанной в качестве аргумента информации в выходной поток добавляется признак **eoln**. На экране это воспринимается как переход на следующую строку.

• Процедуры `write` и `writeln` могут выводить на экран значения переменных и произвольный поясняющий текст.

• Вывод на экран допускается не для всех типов переменных Паскаля. В то же время значения ***всех*** допускаемых компилятором "Компас" типов могут быть выведены.

• Как и в случае ввода, все процедуры вывода реализуются в исполняемом коде "E97" в виде подготовки и вызова соответствующих подпрограмм ПЗУ, что освобождает программиста на Паскале от необходимости знать технические детали ввода/вывода (порты, готовность и т.д.).

3.8. Процедуры и функции

Наиболее сложным, но в то же время интересным является механизм трансляции процедур, определяемых самим пользователем. Речь идет о конструкциях **procedure** и **function**, которые широко применяются в Паскале.

Процедура — это часть программы, оформленная как самостоятельная структура, к которой можно обратиться из других точек программы. Использование про-

цедур позволяет существенно сократить текст программы и сделать его более наглядным.

Часто результатом работы процедуры является единственная величина. В этом случае лучше использовать **функцию**, после выполнения которой результат подставляется в исходное выражение на место ее вызова. Таким образом, функция — это своеобразный частный случай процедуры с единственным выходным параметром, оформленный по особым правилам.

Реализация вызова процедуры или функции базируется на команде процессора *переход с возвратом*. Суть ее состоит в том, что, кроме перехода по указанному адресу, процессор автоматически запоминает в стеке текущее значение счетчика команд, который указывает ***на следующую команду***. После завершения выполнения всех расположенных в подпрограмме действий по специальной команде возврата процессор извлекает из стека предусмотрительно сохраненный там адрес и продолжает прерванное выполнение основной программы.

Описанный механизм вызова процедур легко допускает **вложенность**, т.е. возможность вызова одной процедуры из другой.

Помимо запоминания адреса возврата, стековая память при вызове процедур и функций имеет и другое важное назначение: через нее происходит передача параметров в эти структуры, а также именно в ней создаются **локальные переменные**. Так называются переменные, описанные внутри процедуры или функции; в отличие от них переменные, описанные в основной программе, принято называть **глобальными**. Можно даже сказать, что при вызове процедур в стеке формируется некоторая локальная область памяти, где содержатся параметры, адрес возврата и локальные переменные.

Понимание наиболее общих принципов вызова процедур и функций, по мнению автора, позволяет существенно лучше и глубже понять, как это все происходит. Особенно это относится к передаче входных и выходных значений, а также к создаваемым в ходе работы процедуры локальным переменным.

Для того чтобы сделать излагаемый материал более понятным и наглядным, в программе "Компас" принят целый ряд упрощений:

- в программе может быть только одна процедура или функция; ее описание обязательно должно располагаться непосредственно перед словом **begin** основной программы;
- процедура (функция) всегда имеет один параметр простого типа; он может передаваться любым из двух общепринятых способов: по ссылке (в описании есть слово **var**) или по значению (слово **var** отсутствует);
- процедура (функция) может иметь одну локальную переменную простого типа (допускается отсутствие описания локальной переменной);
- процедура (функция) может содержать только операторы присвоения и вызова стандартных процедур ввода/вывода **read** и **write**;
- как следствие предыдущего, процедура не может содержать вызов другой процедуры или функции, а также самой себя (рекурсия невозможна).

Список ограничений получился довольно внушительным, но наиболее существенные черты рассматриваемых структур все же можно изучить.

Перейдем теперь к описанию конкретных деталей трансляции процедур и функций.

Передача параметров по значению и ссылке

В классическом Паскале приняты два принципиально разных способа передачи параметров в процедуру или функцию. Первый из них носит название "передача параметра **по значению**" и состоит в записи в стек конкретных значений, которые будут являться начальными для входных параметров процедуры. Во втором случае, который принято называть "передача параметра **по ссылке**", через стек передается не само значение, а ссылка на него, т.е. ***адрес*** глобальной переменной, где это значение хранится. Естественно, что в последнем случае процедура получает возможность воздействовать на значение переменной, адрес которой передается. Иными словами, такой параметр одновременно является не только входным, но и выходным.

Рассмотрим простейший пример передачи параметра по значению. Программа

```
program proc1;
var a : integer;
procedure proc (b:integer);
begin b := 3 end;
begin
  a := 7; proc(a)
end.
```

компилируется следующим образом:

Адрес	Код	Комментарии
0000 0002	0E6D 0100	установка начального значения указателя стека SP
0004 0006	1C0D 0018	к началу основной программы (обход тела процедуры)
0008	0E70	**procedure** proc: SP → R0 (адрес локальной памяти в R0)
000A	2131	3 → R1 (значение для *b*)
000C	0103	R0 → R3 (адрес локальной памяти)
000E	2223	R3 + 3 → R3 (адрес *b*)
0010	0117	R1 → (R3) (записать 3 в переменную)
0012	0E30	*стек* → R0 (адрес возврата в R0)
0014	0E33	*стек* → R3 (удалить параметр)
0016	1C00	возврат из процедуры по адресу в R0
0018	2171	7 → R1 (значение для *a*)
001A 001C	011E <*адрес a*>	записать содержимое R1(7) в *a*
001E 0020	01E1 <*адрес a*>	*a* → R1
0022	0E21	R1 → *стек* (значение параметра)
0024 0026	9C0D 0008	вызов процедуры proc
0028	CF98	стоп

Рассмотрим итоговый исполняемый код подробнее.

Сначала происходит запись начального значения 100 в указатель стека SP (подробнее об этом см. в п. 1.4). Следующая команда обеспечивает безусловный переход к адресу 0018, где начинается основная программа, тем самым пропускаются ячейки ОЗУ с адресами 0008–0016 под процедуру proc.

Исполняемый код процедуры начинается с запоминания текущего значения SP в R0: как будет видно из дальнейшего рассмотрения, это значение является адресом небольшого участка "локальной" памяти процедуры proc, где находится адрес возврата в основную программу и параметр процедуры b.

SP + 2	параметр *b* (7, затем 3)
SP	адрес возврата (0028)

***Рис.* 3.7. Расположение информации в стеке при вызове процедуры (программа proc1)**

Это значение будет храниться в R0 и использоваться на протяжении всего времени исполнения процедуры.

Начиная с адреса 000A, расположен машинный код оператора b := 3. Сначала, следуя принятому в компиляторе "Компас" алгоритму трансляции присвоения (см. п. 3.1), значение 3, представляющее собой правую часть оператора, заносится в регистр R1. Далее следуют вычисления адреса параметра b по формуле R3 := R0 + 2 (см. *рис.* 3.7), и командой 0010 число 3 из R1 заносится по сформированному в R3 адресу.

Команды 0012–0016 обеспечивают выход из подпрограммы. Сначала освобождается локальная стековая память, причем адрес возврата считывается в R0, а значение параметра процедуры чисто формально выгружается в R3 (просто для того, чтобы освободить ненужную больше память!). Наконец, команда 0016 осуществляет безусловный переход по адресу, находящемуся в R0, тем самым выполнение процедуры завершается и основная программа продолжает прерванное выполнение.

Опишем теперь основную программу. Она состоит из двух операторов: присвоения значения переменной a и вызова процедуры. Нас будет интересовать только последний из них — команды 001E–0026. Перед вызовом процедуры значение указанного в скобках параметра a помещается в R1 и записывается в стек, после чего следует переход с возвратом на адрес 0008, соответствующий началу процедуры proc. Напомним, что при выполнении перехода с возвратом процессор всегда автоматически заносит в стек адрес следующей команды (в нашем примере это 0028). Таким образом и оказывается сформированной изображенная на *рис.* 3.7 структура информации в стеке.

Так работает передача параметра по значению. Как, видимо, уже понял читатель, суть этого метода состоит в том, что в ячейку стека, где будет располагаться параметр процедуры, помещается необходимое начальное значение. Подчеркнем, что оно совсем не обязательно должно извлекаться из переменной, а вполне может представлять собой константу или результат вычисления выражения, т.е. обращение типа proc(25) при передаче параметра по значению сработает вполне корректно.

И еще один чрезвычайно важный момент. Передаваемый данным способом параметр (в нашем случае b) существует только в момент работы процедуры, а при выходе из нее ***удаляется*** из стековой памяти (см. команду 0014). Совершенно очевидно, что такой параметр не может быть выходным, поскольку его значение после выхода из процедуры просто потеряется.

От указанного недостатка свободен второй способ передачи параметра — по ссылке. Единственное отличие в записи новой программы proc2 появится в заголовке процедуры:

```
procedure proc (var b : integer);
```

Остальной текст полностью совпадает с предыдущим примером.

Наличие служебного слова **var**, указывающее на механизм передачи параметра по ссылке, существенным образом меняет характер передаваемой в процедуру информации: вместо ***значения*** переменной в стек помещается ее ***адрес***.

SP + 2	адрес параметра *b* (адрес *a*)
SP	адрес возврата

***Рис.* 3.8. Расположение информации в стеке при вызове процедуры (программа proc2)**

Результат трансляции оператора b := 3 по сравнению с приведенной выше таблицей содержит на одну команду (0010) больше: при записи числового значения 3 адрес ОЗУ, куда надо поместить информацию, предварительно переписывается из стека в R3:

000A	2131	3 → R1 (значение для *b*)
000C	0103	R0 → R3 (адрес лок. памяти)
000E	2223	R3 + 2 → R3 (адрес *b*)
0010	0173	прочитать ссылку (адрес *a*) в R3
0012	0117	R1 → (R3) (записать 3 в переменную *a*)

И еще одно менее заметное, но важное отличие: при вызове процедуры в программе proc2 в стек заносится не значение параметра, а его адрес

0020 0022	01D1 <*адрес b*>	*адрес a* → R1

(сравните с командой в строке 001E в предыдущей программе proc1!).

В остальном коды программ proc1 и proc2 совпадают, поэтому полная таблица для proc2 здесь не приводится.

Итак, при передаче параметра по ссылке в стек помещается адрес глобальной, т.е. описанной в основной программе, переменной. Поэтому появляется возможность не только взять ее значение в качестве начального, но и ***изменить*** его. Иными словами, передаваемый по ссылке параметр может быть выходным!

Можно считать, что при использовании данного способа передачи параметром фактически является глобальная переменная. Часто говорят об ***отождествлении*** указанной при вызове процедуры переменной с используемым в процедуре формальным параметром: в нашем примере отождествляются переменные a и b.

Очевидно, что обращение `proc(25)` в случае передачи параметра по ссылке является ошибочным, поскольку передать адрес переменной при таком вызове становится невозможно.

Все сказанное выше по поводу передачи параметров в процедуру справедливо и для функции. В то же время применение функции имеет определенную особенность.

Возврат функцией результирующего значения

Главным признаком, по которому отдается предпочтение функции перед процедурой, служит наличие единственного результата. Этот результат возвращается вместо имени функции в любое выражение, в которое оно входит. Рассмотрим, например, простейшую функцию, которая удваивает передаваемый ей аргумент. Текст программы на Паскале будет иметь следующий вид:

```
program func1;
var a : integer;
function udv (x : integer) : integer;
begin udv := 2 * x end;
begin
  a := udv(4)
end.
```

Обратите внимание на следующие особенности.

В конце описания функции `udv` после двоеточия указан тип результата функции (кстати, он совсем не обязан совпадать с типом аргумента, как это получилось в нашем простейшем примере!).

Непременным условием возвращения правильного результата является наличие внутри тела функции строки, которая присваивает значение результату функции. В обсуждаемом примере это `udv := 2 * x`. Заметим, что в нашей функции других операторов нет, что, разумеется, бывает крайне редко. Напротив, синтаксис Паскаля разрешает несколько строк, определяющих результирующее значение функции: важно только следить, чтобы хотя бы одна из них при вычислении функции обязательно "сработала".

Примечание. В последней версии Паскаля — Delphi для обозначения результата функции вводится специальная переменная **Result**, которая может быть использована внутри функции наравне с остальными локальными переменными.

И еще одна особенность. Вызов функции по смыслу не является самостоятельным оператором, а служит частью какого-либо выражения. Чаще всего функция просто содержится в правой части оператора присвоения. Впрочем, современный Турбо Паскаль не очень жестко следит за соблюдением этого положения.

Обратимся теперь к результатам трансляции приведенного выше примера.

Сравнивая с разбиравшимся ранее примером процедуры, мы видим очень много общего. Поэтому целесообразно остановиться на различиях.

При входе в функцию в стеке выделяется дополнительное слово под результат (команда `0008`). При этом начальное его значение, как это принято в "Компасе", не определено. При выходе из функции результат из стека переписывается в регистр `R1`.

Адрес	Код	Комментарии
0000 0002	0E6D 0100	установка начального значения указателя стека SP
0004 0006	1C0D 001E	к началу основной программы (обход тела процедуры)
0008	0E20	**function** func1: выделить место под результат функции
000A	0E70	SP → R0 (адрес локальной памяти в R0)
000C	2121	2 → R1 (значение для умножения)
000E	0102	R0 → R2 (адрес локальной памяти)
0010	2242	R2 + 4 → R2 (адрес *x*)
0012	0561	R1 * (R2) → R1 (2 * *x*)
0014	0114	R1 → (R0) (запись в udv)
0016	0E31	результат функции → R1
0018	0E30	*стек* → R0 (адрес возврата в R0)
001A	0E33	*стек* → R3 (удалить параметр)
001C	1C00	возврат из процедуры по адресу в R0
001E	2141	4 → R1 (значение параметра)
0020	0E21	R1 → *стек* (значение параметра)
0022 0024	9C0D 0008	вызов функции udv
0026 0028	011E <*адрес a*>	запись результата функции в *a*
002A	CF98	стоп

SP + 4	параметр *x* (4)
SP + 2	адрес возврата (0026)
SP	результат функции udv (8)

Рис. 3.9. **Расположение информации в стеке при работе функции (программа `func1`)**

Примечание. Поскольку обмен со стеком всегда осуществляется ***словами***, под результат всегда отводится слово, даже если он имеет тип **char** или **boolean**. Выбор одного или двух байтов будет сделан при чтении результата из R1 после завершения исполнения функции.

Внутри функции всегда должен стоять оператор, заносящий в ячейку результата необходимое значение. В нашем примере это действие реализуется командами `000C`–`0014`, которые извлекают из стека параметр x, умножают его на 2 и заносят результат в отведенную для этой цели ячейку стека. Вычисление необходимых адресов станет более понятным, если воспользоваться *рис.* 3.9.

При выходе из функции полученный результат нельзя оставить в стеке, так как "локальная" стековая память при этом должна быть освобождена. Поэтому приходится перенести результат работы функции в регистр `R1` (команда `0016`), откуда он и будет взят основной программой (команда `0026`). Возврат результата функции через фиксированный регистр используется и в Турбо Паскале [16].

Непосредственно вызов функции ничем не отличается от описанного ранее вызова процедуры. Единствен-

ная тонкость заключается в том, что после вызова вставляется команда 0026, использующая полученное в R1 значение.

Остается обсудить еще одно свойство процедур и функций.

Локальные переменные

Еще раз напомним читателю, что переменные, описанные в основной программе, называются **глобальными**, а описанные в самой процедуре или функции — **локальными**. Например, в приводимом ниже фрагменте программы

```
program  proc3;
var  a : integer;
procedure  prim (x : integer);
var  r : integer;
begin  r := x  end;
```

переменная a является глобальной, в то время как r — переменная локальная.

Локальные переменные также создаются в стеке при входе в процедуру или функцию и освобождаются при выходе из нее. Стек при этом имеет вид:

SP + 4	параметр *x*
SP + 2	адрес возврата
SP	локальная переменная *r*

Рис. **3.10. Расположение информации в стеке при работе процедуры (программа** proc3**)**

Все это очень похоже на предыдущий пример, поэтому предоставим читателю возможность разобраться в приведенной таблице самостоятельно.

Адрес	Код	Комментарии
0008	0E20	**procedure** prim: выделить место под переменную *r*
000A	0E70	SP → R0 (адрес локальной памяти в R0)
000C	0101	R0 → R1 (адрес локальной памяти)
000E	2241	R1 + 4 → R1 (адрес *x*)
0010	0151	(R1) → R1 (значение *x*)
0012	0114	R1 → (R0) (запись в *r*)
0014	0E31	*стек* → R1 (удалить локальную переменную)
0016	0E30	*стек* → R0 (адрес возврата в R0)
0018	0E33	*стек* → R3 (удалить параметр)
001A	1C00	возврат из процедуры по адресу в R0

Наличие механизма локализации переменных является одним из преимуществ Паскаля перед большинством версий языка Бейсик, в которых все переменные являются глобальными. Благодаря этому свойству можно не заботиться о том, что используемое в процедуре или функции обозначение переменной случайно совпадет с каким-либо обозначением вне ее. При большом количестве процедур (тем более если они созданы разными людьми) написать программу, не содержащую ошибок такого рода, без локализации практически невозможно.

Σ Итак, мы рассмотрели целый ряд особенностей компиляции процедур и функций. Наиболее важными из них являются следующие:

- Процедура — это часть программы, оформленная как самостоятельная структура, к которой можно обратиться из других точек программы.
- Функция — это процедура с единственным выходным параметром, который используется особым образом.
- Реализация вызова процедуры или функции базируется на команде процессора *переход с возвратом*, которая обеспечивает возможность автоматического продолжения основной программы после завершения выполнения подпрограммы.
- Реализация процедур и функций существенным образом базируется на использовании стековой памяти: в стеке запоминается адрес программы, к которому нужно вернуться после выполнения процедуры или функции; через стек передаются входные параметры; в стеке располагаются переменные, описанные в теле процедуры или функции, а также во время выполнения функции сохраняется ее выходной параметр.
- В Паскале параметры в процедуру или функцию могут передаваться двумя способами: по значению и по ссылке. В первом случае в стек заносится указанное при вызове значение, а во втором — адрес памяти, где это значение находится, т.е. адрес переменной. Очевидно, что выходные параметры всегда должны передаваться только по ссылке. Для входных параметров допустимы оба варианта, но во избежание "побочных" эффектов изменения значений переменных лучше использовать механизм передачи по значению.
- В отличие от процедуры функция обладает свойством возвращать свой результат в то место выражения, где она используется. Иными словами, Паскаль как бы подставляет полученное значение вместо вызова функции. Функция обязательно должна содержать хотя бы один оператор присвоения, в котором слева стоит ее имя: такая символическая запись позволяет определить возвращаемое функцией значение. В процессе работы функции ее результат находится в стеке, а при выходе переписывается в определенный регистр процессора (в "Компасе" используется R1).
- Помимо объявленных в основной программе глобальных переменных, в процедуре или функции можно описать собственные локальные переменные. Они создаются в стеке при входе в процедуру и уничтожаются при выходе. Наличие локальных переменных позволяет программисту иметь рабочие переменные, никак не связанные и не оказывающие никакого влияния на внешние (глобальные) переменные. Отметим, что передаваемый по значению параметр тоже можно

считать локальной переменной, которая создается и инициализируется указанным значением несколько раньше — при вызове процедуры, а не при входе в нее.

3.9. Выбор начального значения указателя стека при компиляции

Как подробно разбиралось в п. 1.4, компилятор содержит некоторый "свободный" параметр — значение HiMem0, выше которого располагается служебная область. Теперь, когда мы уже в деталях знаем, как используется стек в компиляторе "Компас", можно попробовать подобрать подходящее значение этого параметра.

Для этого необходимо составить программу, "максимально загружающую" стек. Такая программа должна содержать функцию с локальной переменной (как следует из изложенного в п. 3.8, стек в этом случае состоит из четырех слов: "параметр", "адрес возврата", "результат функции" и "локальная переменная"). В программе обязательно надо использовать цикл **for**, требующий для своей реализации еще 2 "этажа" стека (см. п. 3.5). Кроме того, очень полезно, чтобы внутри **function** стоял вызов стандартной процедуры печати целого числа, которая использует несколько подпрограмм ПЗУ, что также активно задействует стек. Всем перечисленным требованиям удовлетворяет приведенная ниже программа:

```
program calcfunc;
var  x, y : integer;
function func(k : integer) : integer;
var r : integer;
begin r := sqr(k); func := 20 * r; write(k)
end;
begin for x := 1 to 5 do
        begin y := func(x); writeln(y)
        end;
end.
```

Эксперименты показали, что для данной программы максимально допустимое значение HiMem0, при котором стек занимает все отведенные под него ячейки, равняется 00E0. Любая другая программа должна использовать стек менее интенсивно, а значит, тем более удовлетворится выбранным значением. Именно данное значение, которое позволяет получить максимальный объем памяти под программу и минимально достаточный под стек (см. *рис.* 1.1), и было выбрано для компилятора "Компас".

3.10. К вопросу об оптимальности компилятора

Изучаемый нами компилятор обрабатывает операторы достаточно простым по логике способом, используя небольшой набор "готовых рецептов" для каждого оператора или даже его частей. Такое стандартное решение легко реализуется, зато оно далеко не всегда является оптимальным: платой за универсальность алгоритма всегда является его некоторая нерациональность.

Поясним это на простом примере трансляции оператора присвоения `w := 20`. Согласно описанному в п. 3.1 алгоритму, компилятор составит программу, которая сначала "положит" константу 20 в R1, а затем запишет значение этого регистра в ячейку, где хранится переменная w. Для этого потребуется 4 машинных слова, т.е. 8 байт. В то же время очевидно, что можно константу 20 ***сразу*** занести в ячейку ОЗУ, минуя R1. Для этого хватит команды, содержащей всего 3 слова (6 байт)! Причина нерациональности очевидна — ради универсальности алгоритма компиляции каждый этап операции транслируется самостоятельно, без учета содержания предыдущих.

Аналогичным образом нерационально будет кодироваться последовательность операторов `x := z; y := x;` или `x := z; writeln(x);` При компиляции первого из них "Компас" запланирует копирование значения z в R1, а затем запись его в x. После этого, "не отдавая себе отчета" в том, что в R1 уже находится значение x, компилятор снова организует чтение того же самого значения в R1! Очевидно, что данная нерациональность вызвана тем, что каждый оператор рассматривается ***независимо от других***.

Не менее впечатляет нас "недогадливость" компилятора при обработке, скажем, оператора `q := 2 + 7`, когда компилятор честно составляет программу суммирования двух констант. Между прочим, Турбо Паскаль содержит специальные встроенные методы оптимизации, способные упростить рассматриваемый пример до `q := 9` [16].

Обсуждение некоторых интересных примеров оптимизации результирующего кода можно найти в последней главе книги [13].

Есть и еще один (сознательно введенный в "Компас") источник нерациональности — использование абсолютных переходов, когда в команде явно хранится адрес, на который необходимо перейти. Абсолютные переходы ***значительно более наглядны***, чем относительные, но зато занимают в памяти ***вдвое больше места***.

Таким образом, в основу компилятора "Компас" положены простота и наглядность, а не оптимальность выходного кода — не следует требовать от демонстрационного компилятора слишком много!

Более того, если вы, уважаемый читатель, уже замечаете нерациональность составленного "Компасом" кода, автор только рад этому: значит, вы уже готовы самостоятельно, без помощи компилятора, писать программу в машинных кодах! По-моему, это просто замечательно!

Окончание следует

Развитие учащихся на уроках информатики

Е.П. Паклина,
г. Бердск, Новосибирская область

Кто не знает, в какую гавань он плывет, для того нет попутного ветра.
Сенека

Знания — дети удивления и любопытства.
Луи де Бройль

Развитие познавательного интереса к предмету

Мотивация учебно-познавательной деятельности — одна из важнейших задач для педагога. Она не возникает самопроизвольно, и ее создание вполне можно сравнить с искусством. Большим подспорьем в этом являются принципы и методы развивающего обучения по Л.В. Занкову, которые можно сформулировать так:

- взаимосвязь обучения, воспитания и развития;
- целенаправленность развития;
- проблемность (коллизии);
- индивидуализация и дифференциация;
- осознание школьниками процесса обучения (что я узнал нового?);
- создание положительного эмоционального фона для учебной работы.

Уроки информатики, как никакие другие, позволяют активизировать творческое начало, основываясь на развитии:

- познавательного интереса к предмету;
- различного вида памяти;
- основ коммуникативного общения;
- приемов умственной деятельности (обобщение, синтез, анализ, абстракция, сравнение, выделение главного).

Хотелось бы поделиться некоторыми приемами работы, позволяющими даже в обычных классах, занимающихся по 110—120-часовой программе, использовать информатику для всестороннего развития учащихся.

Успех урока обуславливается не только четкой формулировкой цели занятия и объяснением важности, в том числе и практической, изучаемого материала, но прежде всего подбором заданий. Я стараюсь подбирать доступные и увлекательные задачи, например: "Оформление визитной карточки" (задание выполняется в текстовом редакторе), "Подсчет стоимости обеда в кафе" (задание выполняется в электронной таблице), "Разработка прайслиста торговой фирмы" (задание выполняется в электронной таблице), "Хобби учеников вашего класса" (задание выполняется с использованием СУБД), создание статических и динамических рисунков средствами языка программирования (мы используем QBasic) — и т.д.

Кроме того, учениками всегда с интересом воспринимаются сведения из истории развития вычислительной техники. Краткие исторические экскурсы придают уроку особую яркость и эмоциональность.

Одной из важных форм работы являются самостоятельные домашние задания, на которые в нашей школе отводится 8 часов. Вот некоторые из тем таких заданий:

- создание и оформление кроссворда по информатике;
- разработка задачи по теме "Электронные таблицы MS WORKS";
- разработка задачи по теме "Информационно-поисковая система MS WORKS";
- разработка и оформление алгоритма по его блок-схеме.

Практикуются и небольшие творческие задания на дом, выполнение которых требует обязательной работы с литературой. Вот две темы таких работ:

- Как работает винчестер и почему он так называется?
- Почему компакт-диск называется CD-ROM? Как на нем записывается информация? Чем отличаются диски CD-R от CD-RW?

Достаточно эффектным является прием "Трудная задача". Вот в чем он заключается: на стенде в кабинете висят конверты по уже изученным и изучаемым темам. Любой желающий может взять из них задачу и попробовать ее решить. Такая попытка никак не контролируется, и лишь в том случае, когда задача решена верно в письменном виде с объяснением решения, ученик получает отличную отметку.

Нельзя забывать и о создании "ситуации успеха", о минимизации стрессовых ситуаций, когда контрольные работы анонсируются, а ученикам всегда дается шанс написать их еще раз.

Ну и, конечно, необходим индивидуальный подход к ученику. При этом необходимо помнить и о психологической этике общения. Например, недопустимо в разговоре с учащимися сравнивать их друг с другом и т.п.

Развитие памяти

Хочется напомнить хорошо известные в практической психологии основные принципы, на которых основывается развитие памяти:

1) лучше всего запоминается то, что связано с преодолением какого-либо препятствия, затруднения;

2) материал, полученный самостоятельно, в процессе активной деятельности, запоминается лучше полученного пассивным путем.

Во многом способствуют лучшему запоминанию и смысловые опорные пункты — примеры, эмоционально окрашенные факты, сравнения, названия, имена, эпитеты.

Приведем пример использования первого приема при изучении темы "Переменная. Имя и значение переменной". В процессе объяснения нового материала на уроке-беседе полезно начертить схему оперативной памяти и в дальнейшем рассматривать ее содержимое после каждой операции:

```
LET A = 20
LET T = 10
LET B = 6
```

Ячейки оперативной памяти распределены следующим образом:

	20		10		6	
	A		T		B	

Теперь рассмотрим, что произойдет после выполнения оператора присваивания:

```
LET T = T + 1
```

Покажем состояние ячеек:

	20		11		6	
	A		T		B	

Прежнее значение 10 в ячейке T безвозвратно пропало. Необходимо подчеркнуть важность этого факта.

Далее рассматривается следующая задача:

Значение переменной A равно 20, значение переменной B равно 6. Составить программу, в результате которой переменные A и B обменяются своими значениями.

Как правило, основная масса учащихся оживленно обсуждает ход решения задачи, и схема обмена данными кажется очень простой:

В процессе обсуждения чаще всего предлагается следующий вариант программы:

```
10 CLS
20 A = 20
30 B = 6
40 A = B
50 B = A
60 PRINT "A=";A
70 PRINT "B=";B
80 END
```

Собственно, к этому моменту учащиеся обладают всеми необходимыми знаниями для решения подобной задачи, но было бы методически неверным выложить им правильное решение "на блюдечке". Запускаем программу и получаем результат:

```
A = 6
B = 6
```

Теперь самое время задать следующие вопросы:

1. Что нужно получить на самом деле?
2. Что же произошло на самом деле?
3. На каком этапе выполнения программы это произошло?
4. Что необходимо делать в жизни, чтобы не потерять что-то очень важное? (Ответы: положить в банк, спрятать, сберечь, ...или — сохранить!)
5. Где можно сохранить значения переменных? Конечно же, в другой переменной.

Далее не составляет труда подправить программу.

Быть может, этот пример и покажется кому-то излишне простым, но не будем забывать, что понятие переменной является фундаментальным и вовсе не очевидным. Простая на первый взгляд программа позволяет объяснить термины "переменная", "имя переменной", "значение переменной".

А нельзя ли обойтись без дополнительной переменной? Получив это задание, ученики, как правило, затрудняются с ответом. Дальнейшая совместная работа учителя и учеников может быть проиллюстрирована следующими утверждениями:

1. Мы можем использовать только переменные A и B и их значения.
2. Мы, очевидно, не можем переменной A сразу присвоить значение переменной B и наоборот.

После обсуждения вполне можно заметить, что для выполнения задачи самый первый оператор должен записывать в переменную A какую-то комбинацию значений переменных A и B. А еще через некоторое время можно предложить в качестве такой комбинации использовать сумму A + B. И вполне можно считать урок успешным, если учащиеся практически самостоятельно получат программу:

```
10 CLS
20 A = 20
30 B = 6
40 A = A + B
50 B = A - B
60 A = A - B
70 PRINT "A=";A
80 PRINT "B=";B
90 END
```

Развитие коммуникативного общения

Для развития коммуникативного общения можно применять следующие приемы:

1. Работа по методу КСО (коллективного способа обучения) в парах сменного состава. Мы используем этот метод при изучении основ программирования на языке QBasic. При изучении особо важных тем, например "Ветвления в алгоритмах", "Организация цикла", "Организация одномерного массива и работа с его элементами", необходимо получить прочные навыки в распознавании задач и их решении. Весьма продуктивным приемом является создание пакетов задач по основным темам. Работая с пакетом, ученики решают задачи, обсуждают процесс решения и результат, разбирают ошибки и дают пояснения друг другу.

2. Прием "Консультант". На уроках фронтальной работы с классом или при разборе контрольных работ один из учеников (с его согласия) назначается консультантом. Причем он может быть и не самым сильным, но хорошо знающим изучаемую тему. Функция консультанта — наблюдать за процессом решения задач, отвечать на вопросы, разъяснять трудные места, контролировать результаты нескольких человек (до половины группы). Учитель по возможности не вмешивается в работу группы. Возможен вариант, когда в конце урока работу консультанта оценивают подопечные.

Развитие приемов умственной деятельности

Под приемами умственной деятельности обычно подразумевают абстракцию, обобщение, синтез, анализ, сравнение и т.п.

Приведу некоторые примеры, которые показывают, как можно развивать у учеников навыки использования конкретных приемов.

Сравнение

Составление аналогичных задач;

Решение задач частично-поисковым методом. Ученики при этом сравнивают ситуации в исходной задаче с уже известными похожими ситуациями ранее и отыскивают причину проблемы (хотя пока и не знают, как ее устранить). Применять этот метод можно, например, при:

— работе с электронной таблицей, изучая понятие "абсолютная адресация";

— изучении понятий "переменная", "имя переменной", "значение переменной";

— изучении оператора IF;

— изучении логических операций;

— изучении оператора FOR;

— работе с одномерными массивами.

Решение специально подобранных задач и заданий, например:

Получение случайного числа на интервале (0, 1) осуществляется по формуле

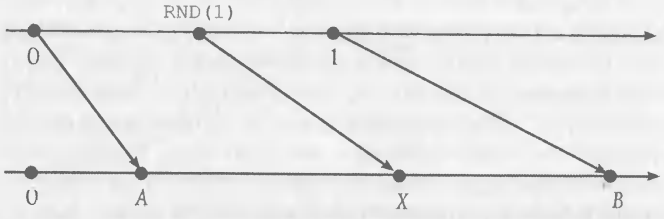
```
X = RND(1)                    (1)
```

Получение случайного числа на произвольном интервале (*A*, *B*) осуществляется по формуле

```
X = RND(1)*(B - A) + A        (2)
```

Вопрос задания: объясните, как из формулы (1) получается формула (2)?

Вполне возможно, что решать это задание придется всем классом, и тогда очень полезной окажется следующая схема:



Выделение главного

Выделение главного — это важный способ фиксации учебного материала в долговременной памяти.

При составлении блок-схем, задач в электронных таблицах, задач для СУБД необходимо добиваться четких, точных названий и заголовков.

Очень важно составлять предварительный план решения, выделяющий основные его этапы.

На основе предварительно созданного плана производится разработка блок-схем методом последовательной детализации.

Анализ — синтез

Напомним, что анализ — это мысленное или фактическое разложение целого на составные части, синтез — это познание предмета как единого целого его частей.

Наиболее характерен прием при построении блок-схемы любого алгоритма. По существу, это выделение отдельных блоков для решения задачи и дальнейшее осмысление решения в целом через эти блоки.

Фактически этот же прием используется и при решении задач в парах сменного состава.

Хороши и такие задания: найти ошибки и написать правильный вариант программы:

```
10 CLS
20 INRUT X=;
30 Y = A^ ^2 - 2
40 PRIN "Y"
50 ENT
```

Абстракция

Абстракция — это процесс вычленения существенного (для решения данной задачи) элемента в заданном материале. Она проявляется, в частности, при:

а) разработке блок-схем в задачах;

б) распознавании блок-схем операторов IF и FOR в общей блок-схеме задачи;

в) применении рисунков, схем при решении задач;

г) решении задач особого рода, например: "Подсчитать количество слов в символьной строке". Здесь важно свести дело к подсчету одиночных пробелов или групп пробелов.

Обобщение

Обобщение — это нахождение общего в заданных предметах или явлениях.

Этот прием часто применяется, например, при изучении языка QBasic — при решении цепочки схожих задач, например:

Пример 1

Задача № 1. Ввести с клавиатуры значения переменных *A* и *B*. Сообщить, равны ли они, или сообщить, какая переменная больше, а какая — меньше.

Задача № 2. Ввести с клавиатуры значение *X*. Вычислить и напечатать значение функции:

$$y = \begin{cases} \sqrt{x}, & \text{при } x > 4 \\ x + 1, & \text{при } x < 1 \\ x^2, & \text{при } 1 \le x \le 4 \end{cases}$$

Пример 2

Задача № 1. Вычислить и напечатать сумму 10 произвольных чисел, вводимых с клавиатуры. Используйте в решении оператор цикла FOR.

Задача № 2. Вычислить и напечатать произведение 10 произвольных чисел, вводимых с клавиатуры.

Пример 3

Задача № 1. Для x, вводимого с клавиатуры, найти и напечатать результат $x + x^2 + x^3 + x^4 + x^5 + x^6$.

Задача № 2. Для x, вводимого с клавиатуры, найти и напечатать результат

$$S = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9} - \frac{x^{11}}{11} + \frac{x^{13}}{13} - \frac{x^{15}}{15} + \frac{x^{17}}{17}$$

(Идея здесь проста: после представления выражения в виде

$$S = \left(x + \frac{x^5}{5} + \frac{x^9}{9} + \frac{x^{13}}{13} + \frac{x^{17}}{17}\right) - \left(\frac{x^3}{3} + \frac{x^7}{7} + \frac{x^{11}}{111} + \frac{x^{15}}{115}\right)$$

можно использовать алгоритм, аналогичный задаче № 1.)

Простая программа для решения квадратных уравнений на языке Visual Basic

В.Ефимов,
школа № 610, Москва

От редакции

Всем нам по собственному опыту хорошо известно, что при освоении нового программного продукта, в частности новой среды программирования, важно сделать первый шаг. В последнем случае, например, важно написать и, что немаловажно, заставить работать первую, пусть даже и очень простую, программу. Возможно, именно по этой причине с программ "Hello World!" (или "Здравствуй, мир!", кому как больше нравится) так часто начинаются учебники программирования. "Начните с простого" – новая рубрика нашей газеты. В ней мы будем публиковать материалы, цель которых – помочь вам сделать первый шаг при освоении того или иного продукта. Интересно отметить, что первая статья в новую рубрику была предложена учащимся одной из московских школ.

В этой статье описывается разработка простой программы для решения квадратных уравнений на языке Visual Basic. Предполагается, что у читателя уже установлена среда Visual Basic, собственно процесс установки здесь не описывается, поскольку он практически никогда не вызывает проблем.

После запуска Visual Basic предлагается выбрать тип проекта (см. *рис.* 1). Речь идет о типе создаваемого исполняемого модуля. Мы выберем первый из предлагаемых вариантов — **Standard EXE**. На рабочем поле появится так называемая "форма" — окно будущей программы. Пока она пустая. Нам требуется расположить на ней необходимые "элементы управления" — поля ввода, поля вывода, кнопки и т.д., в общем, мы можем использовать стандартный "джентльменский набор" элементов управления Windows. Для помещения элемента управления на форму требуется выбрать его на панели инструментов (на *рис.* 2 она расположена слева) и "нарисовать" элемент на форме. Нам потребуются три объекта **TextBox**, три объекта **Label** (в них будет осуществляться ввод и вывод информации) и кнопка **CommandButton**. Элементы управления, размещаемые на форме, получают некоторые имена "по умолчанию". В такой небольшой программе их можно было бы оставить и такими, но все же много проще разбираться в коде, когда объекты имеют содержательные имена. Имя объекта можно задать (изменить) на панели свойств объектов (на *рис.* 2 эта панель — средняя справа). В первой же строке на этой панели определяется

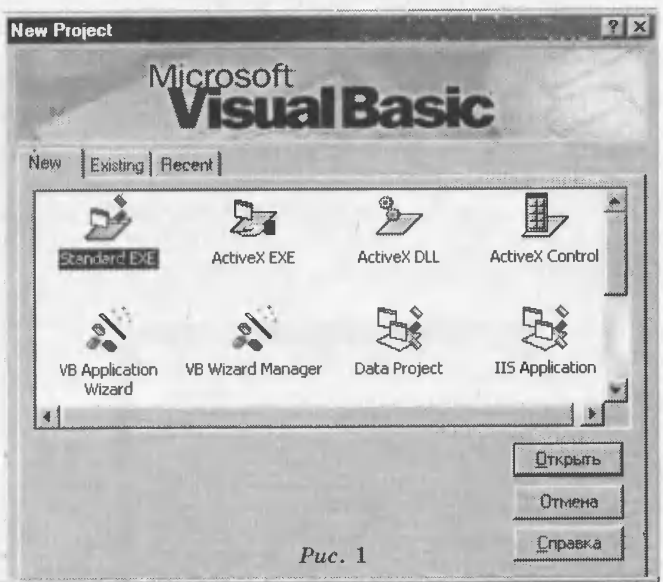


Рис. 1

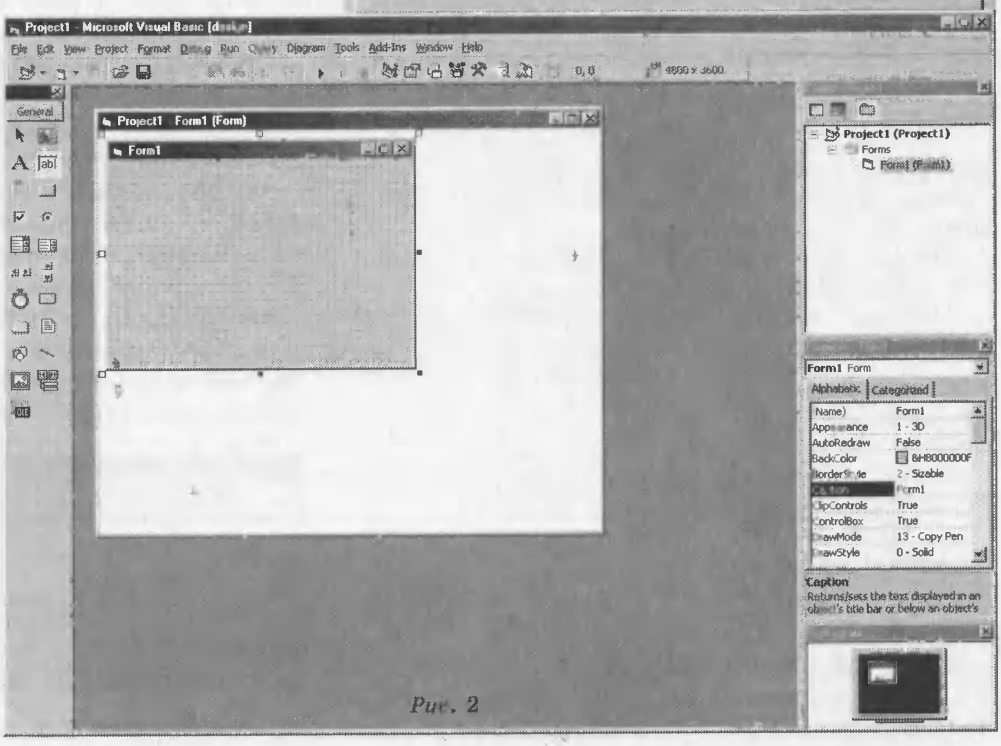


Рис. 2

имя объекта. Мы назовем три поля ввода коэффициентов TextA, TextB и TextC, поля вывода значений корней Label1 и Label2, поле для вывода сообщения Status и кнопку Calculation, при нажатии на которую будут производиться вычисления (см. *рис.* 3).

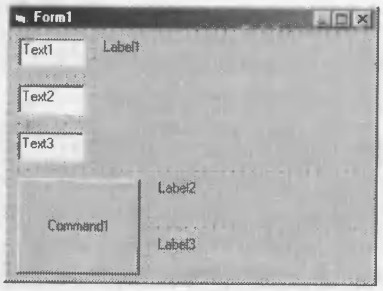


Рис. 3

Двойным щелчком по кнопке, которая теперь называется Calculation, откроем окно связанного с ней программного кода. Этот код будет вызываться при нажатии на данную кнопку. В начале этот программный код "пустой" — мы видим лишь заготовку подпрограммы Calculation_Click. Имя этой подпрограммы состоит из двух частей — имени объекта (кнопки) и названия действия, в ответ на которое данная подпрограмма будет вызвана (см. *рис.* 4). Далее приведен код подпрограммы Calculation_Click. Мы привели некоторые комментарии, которые показались нам полезными.

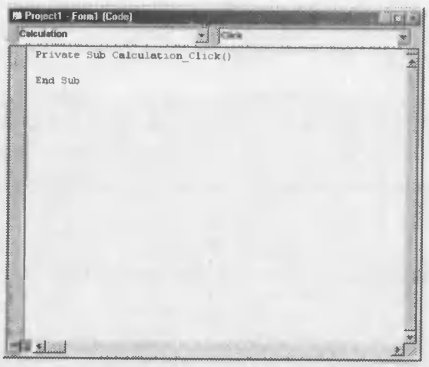


Рис. 4

```
Private Sub Calculation_Click()
' Объявление переменных
Dim a, b, c, D, X1, X2
' Присваиваем значения коэффициентам
a = TextA.Text
' С этими присваиваниями не все так просто, ниже
' приведены соответствующие пояснения
b = TextB.Text
c = TextC.Text
If a = 0 And b <> 0 Then
   X1 = -c / b
   ' Вывод значения в объект Label1
   Label1.Caption = X1
   ' Вывод значения в объект Label2
   Label2.Caption = ""
   Status.Caption = "Было введено линейное
                    уравнение." & _
                   "Один корень."
  ' Выше мы специально показали,
   как продолжать длинную текстовую строку
ElseIf a = 0 And b = 0 Then
   Label1.Caption = ""
   Label2.Caption = ""
   Status.Caption = "Уравнение не имеет смысла"
Else
' Вычисление дискриминанта
   D = b * b - 4 * a * c
   If D > 0 Then
    X1 = (-b + Sqr(D)) / (2 * a)
    X2 = (-b - Sqr(D)) / (2 * a)
    Label1.Caption = X1
    Label2.Caption = X2
    Status.Caption = "Было введено квадратное
                      уравнение." & _
                     " D > 0. Два корня."
   Else If D = 0 Then
    X1 = -b / 2 * a
    Label1.Caption = X1
    Label2.Caption = ""
    Status.Caption = "Было введено квадратное
                      уравнение." & _
                     " D = 0. Два одинаковых корня"
   Else
    Label1.Caption = ""
    Label2.Caption = ""
    Status.Caption = "Было введено квадратное
                      уравнение" & _
                      " D < 0. Корней нет"

   End If
End If
End Sub
```

Отметим, что если в поля TextA, TextB, TextC будут введены не числовые значения, а мы никак не контролируем этот факт, то при выполнении вычислений возникнет ошибка времени выполнения. Мало того, "по умолчанию" все определенные нами элементы управления обозначены на форме не информативными названиями типа **Text1**, **Text2**, **Text3**, **Label1**, **Label2**, **Label3** и **Command1**. То, что мы изменили реальные названия объектов, на форме не отразилось: название объекта в программе может (и чаще всего должно) отличаться от его названия на форме. Для изменения названий и объектов следует изменить соответствующие свойства (снова на панели свойств объектов). Для текстовых полей ввода следует изменить значение свойства **Text**, а для надписей и кнопки — значение свойства **Caption**. Результат изменения соответствующих свойств мы немедленно увидим на форме (сравните *рис.* 3 и *рис.* 5).

Если вы все сделали правильно, то можете немедленно проверить работу программы из среды Visual Basic. Для этого надо выбрать пункт "Run" главного меню. Программа должна заработать. На *рис.* 6 показан результат работы программы. Для создания автономного исполняемого модуля (EXE-файла) следует выбрать в главном меню пункт "Файл", а в нем — пункт "Make".

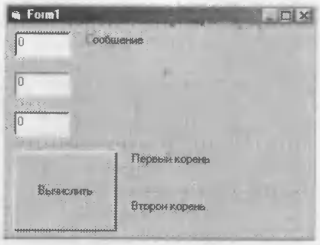


Рис. 5

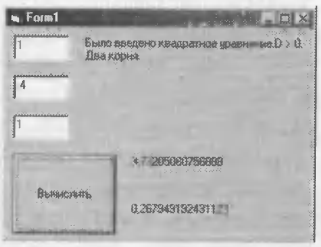


Рис. 6

Даже если вы не успели подписаться с января 2002 г., еще не поздно сделать это с февраля!

Как и в прошлом году, мы предложим подписчикам **две серии специальных тематических выпусков — зимнюю, январскую, и летний трехмесячный марафон, состоящий из 12 номеров**. Летняя серия получит традиционное название "Жаркое лето-2002", а серия январских номеров будет называться "Снег идет".

Новый проект, который стартовал в октябре (№ 37/2001), связан с нашей новой рубрикой **"Страницы повышения квалификации"**. Учителю информатики как никому другому необходимо постоянно поддерживать свою "форму". Но далеко не всегда есть возможность пройти *хорошие* курсы повышения квалификации. "Информатика" предлагает подписчикам несколько лекционных курсов для повышения квалификации, которые публикуются на страницах газеты. Судя по отзывам на первые публикации, эти материалы не просто интересны, но очень полезны нашим читателям.

В первом полугодии 2002 года начнется публикация материалов, которые все мы давно ждем. Это **задачник по Excel**, который готовится специально для "Информатики" (в этом и предыдущем номерах мы представили несколько фрагментов нового задачника).

Еще одна **новая рубрика**, которая появится во втором полугодии, — **"Начните с простого"**. Статьи этой рубрики, написанные простым языком, ориентированы на тех, кто начинает знакомство с вопросом "с нуля". В них вы познакомитесь с множеством интересных и якобы сложных (ведь дело в том, как объяснить!) вопросов: Perl, PHP, Visual Basic, Delphi, SQL и многими другими.

План публикаций лекций курса "Современные педагогические технологии и частные методики обучения информатике" на "Страницах повышения квалификации".

Лекции читает И.Н. Фалина

Номер лекции	Номер газеты
1	37/2001
2	39/2001
3	41/2001
4	43/2001
5	45/2001
6	47/2001
7	5/2002
8	7/2002
9	9/2002
10	11/2002
11	13/2002
12	15/2002

Уважаемые читатели, пожалуйста, обратите внимание на то, что лекции 7—12 будут опубликованы в первом полугодии 2002 г.

План публикаций лекций курса "Олимпиады по информатике. Пути к вершине" на "Страницах повышения квалификации".

Лекции читает Е.В. Андреева

Номер лекции	Номер газеты
1	38/2001
2	40/2001
3	42/2001
4	44/2001
5	46/2001
6	48/2001
7	6/2002
8	8/2002
9	10/2002
10	12/2002
11	14/2002
12	16/2002

Уважаемые читатели, пожалуйста, обратите внимание на то, что лекции 7—12 будут опубликованы в первом полугодии 2002 г.

План публикаций лекций курса "Введение в специальность "учитель информатики" на "Страницах повышения квалификации".

Лекции читает А.Г. Гейн

Номер лекции	Номер газеты
1	6/2002
2	8/2002
3	10/2002
4	12/2002
5	14/2002
6	16/2002

Уважаемые читатели, пожалуйста, обратите внимание на то, что все лекции будут опубликованы в первом полугодии 2002 г.

Уважаемые читатели! Газета "Информатика" распространяется только по подписке, поэтому не забудьте подписаться на нашу газету.

Ф. СП-1

Министерство связи Российской Федерации "Роспечать"

АБОНЕМЕНТ на газету **32291** (индекс издания)

ИНФОРМАТИКА

наименование издания | Количество комплектов

на 20__ год по месяцам

1	2	3	4	5	6	7	8	9	10	11	12

Куда (почтовый индекс) (адрес)

Кому (фамилия, инициалы)

ДОСТАВОЧНАЯ КАРТОЧКА

ПВ | место | литер | на газету **32291** (индекс издания)

ИНФОРМАТИКА

(наименование издания)

Стоимость: подписки _____ руб. | переадресовки _____ руб. | Количество комплектов

на 20__ по месяцам

1	2	3	4	5	6	7	8	9	10	11	12

Куда (почтовый индекс) (адрес)

Кому (фамилия, инициалы)

Читайте в номере

Гл. редактор
С.Л. Островский
Зам. гл. редактора
А.И. Сенокосов
Редакция:
Е.В. Андреева
Н.Л. Беленькая
Л.Н. Картвелишвили
Н.П. Медведева
Дизайн и верстка:
Н.И. Пронская
Корректоры:
Е.Л. Володина,
С.М. Подберезина

выходит четыре раза в месяц
При перепечатке ссылка на ИНФОРМАТИКУ обязательна, рукописи не возвращаются

Адрес редакции и издателя:
121165, Киевская, 24
тел. 249-48-96
Отдел рекламы
тел. 249-98-70

Учредитель: ООО "Чистые пруды"

Зарегистрировано в Министерстве РФ по делам печати. ПИ № 77-7230 от 12.04.2001
Отпечатано в ОИД "Медиа-Пресса',
125993, ГСП-3, Москва, А-40, ул. "Правды", 24.
Тираж 6000 экз.
Срок подписания в печать по графику 28.11.2001.
Номер подписан 28.11.2001.
Заказ № 17873
Цена свободная

ИНДЕКС ПОДПИСКИ
для индивидуальных подписчиков 32291
комплекта изданий 32744

Тел.: (095)249-31-38, 249-33-86. Факс (095)249-31-84

Internet: inf@1september.ru
WWW: http://www.1september.ru

Главный редактор комплекта изданий А.С. Соловейчик

Английский язык (Е.В. Громушкина), индекс подписки — 32025; **Библиотека в школе** (О.К. Громова), индекс подписки — 33376; **Биология** (Н.Г. Иванова), индекс подписки — 32026; **Воскресная школа** (монах Киприан (Ященко), индекс подписки — 32742; **География** (О.Н. Коротова), индекс подписки — 32027; **Здоровье детей** (А.У. Лекманов), индекс подписки — 32033; **Информатика** (С.Л. Островский), индекс подписки — 32291; **Искусство** (Н.Х. Исмаилова), индекс подписки — 32584; **История** (А.Ю. Головатенко), индекс подписки — 32028; **Литература** (Г.Г. Красухин), индекс подписки — 32029; **Математика** (И.Л. Соловейчик), индекс подписки — 32030; **Начальная школа** (М.В. Соловейчик), индекс подписки — 32031; **Немецкий язык** (М.Д. Бузоева), индекс подписки — 32292; **Русский язык** (Л.А. Гончар), индекс подписки — 32383; **Спорт в школе** (Н.В. Школьникова), индекс подписки — 32384; **Управление школой** (А.И. Адамский), индекс подписки — 32652; **Физика** (Н.Д. Козлова), индекс подписки — 32032; **Французский язык** (Г.А. Чесновицкая), индекс подписки — 33371; **Химия** (О.Г. Блохина), индекс подписки — 32034; **Школьный психолог** (М.Н. Сартан), индекс подписки — 32898.