№ 10 (203) март 1999

еженедельное
ПРИЛОЖЕНИЕ К ГАЗЕТЕ
«ПЕРВОЕ СЕНТЯБРЯ»

ИНФОРМАТИК

Электронные версии газеты "Первое сентября" и приложений http://www.1september.ru

В Москве, в Сокольниках, прошла очередная выставка "Школа". На этот раз "99". (Заметьте, что и здесь мы встречаем проблему 2000 года: на следующий год организаторы или должны будут назвать выставку "Школа-2000", или, по аналогии с нынешней, "Школа-00". Думаем, что все же разумно выбрать первый вариант, ибо в противном случае дело дойдет и до "Школы-02", "Школы-03" и т.д.)

Если же серьезно, то выставка оставила крайне противоречивое впечатление. Во-первых, ее, по-видимому, посетило довольно мало учителей. Возможно, у организаторов другие сведения, но на "взгляд со стенда", а "Первое сентября" впервые имело свой большой стенд в самом центре выставочного зала, основные посетители выставки — административные работники и представители фирм, так или иначе связанных с образованием. "Массовку" обеспечили студенты столичных педвузов.

Во-вторых, то, что на выставке учителей было немного, огорчает, так как на стендах было что посмотреть. Если попробовать оценить содержание выставки по привычной пятибалльной шкале, то с некоторой скидкой на нынешнюю ситуацию в образовании вполне можно поставить четверку. Особенно приятно было видеть на выставке стенды школ с работами учителей и учеников. Как знать, может быть, через несколько лет именно такие экспонаты и привлекут на выставку существенно больше народу.

"Информатика", как, кстати, и все приложения к "Первому сентября", провела на выставке встречу с читателями. Народу было немного. Ну что же, может быть, встретимся через год?

На снимке: встреча с читателями "Информатики". Зам. гл. ред. С.Л. Островский и наш постоянный автор Д.М. Златопольский.

МОДЕРНИЗАЦИЯ КОМПЬЮТЕРНОГО КЛАССА

Сжатый диск в Windows 95

В.М. НЕЧАЕВ

Продолжение. См. № 14, 18, 20, 33, 40, 47/98

Подходит к завершению длинная серия статей, начатая более года назад и посвященная одной теме — приведению школьного компьютерного класса в современное по возможности состояние. Были рассмотрены вопросы наращивания оперативной памяти, подключения CD-ROM, установки операционной системы Windows 95, монтажа и настройки локальной сети и, наконец, ограничения прав пользователя в перенастройке компьютера.

Ограничение прав было рассмотрено в применении к отдельному рабочему месту и никак не затрагивало сеть. Такое решение, хотя вполне и приемлемое, все-таки оставляет какое-то чувство неудовлетворенности в душе. В самом деле, иметь в классе локальную сеть — и ходить настраивать машины от одной к другой? Как-то несолидно. Желательно было бы делать это централизованно, с учительского компьютера. Значит, надо учиться администрировать сеть, использовать хотя и не очень богатые, но все же имеющиеся в системе средства управления. И вообще сеть — это такое важное дело, так она сейчас актуальна со всеми своими применениями, среди которых и электронная почта, и Интернет, и совместная работа над общими документами, что я задумал начать новый цикл статей под общим заголовком "Работа в локальной сети на платформе Windows 95".

Но это уже в следующем выпуске рубрики. Даже и не в следующем, а через один, ближе к лету. Дело в том, что в соответствии с намеченным планом сначала в двух ближайших выпусках будет закончена нынешняя серия рассказом об установке на компьютер программных пакетов. Пишу я об этом, как обычно, задним числом; на самом деле я давно уже установил через сеть на свои винчестеры

Продолжение на с. 13

Дорогие читатели!

В первом номере этого года мы опубликовали анкету, в которой попросили вас высказать пожелания, касающиеся содержания и структуры газеты. Нам пришло большое количество писем из разных, в том числе и очень далеких, регионов России. Мы благодарим всех, кто нашел время написать нам. Как всегда, среди читателей, приславших заполненные купоны, мы разыграли небольшие призы, которые в этот раз предоставили издательство "Бином" и российская антивирусная фирма "ДиалогНаука".

Мы поздравляем:

Каюшкину Л.А. (г. Ишимбай, Башкортостан)
Кузина Ю.И. (г. Липки, Тульская область)
Локтионову Г.В. (с. Верхний Любаж, Фатежский район, Курская область)
Матушкина Г.В. (с. Васильевское, Немской район, Кировская область)
Похлебаеву Л.Н. (п. Алексеевск, Саха, Якутия).
Ржаникову Э.П. (п. Юрт, Тайшетский район, Иркутская область)

В следующем номере будет опубликована очередная анкета.

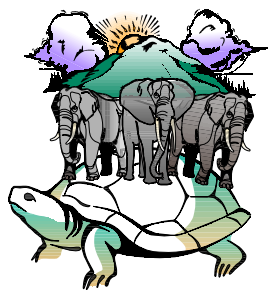
Мы будем благодарны издательствам и компьютерным фирмам, которые пожелают предоставить призы нашим читателям.

НАШИ ДЕТИ БУДУТ ЖИТЬ В XXI ВЕКЕ

12 лекций о том, для чего нужен школьный курс информатики и как его преподавать

Лекции 7—8, 9

А.Г. КУШНИРЕНКО, Г.В. ЛЕБЕДЕВ



Об основных понятиях, идеях и целях школьного курса информатики "по учебнику" *А.Г. Кушниренко, Г.В. Лебедева, Р.А. Свореня* "Основы информатики и вычислительной техники" (М.: Просвещение, 1990, 1991, 1993, 1996). Дается также ряд практических советов, предлагаются соответствующие методические приемы.

Авторы надеются, что материал окажется полезным для учителей и методистов, использующих указанный учебник, а также для тех, кто желает сравнить разные подходы к преподаванию школьного курса информатики или разработать свой собственный курс.

В этом выпуске опубликовано окончание лекций, посвященных методам алгоритмизации, и представлена лекция об устройстве компьютера.

Продолжение следует

ДОСКА ОБЪЯВЛЕНИЙ

PAINTBRUSH: *изучаем играя*

Ю.Л. ВОРОНЦОВА, О.Ю. САЛОВА

В данной работе мы не преследовали цель описать приемы работы с некоторым конкретным графическим редактором. Прежде всего мы стремились показать некую последовательность изучения данной темы. Здесь представлено множество рисунков и заданий, которые можно выполнить как в Paintbrush, так и практически в любом растровом графическом редакторе (например, Corel PhotoPaint). Последовательное выполнение заданий позволит в легкой, игровой форме освоить азы графических технологий. При составлении заданий мы предполагали, что пользователь умеет работать в среде Windows.

Уроки № 1—2. Разукрашки

Цель: познакомить учащихся с графическим редактором, основными инструментами — **Заливка, Лупа, Карандаш, Ластик**; отработать навыки запуска графического редактора, открытия и сохранения файла рисунка.

Начните урок с объяснения того, что такое графический редактор и что с его помощью можно делать. Для выполнения практической работы вам понадобятся 10 картинок различной сложности, причем для работы с инструментами **Лупа** и **Карандаш** потребуется пара рисунков с контурными линиями, имеющими разрыв. В процессе выполнения работ научите ребят открывать файлы, работать с инструментом **Заливка**, закрывать файл не сохраняя. Предоставьте ребятам возможность поработать и поэкспериментировать с цветом. Не забудьте объяснить, что последнее действие можно отменить, воспользовавшись соответствующей командой в меню **Правка**.

Рис. 1. Примеры рисунков для раскрашивания

Урок № 3. Рисовалки

Цель: научить задавать атрибуты страницы, очищать графический экран, рисовать карандашом и кисточкой.

Начните этот урок с установки атрибутов страницы. Для этого выберите команду **Рисунок—Атрибуты** и в диалоговом окне укажите число пикселей по горизонтали (550) и по вертикали (330) (таким образом, у рисунка не будет линий прокрутки). Очистите графический экран командой **Рисунок—Очистить** и предложите ребятам **Карандашом** или **Кисточкой** выполнить узоры (образцы узоров приведены на *рис.* 2).

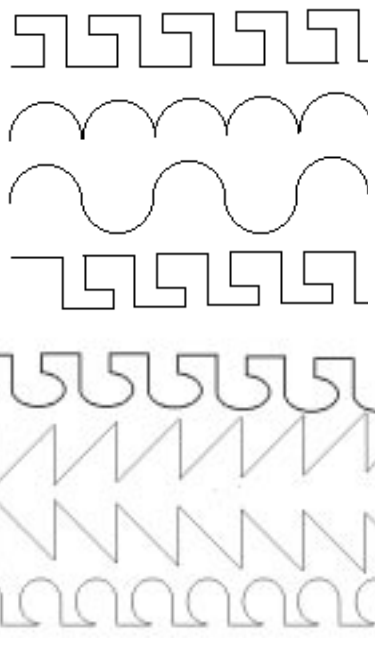


Рис. 2. Узоры для отработки навыка владения инструментом **Кисть**

Мы советуем вам последующие три урока начинать именно с таких упражнений, что позволит ребятам приобрести некоторый опыт при работе с графическим листом, мышкой и кисточкой. В завершение третьего урока предложите ребятам нарисовать один из рисунков (*рис.* 3).

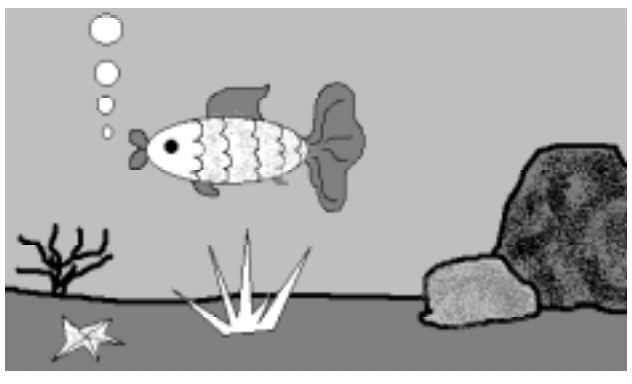


Рис. 3. Образцы рисунков

Ключ к заданию

Используются инструменты: **Кисть** для прорисовки контуров изображений, **Заливка цветом** для окрашивания замкнутых контуров, **Ластик** для правки, **Карандаш** для отрисовки более тонких контуров (чешуйки у рыбки и оперение цапли), **Распылитель** для тонирования камней.

Уроки № 4—5. Распылялки

Цель: научить пользоваться распылителем.

Начните урок с того, что расскажите о некоторых стилях рисования — это и графика, и акварель, и масло, в том числе и мазковая графика. Покажите ребятам репродукции картин, желательно, чтобы среди них было побольше пейзажей, это поможет выбрать сюжеты для последующих рисунков. Первый рисунок обговорите поэтапно: что, за чем, как и какими инструментами выполняется. Обычно ребята не могут сразу придумать свой сюжет, разложить рисунок на составляющие и определить, что за чем рисуется, — это придет с опытом.

Рис. 4. Образец рисунка, выполненного **Распылителем**

Ключ к заданию

1. Нарисовать линию горизонта.
2. **Распылителем** нанести контур леса.
3. **Ластиком** стереть линию горизонта за лесом.
4. Меняя цвет и радиус рассеивания **Распылителя**, нарисовать лес и тучу.
5. Наложением двух окружностей нарисовать луну.

Рис. 5. Образец рисунка, выполненного **Распылителем**

Ключ к заданию

1. Нарисовать кистью линию горизонта.
2. **Залить** темно-голубым цветом небо и светло-голубым — снег.
3. Стереть линию горизонта **Ластиком**.
4. **Распылителем** маленького радиуса нарисовать белого снеговика, зеленые елочки, желтый месяц, белый снег на елочках.
   Распылителем среднего радиуса нанести белые, синие, фиолетовые и желтые "искры" на снегу, а самым большим **Распылителем** белого цвета нанести снежинки в воздухе.
5. **Карандашом** дорисовать глаза и нос-морковку снеговику.

На примере данного упражнения хорошо продемонстрировать, как густота распыляемых точек зависит от скорости движения мышки.

Уроки № 6, 7, 8, 9. Рисовалки фигурами

Цель: научить пользоваться инструментами **Окружность, Прямоугольник, Линия, Ломаная, Кривая**.

Для приобретения и закрепления навыков работы с каждым инструментом подберите свой рисунок, например:

1. **Линия, Ломаная**.

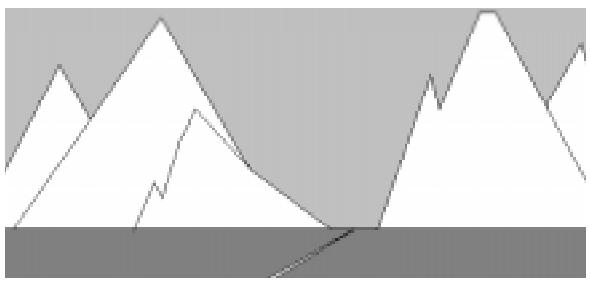


Рис. 6. Рисунок **Линиями**

2. **Прямоугольник**.

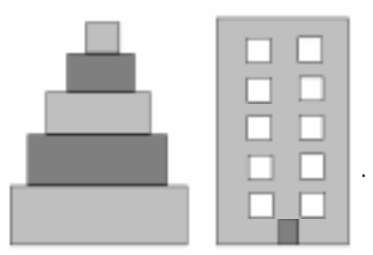


Рис. 7. Рисунок, выполненный **Прямоугольниками**

3. **Окружность**.

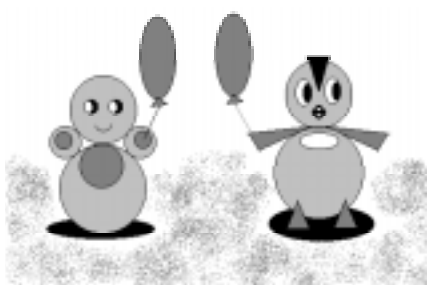


Рис. 8. Рисунок, выполненный **Окружностями**

4. **Кривая** (линия Безье).

Рис. 9. Рисунок, выполненный **Кривыми**

5. Все изученные инструменты.

Предложите ребятам самостоятельно придумать рисунок, для выполнения которого понадобились бы все изученные к этому моменту инструменты. Это должна быть их творческая работа.

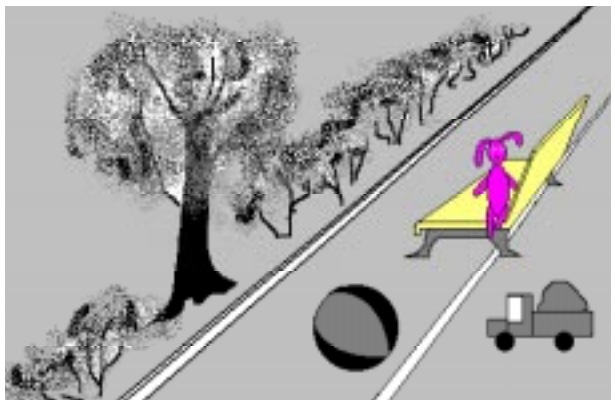


Рис. 10. Рисунок, выполненный несколькими инструментами

Ключ к заданию

1. **Линией** рисуем дорогу и контуры скамьи, ножки скамьи надо дорисовать кистью или карандашом.
2. **Кистью** — ствол и сучья дерева.
3. **Заливкой** окрасить траву, дорогу, скамью.
4. **Карандашом** рисуем веточки кустов.
5. **Распылителем** накладываем листву на дереве и кустах.
6. Для рисования игрушек используем:
   — мяча: **Круг** и **Дугу**;
   — машинки: **Прямоугольники** и **Круги**;
   — зайчика: **Овал, Дугу, Карандаш**.
   Раскрашиваем с помощью **Заливки** или, где можно, рисуем заранее заданным цветом.
7. Одуванчики на траве, выполненные маленьким желтым **Распылителем**, довершат картину.

Обратите внимание на работу учащихся по редактированию мелких деталей на увеличенном изображении.

Уроки № 10, 11, 12. Копирование

Цель: научить выделять фрагмент рисунка, перемещать его, копировать, вставлять, отражать слева направо и сверху вниз.

Одной из основных функций любого графического редактора является работа с выделенными фрагментами рисунка. Научите ребят выделять фрагмент и перетаскивать его по листу. Объясните, как вырезать, скопировать и вставить выделенный фрагмент. Для закрепления навыков предложите нарисовать картинку "Флот".

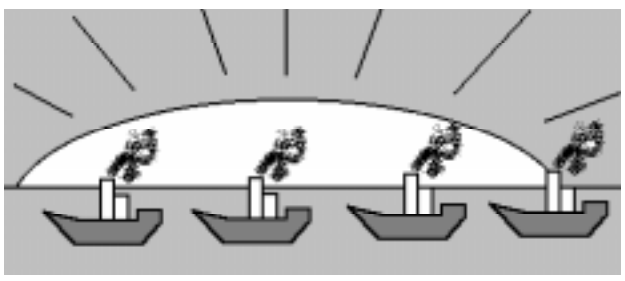


Рис. 11

Ключ к заданию

1. Нарисовать с помощью инструмента **Линия** линию горизонта.
2. Нарисовать с помощью инструмента **Эллипс** солнце, с помощью **Ластика** удалить нижнюю часть эллипса.
3. **Заливкой** окрасить море, небо, солнце.
4. С помощью инструментов **Линия, Распылитель, Прямоугольник, Заливка** на свободном месте экрана нарисовать кораблик, выделить его и скопировать в буфер или используя клавишу Ctrl (копирование осуществляется при перемещении с нажатой клавишей Ctrl).
5. Вставляя кораблики из буфера, расположить их по полю рисунка.

Следующий этап — отражение фрагментов рисунка. Обратите особое внимание на то, что если не выделен фрагмент, то выполняется отражение всего рисунка. Для дальнейшей работы потребуются несколько заготовок, копирование и отражение которых приводит к появлению интересных рисунков.

Ключ к заданию

1. Выделить рисунок;
2. **Скопировать** его в буфер;
3. **Вставить** из буфера;
4. **Отразить** слева направо;
5. Переместить выделенный фрагмент для объединения обеих половинок корзинки.

Рис. 12. Фрагмент корзинки для рисунка "Цветы"

Рис. 13. Корзинки, полученные **Копированием** и **Отражением** фрагмента

Более сложным, но не менее интересным заданием будет рисунок "Лягушка на болоте".

Ключ к заданию

1. Нарисовать три **Овала**: голову, туловище и животик.
2. Нарисовать глаз из трех **Окружностей** и **Скопировать** его на голову лягушки (необходимо в режиме увеличенного изображения **Ластиком** стереть границу овала — головы за глазами).
3. На свободном поле рисунка **Дугами** нарисовать переднюю лапку, дополнить ее пальчиками при помощи **Карандаша**, **Скопировать** и **Отразить** слева направо вторую лапку, переместить обе лапки на туловище.

Рис. 14. "Лягушка на болоте"

4. Нарисовать **Овал, Наклонить** его на 45^0, дорисовать **Карандашом** лапку. Готовую заднюю лапку **Скопировать**, **Отразить** слева направо и переместить обе лапки к туловищу.
5. Рот можно нарисовать **Дугами** или вырезать половину от двух одинаковых смещенных овалов, поместить рот на нужное место; нос — две точки, поставленные **Кистью**.
6. Ресницы — это скопированные и перемещенные к глазам части овала.
7. Переместить готовую лягушку в правый верхний угол рисунка, а в центре его нарисовать большой овал и подправить его **Карандашом** и **Ластиком** до формы листа кувшинки.
8. "Посадить" лягушку на лист и раскрасить **Заливкой** весь рисунок.

Уроки № 13, 14, 15, 16. Ввод текста

Цель: научить размещать текст на рисунке.

Редко кому не приходилось оформлять приглашения, делать открытки, придумывать иллюстрации к стихотворению, рассказу, сочинению. Или, что более современно, оформлять рекламный лист, визитку, прайс-лист... Начните с создания приглашения. Разработайте рисунок приглашения и разместите на нем текст с помощью инструмента **Текст**.

Рис. 15. Заготовка для создания приглашения

Затем предложите ребятам придумать иллюстрации к известным стихотворениям или сказкам.

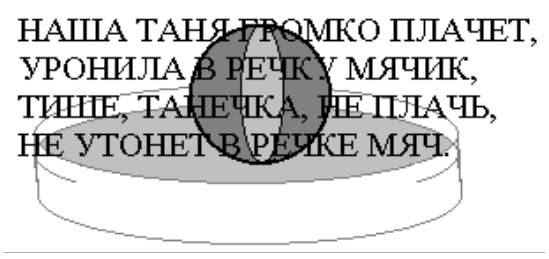


Рис. 16. Иллюстрация к стихотворению

Ключ к заданию

1. Нарисовать иллюстрацию.
2. Напечатать текст, выделить его и, перемещая по полю рисунка, наложить на иллюстрацию (а не наоборот!).

При размещении текста на рисунке часто трудно угадать размер шрифта и точное расположение текста. В подобных случаях мы рекомендуем размещать текстовые фрагменты на свободном поле рисунка с последующим их выделением и перемещением.

Рис. 17. Иллюстрация к песне

Ключ к заданию

1. При помощи инструмента **Линия** нарисовать лесенку и нотный стан.
2. Инструментом **Круг** нарисовать овалы.
3. **Кистью** или **Карандашом** нарисовать ключ "соль" начиная со второй линейки.
4. На свободном поле рисунка, подобрав размер шрифта, напечатать слова "раз, два, ступенька, словечко, будет" и, выделяя их **Ножницами**, перемещать в нужное место. Слова "ЛЕСЕНКА" и "ПЕСЕНКА" необходимо перемещать по одной букве.

Реклама вторглась в нашу жизнь и стала неотъемлемой ее частью. Поэтому умение разработать рекламный лист может очень пригодиться в будущем. Предложите ребятам несколько тем для оформления рекламного объявления, разработайте одно вместе. Например, можно разработать рекламу школы, рекламу издательства, рекламу фирмы, рекламу продукции и т.д.

Рис. 18. Пример рекламного объявления

Визитки — это реклама самого себя и своего предприятия. При создании визиток нет никаких ограничений, кроме размеров. Размер визитки — 9 см × 5 см. Не забудьте установить соответствующие размеры в **Атрибутах**.

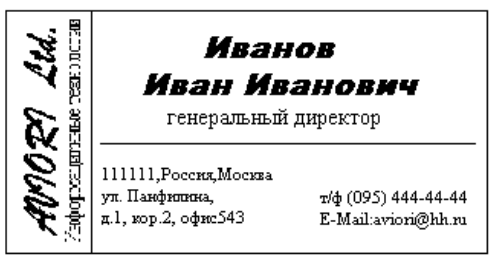


Рис. 19. Пример визитки

Ключ к заданию

1. Установить размеры графического экрана посредством команды **Рисунок—Атрибуты**.
2. Напечатать название фирмы с выбором размера и стиля шрифта, выделить текст и повернуть его на 270 градусов, переместить его к краю визитки.
3. Выбирая размер и гарнитуру шрифта, напечатать фрагменты текста визитки и разместить их на рисунке.

Ресторан

С.М. ОКУЛОВ

В ресторане собираются N посетителей. Посетитель с номером i, имеющий сумму денег P_i и полноту S_i, подходит к двери ресторана во время T_i. Входная дверь ресторана имеет K состояний открытости. Состояние открытости двери может изменяться на одну единицу за единицу времени: дверь открывается на единицу или остается в том же состоянии. В начальный момент времени дверь закрыта (состояние 0). Посетитель с номером i входит в ресторан только в том случае, если дверь открыта специально для него, то есть когда состояние открытости двери совпадает с его степенью полноты S_i. Если в момент прихода посетителя состояние двери не совпадает с его степенью полноты, то посетитель уходит и больше не возвращается.

Ресторан работает в течение времени T.

Требуется, правильно открывая и закрывая дверь, добиться того, чтобы за время работы ресторана в нем собрались посетители, общая сумма денег у которых максимальна.

Входные данные

• В первой строке находятся значения N, K и T, разделенные пробелами ($1 \le N \le 100$, $1 \le K \le 100$, $0 \le T \le 30000$).

• Во второй строке находятся времена прихода посетителей T_1, T_2, ..., T_N, разделенные пробелами ($0 \le T_i \le T$, для всех $i = 1, 2, ..., N$).

• В третьей строке находятся величины суммы денег у посетителей P_1, P_2, ..., P_N, разделенные пробелами ($0 \le P_i \le 300$, для всех $i = 1, 2, ..., N$).

• В четвертой строке находятся значения степени полноты посетителей S_1, S_2, ..., S_N, разделенные пробелами ($1 \le S_i \le K$ для всех $i = 1, 2, ..., N$). Все исходные данные — целые числа.

Выходные данные

Выводится одно число — максимальная сумма денег у посетителей ресторана. В случае, если нельзя добиться прихода в ресторан ни одного посетителя, выходной файл должен содержать значение 0.

Пример входных данных:

```
 4   10   20
10   16    8   16
10   11   15    1
10    7    1    8
```

Результат:

```
26
```

Пример входных данных:

```
 2   17   100
 5    0
50   33
 6    1
```

Результат:

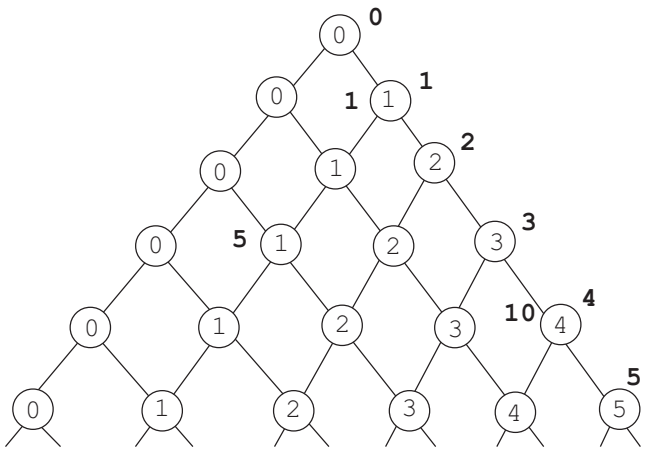
```
0
```

Решение

Это классическая задача на метод динамического программирования. Состояния открытости двери можно представить "треугольной решеткой", изображенной на рисунке. Каждая вершина определяет степень открытости q в момент времени t. Некоторым вершинам решетки приписаны веса, равные сумме денег у посетителя, приходящего в момент времени t и имеющего степень полноты, равную q. Требуется найти путь по решетке, проходящий через вершины, сумма весов которых имеет максимальное значение. Следует отметить, что нет необходимости хранить оценки всех вершин. Нам необходимо подсчитать оценки для момента времени t. Это возможно, если известны их значения для всех предыдущих моментов времени, однако для подсчета достаточно помнить только оценки для момента времени $t-1$. Приведем более простой пример входных данных:

```
3    5    6
3    4    1
5   10    1
1    4    1
```

Соответствующая решетка приведена на рисунке. Справа на рисунке указаны моменты времени. Слева от некоторых вершин решетки приведены суммы денег у посетителей, приходящих в этот момент времени и имеющих степень полноты, совпадающей со степенью открытости двери, записанной в вершине решетки. Ответ для данного примера очевиден: 11.



Данные

```
Const MaxN=500;
      MaxK=100;
Var   N,K,Tend:Integer;
      S,P,T:Array[0..MaxN] Of Integer;
```

Основная программа

```
Begin
 Init;
 Solve;
 Done;
End.
```

Приведем только процедуру `Solve`, остальные достаточно очевидны.

```
Procedure Solve;
 Var i,j:Integer;
SOld,SNew:Array[0..MaxK] Of LongInt;{массивы
для формирования оценок вершин решетки}
 Begin
 SOld[0]:=0;
 For i:=1 To K Do SOld[i]:=-MaxLongInt;
 SNew:=SOld;
 For i:=1 To Tend Do Begin
 {цикл по моментам времени}
  SNew[0]:=SOld[0];
  For j:=1 To i Do
  {цикл по достижимым состояниям решетки}
   SNew[j]:=Max(SOld[j-1],SOld[j]);
    {реализация функции Max очевидна}
    {формирование оценок вершин решетки}
  For j:=1 To N Do {цикл по посетителям}
   If (T[j]=i) And (SNew[S[j]]<>-MaxLongInt)
   {если время прихода посетителя совпадает
    с рассматриваемым моментом времени
    и состояние достижимо, то изменяем
    оценку вершины, соответствующей полноте
    посетителя}
    Then Inc(SNew[S[j]],P[j]);
 SOld:=SNew;{запоминаем массив оценок}
End;
Res:=-MaxLongInt;
For i:=1 To K Do Res:=Max(Res,SNew[i]);
  {находим максимальное значение
   в окончательном массиве оценок}
End;
```

ЗАДАЧИ

МИНИСТЕРСТВО ОБЩЕГО И ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ РФ
ФОНД НОВЫХ ТЕХНОЛОГИЙ В ОБРАЗОВАНИИ "БАЙТИК"
ДЕПАРТАМЕНТ ПО ОБРАЗОВАНИЮ МОСКОВСКОЙ ОБЛАСТИ
ЦЕНТР НОВЫХ ПЕДАГОГИЧЕСКИХ ТЕХНОЛОГИЙ
ИНСТИТУТ ЮНЕСКО ПО ИНФОРМАЦИОННЫМ ТЕХНОЛОГИЯМ В ОБРАЗОВАНИИ
COMPUTER USING EDUCATORS INC, USA

X МЕЖДУНАРОДНАЯ КОНФЕРЕНЦИЯ "ПРИМЕНЕНИЕ НОВЫХ ТЕХНОЛОГИЙ В ОБРАЗОВАНИИ"

Троицк, Московская область
30 июня — 3 июля 1999 года

Приглашаем вас принять участие в юбилейной X международной конференции "Использование новых технологий в образовании" в рамках the 10th International Technology Institute, которая пройдет в период с 30 июня по 3 июля 1999 года на базе фонда новых технологий в образовании "Байтик". В конференции примут участие американские представители организации Computer Using Educators (CUE) и специалисты других стран.

Официальные языки — русский и английский. Конференция посвящена вопросам практического использования новых технологий в образовании и будет проводиться по следующим направлениям:

1. Новые технологии для детей дошкольного и младшего школьного возраста.
2. Преподавание школьных дисциплин: информатика, естественные предметы, гуманитарные предметы, экономика и иностранные языки.
3. Новые технологии в учебном процессе, дистанционное обучение и Интернет.
4. Информационные технологии в профессиональной подготовке.
5. Учебники и учебные пособия.
6. Олимпиады и конкурсы по информатике.
7. Методики контроля знаний обучаемых.
8. Подготовка специалистов в области информационных технологий.
9. Компьютер для администрации в образовательном учреждении.

В программу конференции будут включены:

• "Круглые столы" для обсуждения проблемных вопросов компьютерного образования.

• Выставка-ярмарка программных и технических средств, учебников, а также методических разработок российских и зарубежных фирм.

Заявку на участие в работе секций, тексты тезисов докладов необходимо выслать в адрес оргкомитета конференции до 1 июня 1999 года. Полную информацию о программе конференции, порядке оформления материалов, оплате оргвзноса вы можете получить в оргкомитете.

Адрес:
142092, Московская обл., г. Троицк, Сиреневый б-р, д. 11.
Фонд "Байтик"
Тел./факс (095)334-03-67

E-mail:
cue@bytic.troitsk.ru
cue@relay.bytic.troitsk.ru

Web:
http://www.bytic.troitsk.ru/russian/cue_reg99.html
http://stuff.bytic.troitsk.ru/cue/cue_reg99.html

ЛИРИЧЕСКОЕ ОТСТУПЛЕНИЕ. Честно говоря, я не знаю других учебников, в которых этот материал подкреплялся бы какими бы то ни было упражнениями вообще. Тут вы можете ярко наблюдать следствие основных целей и методики преподавания нашего курса, изложенных в начале пособия.

Рассмотрим простейший алгоритм — упр. 2 на с. 168 учебника. Это маленький, но настоящий интерпретатор, который анализирует некоторую информацию (строку из букв “В”, “Н”, “Л”, “П”, “К”) и заставляет Робота выполнить соответствующую последовательность команд. Вспомните схему программного управления (см.: Лекция 1, № 1/99). Человек может описать некоторый алгоритм управления Роботом в виде строки из этих букв и “отдать” программно-аппаратному комплексу (ЭВМ + КуМир + алгоритм из упр. 2), а дальше эта строка выполнится и Робот получит соответствующие команды уже без участия человека. То есть это — абсолютно “настоящий”, хотя и очень простой, интерпретатор.

Точно так же программа-интерпретатор с Бейсика анализирует программу на Бейсике и выдает команды, скажем, терминалу и клавиатуре. Точно так же система КуМир анализирует алгоритм на алгоритмическом языке и командует, скажем, экраном, изображая и сам алгоритм, и Робота. Другими словами, в плане “воды и ребенка” все в порядке — никакое содержание не потеряно. А алгоритм (простейший интерпретатор) при этом содержит меньше десятка строк!

Поэтому, даже когда мы говорим о таких понятиях, как интерпретация и компиляция, все эти понятия можно подкрепить простыми упражнениями, десятистрочными простыми алгоритмами. Единственное, что от нас требуется, — выделить содержание изучаемого явления “в чистом виде”, сконструировать такую учебную обстановку, чтобы можно было формулировать и решать задачи на компиляцию и интерпретацию.

Я еще раз хочу подчеркнуть, что хотя сами эти компиляторы и интерпретаторы маленькие (10, 15, максимум 20 строк), тем не менее это — настоящие компиляторы, настоящие интерпретаторы.

Конечно, если мы в интерпретируемый текст добавим хотя бы возможность написать цикл **n раз**, то это сильно все усложнит. Можете предложить сильным ученикам сделать так, чтобы в строке можно было написать “К9(НЛ)К” и чтобы это означало

```
закрасить
нц 9 раз
| вниз
| влево
кц
закрасить
```

Интерпретатор станет намного сложнее. После этого можно сказать: “А представьте себе, что разрешены ***все*** изученные нами алгоритмические конструкции. Какой получится интерпретатор? Конечно, очень и очень сложный и большой. Но ***в принципе*** понятно, как это делать. Написать такой интерпретатор — большая и трудная работа, но ***в принципе*** понятная. Ничего сверхъестественного в этом нет”. Я также хочу обратить ваше внимание на алгоритм А92 (с. 176 учебника), который является интерпретатором для специальным образом закодированных программ управления Чертежником. При желании вы можете пройти п. 21.6 § 21 прямо здесь, в рамках изучения интерпретации и компиляции. А потом лишь сослаться на него при изучении информационных моделей как на уже пройденный материал.

Ну и, наконец, прежде чем перейти к следующей главе, я сделаю еще одно замечание ко всей главе 2. Эта глава в каком-то смысле дань программе курса “Основ информатики ***и вычислительной техники***”. На наш взгляд, этот материал правильнее было бы изучать отдельно. Физические основы ЭВМ, транзисторы, вентили, триггеры и пр. — это изложение конкретного материала, развитие кругозора учеников. Значительная часть главы вообще относится скорее к физике и к логике, чем к развитию алгоритмического стиля мышления.

Если вся первая глава была в точности выдержана в соответствии с целями курса, то глава 2 в этом смысле несколько “привнесенная”. Она важна для формирования мировоззрения, для формирования целостного представления о том, что происходит, когда мы включаем компьютер и пр., но она как бы из другого курса. В идеале такой курс мог бы читаться независимо и, может быть, даже параллельно с курсом информатики. Его можно перенести и в конец (после главы 3), но мы поставили его в середину, между главами 1 и 3, чтобы разбить, разнообразить материал. Всю первую главу шел алгоритмический язык, потом — перерыв: вентили, коды, работа ЭВМ. Отдохнули, отвлеклись — и снова алгоритмический язык (глава 3) на новом, более высоком уровне. Чтобы курс был не совсем уж монотонный, чтобы были какие-то смены материала и рода деятельности, разнообразие.

Поэтому главу 2 можно спокойно двигать по курсу так, чтобы она попала на нужное место, туда, где вам нужен такой перерыв. Либо, если у вас курс сжатый и времени мало, главу 2 можно и вовсе опустить. При этом все остальное можно пройти. В главе 3 есть отдельные ссылки на материал главы 2, но их легко заменить на непосредственное изложение нужного материала в нужном месте.

Список литературы

[ПдМ] *Кушниренко А.Г., Лебедев Г.В.* Программирование для математиков. М.: Наука, 1988.
[Первин] *Первин Ю.А., Дуванов А.А., Зайдельман Я.Н., Гольцман М.А.* Роботландия. М.: ИТО, 1990.
[Звенигородский] *Звенигородский Г.А.* Вычислительная техника и ее применение. М.: Просвещение, 1987.
[Гейн] *Гейн А.Г., Житомирский В.Г., Линецкий Е.В. и др.* Основы информатики и вычислительной техники. М.: Просвещение, 1991.
[Ершов] Под ред. *А.П. Ершова, М.В. Монахова.* Основы информатики и вычислительной техники. Чч. I и II. М.: Просвещение, 1986.
[ЗК] Знакомьтесь: компьютер, М.: Мир, 1989.

№ 10 (203) март 1999

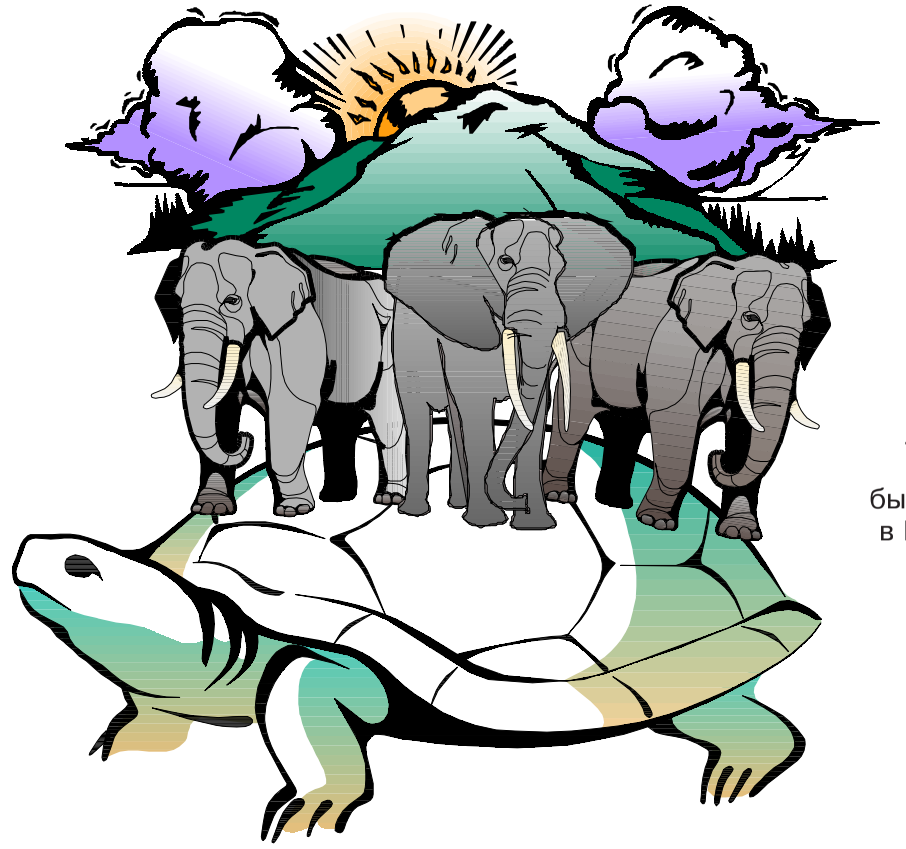
еженедельное
ПРИЛОЖЕНИЕ К ГАЗЕТЕ
«ПЕРВОЕ СЕНТЯБРЯ»

ИНФОРМАТИК

12 лекций о том, для чего нужен школьный курс информатики и как его преподавать

Лекции 7—8, 9

А.Г. КУШНИРЕНКО, Г.В. ЛЕБЕДЕВ



Введение,
Лекции 1, 2, 3, 4,
5, 6, 7—8
были опубликованы
в № 1, 3, 5, 6, 8/99

Выражаем признательность издательству “Дрофа” за содействие в подготовке публикации.

СОДЕРЖАНИЕ

Наконец, последнее, на что я еще раз хочу обратить ваше внимание. “Подключение” Робота к ЭВМ и кодирование алгоритмов управления Роботом на уровне машинных команд дает некоторое общее представление об основах работы ЭВМ. Дело в том, что такое кодирование убирает разрыв между тем, что мы обычно пишем на языке программирования, и тем, что делает машина. Решение подобных упражнений позволяет представить себе, что же происходит “внизу” — на уровне машинных команд.

Вместе с простейшим графическим редактором (упр. 9) все это должно сформировать у учеников правильное представление об исполнителях, а также показать, “как” работает ЭВМ, что происходит при работе на клавиатуре, в графическом редакторе, как микропроцессор может управлять какими-то устройствами в автомобиле и пр.

§ 20. Работа ЭВМ в целом

Это — последний параграф раздела, посвященного в целом, я повторю, демистификации ЭВМ. Демистификация происходит на четырех уровнях (по одному параграфу учебника на уровень):

1) на уровне электрических процессов, напряжения, включения и выключения транзисторов (§ 17);

2) на уровне машинных команд (§ 18);

3) на уровне взаимодействия различных частей ЭВМ (§ 19);

4) и, наконец, на уровне ЭВМ и программного обеспечения вместе как единого аппаратно-программного комплекса (§ 20).

Последний уровень (§ 20) посвящен увязке ранее пройденных понятий с реальным использованием ЭВМ. Вот школьник приходит в компьютерный класс, на экране сразу возникает система КуМир или Бейсик, и в них прямо можно работать. Как это? Что происходит в ЭВМ? Что она делает? Как это связано с устройством процессора, машинными командами и пр.? Какие машинные команды выполняет процессор, когда мы в системе КуМир на школьном алгоритмическом языке набираем алгоритм? Ответу на эти вопросы и посвящен § 20.

В самом параграфе ничего особенно содержательного или сложного нет. Параграф абсолютно традиционный и содержит только описания (слова), без каких-либо примеров программ, кодов и пр.

Говорится, что когда ЭВМ работает, процессор выполняет программу (алгоритм), написанную не нами, а кем-то другим. Когда мы нажимаем на кнопки, то в соответствии с этой программой (скажем, программой КуМир или Бейсик) эти нажатия соответствующим образом обрабатываются, и в памяти, на экране и пр. происходит то, что должно происходить. При работе в системе КуМир, например, в памяти формируется совсем не то, что мы набираем, а некоторое внутреннее представление алгоритма. Если мы набрали что-то неправильно, то КуМир (точнее, компьютер в ходе выполнения программы КуМир) проанализирует и сразу сообщит на полях (на экране) об ошибке. Когда мы нажимаем на кнопку “выполнить”, КуМир начинает строка за строкой выполнять наш алгоритм, показывая ход исполнения на экране, и т.п. Этот механизм, когда на самом деле компьютер выполняет не наш алгоритм, а некоторую другую программу (например, КуМир) и в ходе ее исполнения “изображает”, “имитирует” выполнение нашего алгоритма, называется интерпретацией. Интерпретация — это когда в памяти хранится не машинная программа для нашего алгоритма, а лишь некоторая информация об алгоритме, которая интерпретируется (“понимается и обрабатывается”) с помощью программы-интерпретатора, выполняемой ЭВМ.

Другой подход — компиляция — состоит в том, что наш алгоритм с помощью программы-компилятора переводится прямо в выполняемую программу (в машинные коды), а потом полученная программа выполняется “сама по себе”.

Эти два понятия, компиляция и интерпретация, объясняют, как вообще программа на языке программирования выполняется ЭВМ. Обычно Паскаль является компилирующей системой, а Бейсик — интерпретатором.

Интерпретация и компиляция — это две крайние, “чистые” позиции. Естественно, в жизни часто бывают разнообразные “смеси”, но мы их рассматривать не будем. КуМир, например, система более сложная, и в реальности она сделана по технологии следующего уровня, называемой синтезом программ, но в рамках учебника ее тоже можно считать интерпретатором.

Что еще можно сказать про интерпретацию и компиляцию? При интерпретации выполняется (реально, “машинно”) программа-интерпретатор, которая “имитирует” выполнение нашего алгоритма. При компиляции алгоритм сначала переводится в машинную программу, а потом выполняется. При компиляции на этапе выполнения ничего лишнего не делается. Поэтому обычно скорость выполнения при интерпретации существенно (раз в сто) ниже, чем при компиляции. Алгоритм на Бейсике выполняется в сотню раз медленнее, чем такой же алгоритм на Паскале.

Для полного завершения формирования картины мира осталось объяснить самую малость: откуда и как в памяти ЭВМ появляются КуМир или Бейсик? И что вообще происходит при включении компьютера в сеть? Какую программу и почему он начинает выполнять?

Ответу на этот вопрос посвящены два оставшихся раздела (пп. 20.5 и 20.6). Важно, что описание программы начальной загрузки, функций операционной системы производится в уже освоенных терминах машинных программ и процессора, который занят беспрестанным (бесконечным) их выполнением. Скажем, что значит: начинает выполняться программа из постоянного запоминающего устройства? Это значит, что на заводе в эту память поместили соответствующую машинную программу, а сам компьютер сделали так, что при включении питания в счетчик команд попадает адрес начала этой программы. Что значит: считывается с диска? Это значит: посылаются соответствующие команды исполнителю “диск” (или, на более низком уровне, на проводах магистрали появляются соответствующие высокие и низкие напряжения). И т.д. Главное, теперь за всеми этими словами стоит (должно стоять) реальное понимание, реальное наполнение — что именно и как делает компьютер.

Формирование такого целостного (без разрывов) восприятия от выполнения алгоритма на алгоритмическом языке до включений и выключений транзисторов под действием электрического напряжения и является целью всей главы.

Сам § 20, я повторяю, является чисто описательным. Но, памятуя о “черепахе”, для реального усвоения этого материала школьники должны не просто читать и слушать, а сами решать какие-то задачи. Полезно заставить их писать компиляторы и интерпретаторы. Конечно, не настоящие (это слишком сложно и трудоемко), а какие-нибудь простенькие, учебные. Именно этому и посвящены упражнения на с. 168 учебника.

В параграфе имеется два очень важных момента для формирования информационной культуры, для понимания общей картины мира, в котором появились и используются компьютеры. Это —

1) управление исполнителями Экран, Клавиатура и др. непосредственно из алгоритмического языка;
2) "физическое" подключение Робота к ЭВМ (в упражнениях).

Кратко остановимся на этих моментах.

Мы сказали, что клавиатура, экран, принтер — просто исполнители, значит, с ними можно работать так, как с Роботом, посылая им команды, управляя ими из алгоритмического языка. Подобно тому, как мы командовали Роботу "вправо", "влево", "вверх", "вниз", мы можем скомандовать экрану "вывести символ" или клавиатуре "ввести символ". И в параграфе приводятся некоторые команды исполнителей Экран и Клавиатура (пп. 19.7 и 19.8), команды алгоритмического языка, которые позволяют работать с экраном и клавиатурой: "цвет", "точка", "позиция", "вывести символ" и т.д. Это действительно реальные команды, и в системе КуМир можно писать алгоритмы с использованием этих команд, решать задачи типа "написать алгоритм, который очищает экран целиком" (помните "черепаху" — "понимание через делание"?). Или "написать алгоритм, который определяет, нажимает ли человек стрелки вправо или влево на клавиатуре и рисует ли на экране линию в соответствующем направлении". Последняя задача — это простейший прототип графического редактора, алгоритм буквально всего из 15 строчек на алгоритмическом языке: ввод символа от человека и вызов вспомогательного алгоритма, рисующего линию. Зато какое понимание сути работы настоящих — весьма сложных — графических редакторов получит ученик, однажды написавший такой алгоритм! (См. также упр. 6—9 к этому параграфу.)

Помните, я рисовал роль понятия "исполнитель" (см.: Лекция 2, № 3/98)? Теперь, я думаю, вы понимаете, что я имел в виду. Тут важно даже не столько то, что монитор, дисковод и пр. оказались просто исполнителями, сколько, я повторюсь, возможность изучать и понимать их ***содержательно*** в рамках тех понятий, которыми мы уже овладели.

В учебнике приведены некоторые команды клавиатуры и экрана. У исполнителя "внешняя память" (диск) система команд более сложная — на содержательном уровне там надо сразу вводить понятие файла и т.п., однако и это возможно. Соответствующие исполнители есть в системе КуМир, и вы можете более сильных учеников загружать изучением более сложных устройств ЭВМ, более сложных исполнителей.

Второй важный момент, о котором я упомянул, — "физическое подключение" Робота к ЭВМ. Этот момент, правда, разбит на две части: в самом учебнике сказано лишь, как в принципе подключить какого-то исполнителя к ЭВМ (п. 19.5 — магистраль, адрес). А все, что касается Робота, сформулировано в качестве упражнений (упр. 1 на с. 164 учебника) и предлагается придумать самим ученикам. Это вторая крайне важная составляющая для глубокого осознания понятия "исполнитель". Можно клавиатурой управлять из алгоритмического языка (первая составляющая), а для управления Роботом составлять программы в машинных кодах (вторая составляющая). Общая цель всех этих упражнений — стереть всякое различие между Роботом, клавиатурой, экраном и другими исполнителями. Показать, что Робот может оказаться "машинным" исполнителем, а клавиатура — "алгоритмическим".

Я очень бегло повторю, что значит "подключение" Робота к ЭВМ. Во-первых, это значит, что Робота надо подключить к магистрали, т.е. на *рис.* 81 учебника (с. 162) добавить прямоугольник "Робот". Во-вторых, по аналогии с другими устройствами Роботу следует выделить "его" адреса: один для передачи команд Роботу, а другой для получения информации от Робота (что и сделано в формулировке упр. 1). При реальном подключении Робота его команды должны быть как-то закодированы числами, так что "запись" соответствующего числа по адресу 65352 будет означать выдачу нужной команды Роботу, а чтение числа по адресу 65354 будет получением информации от Робота.

Эту кодировку ученики должны придумать сами. Например, "вправо" — 1, "влево" — 2, "вверх" — 3, "вниз" — 4, "закрасить" — 5, "справа стена?" — 6 и т.п. При этом команда "записать по адресу 65352 число 6" будет означать вопрос "справа стена?", по которому Робот должен проанализировать, есть справа стена или нет, и запомнить ответ, который далее можно будет, например, прочесть по адресу 65354. Это значит — формулировка громоздкая, но правильная до самых деталей, — что по команде "переслать из 65354 в *x*" Робот, обнаружив "свой" адрес, перешлет в *x*, скажем, 0, если он до того запомнил **нет** (стены справа не было), либо 1, если он запомнил **да** (стена справа была). Естественно, надо будет придумать какие-то комбинации 0 и 1, кодирующие также температуру и радиацию.

После того как такая кодировка команд Робота будет придумана, алгоритмы управления Роботом (самые простые — начиная с А1) можно переписать ***как реальную*** программу в ***кодах реальной*** ЭВМ. Единственная условность — что само "подключение Робота" у нас воображаемое, реально таких устройств, подключаемых к ЭВМ, наша промышленность пока не производит (хотя и могла бы — это совсем не сложно).

И, что самое важное, все это "настоящее". Именно так в действительности новые устройства подключаются к компьютеру. Именно так бортовые компьютеры по заданному алгоритму управляют космическим кораблем или системой впрыска и зажигания в автомобиле. И я глубоко уверен, что, не разобравшись — хотя бы в принципе, — как такое возможно, нельзя создать "адекватную информационную картину мира", которую мы сформулировали как третью цель нашего курса.

> МЕТОДИЧЕСКОЕ ОТСТУПЛЕНИЕ. Я никогда не мог это понять, но, как это ни странно, подобное "машинное" управление Роботом почему-то полностью снимает с Робота налет "игрушечности". Стоит управление Роботом погрузить в дебри машинного кодирования, нулей и единиц, как даже запись алгоритма А1 начинает казаться чем-то "настоящим". Хотя алгоритмическая сложность А1 от этого никак не меняется — алгоритмически задача остается абсолютно тривиальной.
>
> Этот же феномен мы наблюдаем и с Бейсиком — запись абсолютно тривиальных задач на Бейсике выглядит как что-то содержательное, в то время как решение содержательной задачи на алгоритмическом языке с Роботом может казаться простым и "игрушечным".
>
> И я еще раз обращаю ваше внимание на этот эффект и повторяю, что наша цель — развитие алгоритмического мышления учащихся, решение содержательных задач. И именно поэтому мы сделали выбор в пользу содержательных задач на алгоритмическом языке с Роботом, а не в пользу элементарных задач в кодах или на Бейсике.

Лекции 7—8

§ 16. Методы алгоритмизации

16.3. Метод 3 – инвариант цикла

Мы переходим к третьему методу этого параграфа, который называется инвариантом цикла. Точнее, — и к великому моему сожалению, — в учебнике изложен не столько метод "проектирования цикла с помощью инварианта", сколько просто понятие инварианта цикла. Содержательное и практическое применение этого понятия для решения алгоритмических задач, т.е. собственно метод алгоритмизации, по большому счету осталось за рамками учебника. И даже то, что в учебнике есть, сделано, конечно, на достаточно простом школьном уровне. Тех, кто хочет овладеть методом по-настоящему, содержательно, я опять отсылаю к учебнику [ПдМ], раздел "Проектирование цикла с помощью инварианта".

Итак, инвариант цикла. Повторю, что в учебнике мы ограничились введением самого понятия, а метод как таковой не разбирали, хотя о каких-то простейших применениях понятия инварианта и его пользе при составлении алгоритмов немного рассказали.

Что такое инвариант цикла.

Инвариант цикла — очень простое понятие. Какой бы ни был цикл, внутри него обычно что-то меняется. Например, положение Робота, значения каких-то величин, возможно, что-то еще — состояния каких-то объектов и значения каких-то величин. В примере, который мы только что обсуждали, менялись положение Робота и значения величин *N, M, x*. Можно попытаться описать взаимосвязь между всеми этими величинами и положением Робота. Фактически я так и делал, когда говорил, что *x* — температура в той клетке, где стоит Робот; *N* — максимальная из сумм, которые кончаются на правом краю пройденной части коридора; *M* — максимальная из всех сумм вообще в пройденной части коридора. Слова "пройденная часть коридора" связывают значения этих величин с положением Робота.

Грубо и приближенно такое описание взаимосвязи между меняющимися в цикле объектами (в данном случае между положением Робота и значениями величин *M* и *N*), такое общее описание взаимосвязи и называется инвариантом цикла. Главное, чтобы это описание было общим — верным для любого шага цикла (в нашем примере — для любой клетки коридора), не зависящим от того, выполнилось тело цикла один, два, миллион раз или не выполнилось ни разу, т.е. чтобы наше описание взаимосвязи не менялось в ходе выполнения цикла, оставалось неизменным (по-латыни *invariant* и значит "неизменный") независимо от того, сколько раз выполнялось тело цикла.

Высказывание "значение величины *x* равно температуре в клетке, где стоит Робот" в терминах информатики — ***условие***. Оно может быть верным, если *x* действительно равен указанной температуре, либо ложным, если в силу каких-то причин значение величины *x* окажется не совпадающим с температурой в названной клетке.

Неизменное условие, описывающее взаимосвязь меняющихся в цикле объектов, и называется инвариантом. Здесь, конечно, мы сильно упрощаем. На "настоящий" инвариант накладывается еще целый ряд ограничений, которым он должен удовлетворять. Отнюдь не любое неизменное условие называется инвариантом. Реальный инвариант должен быть еще и "полезным", но понятие "полезно" (как и стоящие за ним строгие математические рассуждения) я здесь раскрывать не буду. Мы это пропустим.

В рамках школьного курса акцент следует сделать на двух свойствах инварианта.

Первое. При составлении любого алгоритма полезно описать взаимосвязь между меняющимися в цикле объектами (т.е. явно выписать инвариант), чтобы лучше понимать, что происходит. Ведь такое описание будет статическим, неизменным, не зависящим от динамики выполнения цикла. А опыт показывает, что человек гораздо лучше воспринимает и понимает статические (неизменные) условия. Вспомните, насколько статические **дано** и **надо** проще понять, чем текст алгоритма. Кроме того, здесь фактически происходит связь и опора на другую культуру мышления.

И **второе**. Если уж мы такое неизменное условие — "инвариант" — придумали, если взаимосвязь между объектами выразили, то, зная, как в ходе выполнения тела цикла должен меняться один из объектов, мы можем не придумывать, а ***выводить***, как будут меняться остальные объекты. В этом состоит практическая и немедленная польза от использования инварианта. В задаче с суммами, сформулировав взаимосвязь между положением Робота и величинами *N* и *M* и зная, что Робот должен сместиться на одну клетку вправо по коридору, мы, в сущности, ***выводили***, как должны меняться *N* и *M*.

Основное практическое значение инварианта состоит в том, что, зная, как меняется одна величина, и зная взаимосвязь между величинами, можно выводить, как должны меняться другие. Не придумывать, а выводить.

Обычно инварианты достаточно просты. Например, для задачи подсчета числа стенок над Роботом (*рис.* 1) инвариант цикла можно сформулировать так:

"N = числу стенок в пройденной части, включая клетку, в которой стоит Робот (т.е. число стенок слева или над Роботом)".

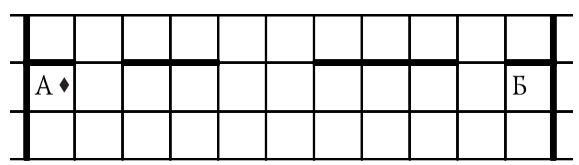


Рис. 1

Этот инвариант связывает значение величины *N* и положение Робота. Теперь, имея инвариант, имея такое описание взаимосвязи, мы можем сказать, что в начальный момент *N* ***должно быть*** 1, если над первой клеткой есть стена, либо *N* ***должно быть*** 0, если стены над первой клеткой нет. Обратите внимание на это ***должно быть***, которое мы ***вывели*** из инварианта, ***подставив*** в инвариант в качестве местоположения Робота первую (левую) клетку.

Аналогично при переходе в следующую клетку (при "удлинении" пройденной части) мы, рассмотрев инвариант *до* перехода и *после* перехода, можем ***вывести***, как должно меняться N при таком переходе (при смещении Робота на одну клетку вправо):

$$N_{\text{новое}} = \begin{cases} N_{\text{старое}} + 1, & \text{если над Роботом первая клетка (левый край) новой стенки} \\ N_{\text{старое}}, & \text{иначе} \end{cases}$$

Мы придумали взаимосвязь (инвариант) и решили, что Робот будет двигаться вправо шаг за шагом, клетка за клеткой, каждый раз в теле цикла смещаясь на одну клетку вправо. После этого нахождение начальных значений величин и формул их преобразования в теле цикла уже не требует анализа всей задачи. Достаточно "подставить" в инвариант соответствующие положения Робота и ***вывести***, чему должны быть равны и как должны меняться эти величины, чтобы инвариант (т.е. описанная взаимосвязь объектов) сохранился.

В большинстве задач инварианты достаточно просты и, как правило, очевидны. Однако даже для одной и той же задачи обычно можно придумать несколько разных инвариантов и, соответственно, получить разные алгоритмы.

В примере выше я полагал, что Робот расположен в правой клетке пройденной полосы, и получил выписанный выше инвариант. Если считать, что "пройденная полоса" не включает клетку с Роботом, то получится другой инвариант и другой алгоритм.

Конечно, на этом примере трудно показать полезность, осмысленность инварианта. Задачу можно решить и без инварианта, и явное выписывание инварианта в данном случае скорее проясняет смысл понятия инвариант, чем реально помогает решить задачу (составить алгоритм). Просто в этой задаче взаимосвязь между числом препятствий и положением Робота очевидна.

Сейчас я покажу, что такая взаимосвязь не всегда очевидна и инвариант иногда здорово помогает либо понять алгоритм (первое свойство инварианта), либо составить его (второе свойство — "вывод", как должны меняться величины). Я приведу пример из учебника, в котором этот инвариант неочевиден и, более того, кажется странным.

Но до того повторю еще раз, что инвариант — это неизменное условие, описывающее взаимосвязь объектов (величин) в цикле. Прелесть его состоит в том, что он:

1) помогает лучше понять, что делает цикл, когда мы анализируем уже составленный нами или другим человеком алгоритм;
2) помогает записать цикл при составлении алгоритма за счет "вывода" того, как должны меняться какие-то объекты, величины в цикле, чтобы сохранить инвариант при изменении основного объекта за один шаг выполнения тела цикла (в наших примерах — при изменении положения Робота).

Итак, задача, которая есть в учебнике, — алгоритм А87:

```
алг вещ степень (арг вещ a, арг цел n)
    (А87)
дано n>0
надо знач = a**n
нач цел k вещ t, b
| b:=1; t:=a; k:=n
| нц пока k≠0
|     если mod(k,2)=0
|     |   то t:=t*t; k:=k/2
|     |   иначе b:=b*t; k:=k-1
|     все
| кц
| знач:=b
кон
```

Утверждается, что этот алгоритм вычисляет a^n. Если непредвзято на него взглянуть, то мы вынуждены будем признать, что вообще непонятно, какое отношение этот алгоритм имеет к задаче вычисления a^n. Казалось бы, если надо a возвести в степень n, то должен появиться цикл **n раз**, в котором что-то будет n раз умножаться на a. Но приведенный алгоритм — совсем другой. Какое отношение имеет он к нашей задаче?

Зададим простейший вопрос: как проверить, возводит этот алгоритм a в степень n или не возводит? Получится в ответе a^n или не получится? Как это понять?

Используем первое свойство инварианта (ведь алгоритм уже написан и нам надо "всего лишь" его понять!). Инвариант цикла в этом алгоритме выглядит так:

$$b \cdot t^k = a^n.$$

Величины **a** и **n** остаются неизменными (в цикле не меняются), а величины **b**, **t** и **k** меняются при выполнении тела цикла. Инвариант описывает неизменную взаимосвязь между этими меняющимися величинами — между **b**, **t** и **k**.

Что значит "взаимосвязь описывается инвариантом"? Это значит, что всякий раз после выполнения тела цикла условие (инвариант) выполнено.

Прежде чем двигаться дальше, нам необходимо проверить, что выписанное мною условие — действительно инвариант, что оно действительно будет выполнено до и после любого выполнения тела цикла, что выполнение тела цикла наш инвариант сохраняет, не разрушает.

Давайте это проверим. Вначале величинам **b**, **t**, **k** присваиваются значения **b:= 1**, **t:= a**, **k:= n**, поэтому $b \cdot t^k = 1 \cdot a^n = a^n$, т.е. инвариант выполнен. Теперь нам надо показать, что, как бы ни выполнялось тело цикла, инвариант сохранится. В теле цикла у нас бывает два разных случая — для четных и нечетных k. Если k — четное, то

$$t^k = t^{2(k/2)},$$

и поэтому переход **t:= t²; k:= k/2** инвариант не меняет. Если k — нечетное, то

$$b \cdot t^k = (bt) \cdot t^{(k-1)},$$

и поэтому переход **b:= b·t; k:= k - 1** инвариант также не меняет. Таким образом, сколько бы раз и как бы ни выполнялось тело цикла, инвариант сохранится. Заметьте, что мы показали это ***чисто математически***.

Итак, выписанное нами условие — действительно инвариант. И оно сохраняется, сколько бы раз ни выполнилось тело цикла. Поэтому оно будет выполнено и после окончания выполнения всего цикла целиком. Но цикл у нас заканчивается только тогда, когда $k = 0$. Таким образом, после окончания цикла (при $k = 0$) наш инвариант превратится в условие $b \cdot t^0 = a^n$, т.е. $b = a^n$. И снова мы чисто математически доказали, что после окончания цикла b будет равно a^n. Заметьте, что мы не говорим, ***как*** будут меняться величины

По сравнению с первыми учебниками [Ершов] оставлены только 16 команд вида "добавить содержимое по адресу 100 к содержимому по адресу 94" (с. 156 учебника). Никаких регистров, никаких видов адресации, просто содержимое по некоторому адресу памяти берется и добавляется к другому содержимому другого адреса памяти. Все остальное — как технические детали — убрано.

Из всего связанного с регистрами остались только счетчик команд, содержащий адрес команды, которая будет выполняться следующей, и биты-условия в слове состояния процессора (ССП), хранящие информацию о том, каким был результат предыдущей команды — отрицательным, положительным или нулем.

Материал этого параграфа является упрощением материала из учебников [Ершов]. Детальное (более глубокое) изложение и описание есть в любой технической литературе по серии машин PDP-11 (к ней относятся и наши УКНЦ, ДВК, БК и СМ-4). При желании материал параграфа можно заменить на аналогичный материал на базе другого компьютера. Выбор УКНЦ обусловлен тем, что, с одной стороны, это ***реальный*** компьютер, а с другой — его система команд очень проста, стройна, логична и легко может быть изложена в школе. Команды — настоящие, машины — настоящие, приведены простейшие команды, но из реального, настоящего компьютера. Здесь нет ничего придуманного.

При изложении параграфа акценты надо сделать на двух основных вещах. Первое — основной алгоритм работы процессора в п. 18.2, обеспечивающий автоматизм работы процессора и переход от выполнения одной команды к выполнению другой. Этот алгоритм, правда, сразу написан в виде **нц**–**кц**, без всяких объяснений, почему мы так пишем. Я хочу подчеркнуть, что здесь это не алгоритмическая конструкция, а просто форма записи. Процессор в этом цикле работает бесконечно (пока не встретит команду **Стоп** и не остановится).

И второе — машинная реализация управляющих конструкций (**пока**, **если**) за счет команд перехода (пп. 18.5 и 18.6).

Поскольку здесь все традиционно, я подробнее останавливаться ни на чем не буду. Как и всюду в учебнике, усвоение материала предполагается через решение учениками задач и упражнений, приведенных в конце параграфа.

Итак, § 17 — это физический уровень, уровень транзисторов, схем, вентилей, высоких, низких напряжений. § 18 — это следующий уровень, уровень машинных команд и программ в кодах.

§ 19. Составные части ЭВМ и взаимодействие их через магистраль

Параграф называется "Устройства ввода-вывода информации", потому что традиционно компьютер делится на процессор, память и все остальное. И это "все остальное" традиционно называется устройствами ввода-вывода информации.

Начало параграфа также вполне традиционно. Говорится, что есть такое устройство — монитор, по нему бегает луч, и точки экрана вспыхивают или гаснут так, что на экране образуется картинка. Есть клавиатура, принтер, дисковод.

Кстати, мы с удивлением обнаружили, что некоторые ученики считают магнитный диск квадратным (из-за внешней упаковки) и поэтому — специально — на цветной вклейке поместили фотографию дискеты с разорванной обложкой (фото 26). Я рекомендую вам показать эту фотографию ученикам, а еще лучше — вскрыть какую-нибудь старую и ненужную дискету и использовать ее в качестве наглядного пособия.

Также более или менее традиционным является и раздел "Взаимодействие основных частей ЭВМ. Магистраль" (п. 19.5, с. 162). В определенной степени это — повторение того, что было в учебнике [Ершов]. Единственное, на что здесь следует обратить внимание, — на управление устройствами ввода-вывода через команды записи и чтения по специальным адресам — адресам этих устройств. Если нужно что-то напечатать, то в машинных командах это выглядит просто как запись какой-то информации по специальному адресу. Когда такая команда — "записать по такому-то адресу число" — выполняется, т.е. на проводах магистрали появляются соответствующие высокие или низкие уровни напряжения, все устройства ЭВМ, включая память, анализируют этот адрес. Если указан адрес обычной памяти, то она — физическое устройство "память" — срабатывает и записывает число по соответствующему адресу. Если же, например, указан специальный адрес, приписанный принтеру, то память не срабатывает никак, а физическое устройство "принтер", обнаружив "свой" адрес, воспринимает число как команду принтера и, соответственно, что-то печатает. Аналогично физическое устройство "терминал" ("монитор"), обнаружив свой адрес, выполняет свою команду, например, выводит какой-то символ на экран, и т.д.

Одна и та же команда — "записать по такому-то адресу число" в зависимости от адреса может либо действительно записать число в память, либо (если адрес принтера) что-то напечатать или изменить параметры печати, либо (если адрес монитора) вывести что-то на экран, либо (если адрес клавиатуры) перевести клавиатуру в какой-то иной режим и т.п. Машинная команда одна и та же, а смысл и результат выполнения разные.

Дальше (с п. 19.6) начинаются особенности, связанные именно с нашим курсом и учебником, а именно — объяснение работы устройств ввода-вывода происходит с привлечением понятия "исполнитель". Вспомните схему "программного управления" (см. с. 13): человек пишет алгоритм, отдает его ЭВМ, а ЭВМ исполняет алгоритм и управляет исполнителями. Под словом "ЭВМ" в этой схеме понимается собственно ЭВМ — процессор и память. В рамках такой схемы все устройства ввода-вывода — это просто исполнители, которыми управляет ЭВМ. Исполнителями могут быть Робот, Чертежник, клавиатура, дисковод, принтер и пр.

Мы составляли алгоритмы на примерах управления Роботом. Но управление экраном, клавиатурой, принтером устроено аналогично и мало чем отличается от управления Роботом. Другая система команд, другие понятия, другие действия, но стиль взаимодействия тот же самый: мы пишем алгоритм, при выполнении которого ЭВМ отдаст необходимые команды соответствующим исполнителям и заставит их выполнить нужные действия.

Поэтому здесь чрезвычайно важно — и я хочу особо обратить на это ваше внимание — концептуальное единство построения нашего курса с точки зрения понятия "исполнителя". Важно, что все многообразие разных устройств ЭВМ — просто разные исполнители со своими системами команд. Это дает нам возможность изучать устройства не внешне (какого размера и цвета соответствующее устройство и как оно выглядит), а по содержанию (какова система команд исполнителя Экран? как им можно управлять из ЭВМ? и т.п.). И понимать, что управление этими исполнителями ничуть не сложнее, чем управление Роботом.

сание процесса производства транзисторов (приведены фотографические пластинки, показано, что на что напыляют, очень красивые — по-американски — цветные картинки, цветные стрелки, в общем, замечательно все нарисовано). И я вас отсылаю к этой книге, если вы захотите показать, как сделать МОП-транзистор и почему он именно так работает.

> МЕТОДИЧЕСКОЕ ОТСТУПЛЕНИЕ. Вы, впрочем, можете взять практически любой учебник по электротехнике (электронике) и найти там детальное описание работы электромагнитного реле — по сути, обычного выключателя, который включается и выключается также с помощью электричества. Такое реле даже можно собрать и продемонстрировать в классе (желательно на уроках физики).

Мы же (в учебнике) считали МОП-транзистор "базовым кирпичиком" и только описали, как он работает, не объясняя ни как его изготовить, ни почему он так работает. Школьнику, мы полагаем, интуитивно ясно, что столь простое устройство физически как-то изготовить можно. И сконцентрировали все внимание в этом параграфе на двух важнейших вещах: показать, как компьютер может хотя бы один бит информации обработать и как он в принципе может хотя бы один бит хранить.

Если отвлечься от замены реле на МОП-транзисторы, то этот параграф учебника вполне традиционен, соответствующий материал можно найти в любой книжке, посвященной устройству ЭВМ. В частности, в журналах "Квант" в свое время было достаточно много статей о том, например, как собрать сумматор из подобного рода переключателей. Материал учебника достаточно хорошо дополняется материалом в методпособии [Авербух], где, в частности, изложено, как из вентилей собрать сумматор и полусумматор (в учебнике этого нет), т.е. электронную схему, которая может ***складывать*** двоичные числа. Разбор работы такого сумматора, конечно, полностью демистифицирует, дегероизирует ЭВМ, показывает, что ничего "мистического", непознаваемого в ЭВМ нет. Как бы ни была сложна ЭВМ в целом, она "собрана" из частей, похожих на сумматор. И можно понять, как она в принципе работает.

Поскольку материал главы в целом достаточно традиционен, я только опишу основные цели и скажу, что в каком параграфе. А в остальном материал общепринятый, литературы много, мы почти ничем не отличаемся от других учебников, книжек, журнала "Квант" и прочего, разве что, может быть, последовательностью изложения материала.

Основные цели § 17.

Первое — показать, что электронная схема может в принципе производить формальное преобразование информации (как в игре в "черные ящики"). Продемонстрировать на чисто физическом уровне, на уровне напряжений на проводах, что соответствующие биты информации могут как-то обрабатываться, что, соединяя по-разному вентили, транзисторы и проч., мы можем получить разные схемы преобразования информации. Показать, как в принципе устроена обработка информации внутри компьютера, как и за счет чего работает процессор.

Второе — показать, что электронная схема может хранить информацию. Продемонстрировать схему устройства для хранения одного бита информации — триггер (с. 150 учебника), т.е. показать, как можно собрать триггер из вентилей. И затем продемонстрировать, что если определенным образом подать напряжение, а потом снять, то триггер запомнит и будет хранить 1, если же напряжение подать по-другому, то триггер запомнит и будет хранить 0. А после этого можно будет узнать ("прочесть"), какой именно бит (1 или 0) хранится в триггере.

Части параграфа "Взаимодействие процессора и памяти", "Поколения ЭВМ" и "Изготовление микросхем" менее важны. Хотя, например, понятия разрядности процессора, шины адреса и шины данных могут пролить свет на некоторые — в противном случае загадочные — особенности компьютеров. Я, однако, хочу обратить ваше внимание только на одну мелкую, но очень важную деталь п. 17.11 "Изготовление микросхем", а именно: что микросхемы изготовляются не поодиночке — за счет процесса, содержащего десяток операций и напоминающего фотопечать (см. [ЗК]), изготовляются одновременно миллионы микросхем. Вот почему при совершенствовании технологии компьютеры становятся все более мощными и в то же время все более дешевыми.

Вот и все про § 17. Теперь вспомним о "черепахе" курса — усвоить что бы то ни было можно только через работу. Поэтому для освоения материала параграфа ученики должны решать задачи и упражнения: составлять схемы из вентилей, реализующие те или иные операции обработки информации. В принципе задачи такого типа — это логические задачи с графическим изображением результата; решать их можно в любое время, поэтому допускается смещать материал параграфа (да и главы в целом) в широких пределах вперед или назад по курсу. Например, с целью смены характера проходимого материала, создания "перерыва" в изучении алгоритмизации.

> ЛИРИЧЕСКОЕ ОТСТУПЛЕНИЕ. Можно долго обсуждать, чему нужно и чему не нужно учить в школьном курсе информатики и какова, собственно, цель школьного образования. Одна из распространенных точек зрения — и мы ее разделяем — состоит в том, что общее образование должно создать у ученика какое-то цельное представление об окружающем мире. И коль скоро в этом мире появились компьютеры, алгоритмы, целая новая информационная индустрия, объяснением этой части мира на каком-то уровне должна заниматься и школьная информатика. И если в школьном курсе ни слова не сказать, как устроен компьютер внутри, то картина будет неполная и "кособокая".

§ 18. Команды и основной алгоритм работы процессора (программирование в кодах)

§ 18 посвящен описанию работы процессора на уровне элементарных машинных команд. Это то, что обычно называют программированием в кодах. Этот материал имелся и в самых первых учебниках А.П. Ершова (в начале второй части). Только там были и регистры, и виды адресации, и много всего остального. Мы все это убрали и на базе той же машины (PDP-11, или УКНЦ — с той же системой команд) оставили только простейшие команды, понятие счетчика команд и описание, как, за счет чего обеспечивается автоматизм работы процессора, как он переходит от выполнения одной команды к другой и как могут быть реализованы алгоритмические конструкции, например, цикл **пока**.

при выполнении алгоритма, мы рассуждаем совершенно по-другому, в какой-то совершенно иной плоскости.

Теперь нам осталось только показать, что цикл рано или поздно кончится, что алгоритм не "зациклится". Как это доказать? (Ведь мы не разбирались и даже не знаем, что и как будет происходить с величинами при выполнении цикла!) Мы начинаем с $k \ge 0$, а цикл кончается при $k = 0$. Но ***при каждом*** выполнении тела цикла k уменьшается либо вдвое (`k:= k/2`), либо на 1 (`k:= k - 1`). Значит, через какое-то время k станет равным 0 и цикл закончится.

Вот теперь действительно все. Мы показали, что:

а) выписанное нами условие — действительно инвариант;

б) ***если*** цикл закончится, то в b будет ответ — a в степени n;

в) цикл обязательно закончится. А значит, в b получится ответ. Наш алгоритм действительно считает a^n. Мы это доказали, хотя при этом мы даже не интересовались, ***как*** это получится, какой будет процесс, как будут меняться величины и пр. Алгоритм написан, его правильность доказана, а как он будет выполняться и что при этом будет происходить — это другой вопрос. Как выяснилось, этого мы можем даже и не понимать — и это нам совсем не мешает!

Вообще такое впечатление, как будто я вам показал какой-то фокус, не правда ли? "Ловкость рук, и никакого мошенства". Вроде бы и алгоритм написали, и доказали, что он правильный и считает, что надо, а все равно непонятно — как это? Как алгоритм это делает?

Заметьте, что этот эффект более или менее проявлялся при применении всех методов алгоритмизации. Алгоритмы получаются простые, но их связь с первоначальной постановкой задачи отнюдь не очевидна. Если бы я алгоритм подсчета максимума из сумм сразу выписал на доске, то тоже было бы не очень понятно, почему этот алгоритм дает правильный результат. А здесь мы не только сначала алгоритм выписали (а не составили!), но к тому же и правильность его доказали, не разбираясь полностью с тем, как он будет выполняться. Это еще одна иллюстрация к замечательным результатам применения методов алгоритмизации.

На самом деле я, конечно, должен выйти за рамки учебника и рассказать вам еще одну вещь. Я обязан это сделать, чтобы вы все-таки представляли себе реальное место и роль инварианта. Дело в том, что приведенный выше алгоритм, конечно, не "упал с неба". На самом деле я ***сначала*** придумал этот инвариант, а уж ***потом*** вывел из него алгоритм. Я обязан в этом признаться, иначе это действительно будет больше похоже на фокусы, чем на применение метода алгоритмизации при составлении алгоритмов.

Как на самом деле был составлен этот алгоритм?

Давайте я бегло вам изложу, как на самом деле был составлен этот алгоритм. Вначале был придуман инвариант. Точнее, кроме собственно инварианта и одновременно с ним были придуманы еще две вещи (это два ключевых требования при придумывании инварианта):

1) набор таких чисел, которые ***легко*** подставить в инвариант в самом начале, чтобы инвариант оказался выполненным (это $b = 1$, $t = a$, $k = n$, при которых инвариант превращается в тождество $1 \cdot a^n = a^n$). Наличие такого ***легко получаемого*** стартового набора — одно из основных требований к инварианту;

2) такие значения ***некоторых*** величин, при которых из инварианта получается ответ (это $k = 0$, когда инвариант превращается в равенство $b = a^n$). Получение из инварианта ответа — второе ключевое требование к инварианту.

После этого переход от набора величин `1` к состоянию `2` был представлен (воображен) ***как процесс*** последовательного уменьшения k: от $k = n$ до $k = 0$. В этот момент я еще ***не знал***, как именно мы будем уменьшать k, как будут меняться величины и пр. Я просто вообразил некоторый процесс уменьшения k, который от стартового набора величин ведет к ответу, ***никак не детализируя*** этот процесс, даже не зная еще, возможно ли это все в принципе. Обратите внимание: если я ***потом*** научусь, все равно как, уменьшать k, не меняя инварианта, то в конце при $k = 0$ я получу ответ.

Описанное выше "придумывание" — безусловно, творческое действие. Конечно, знание того, что такое инвариант, знание, что нужно придумать, а также некоторый опыт в придумывании инвариантов позволяют тратить на придумывание инварианта не больше творческих усилий, чем на придумывание алгоритма "в лоб". Но тем не менее это творческая деятельность.

А вот после этого началась "рутина" — просто последовательное (и достаточно "тупое") применение метода. Прежде всего, глядя на инвариант, надо было придумать, что будет ***шагом*** в воображаемом пока процессе перехода от старта к ответу, т.е. придумать такие действия, ***уменьшающие k***, которые бы сохраняли инвариант. Сначала я понял, что k всегда можно уменьшить на 1, а потом — что для четных k можно уменьшать k вдвое и при этом k будет уменьшаться намного быстрее. Глядя на инвариант и зная, что надо как-то так уменьшить k и изменить другие величины, чтобы инвариант сохранился, сделать это было уже нетрудно. Параллельно я получил формулы, которые мы писали выше, показывающие, как при уменьшении k надо изменить b и t, чтобы сохранить инвариант.

А уж после этого я просто записал приведенный выше алгоритм.

Заметьте, что мы, анализируя алгоритм, шли в обратном порядке. При практическом составлении алгоритма по методу проектирования цикла с помощью инварианта ничего доказывать не надо — все эти рассуждения и "доказательства" у нас появляются по ходу придумывания инварианта и тела цикла (шага процесса), по ходу составления (точнее сказать, "получения") алгоритма.

И — вспомните второе свойство инварианта ("если известно, как меняется один объект, то инвариант позволяет выводить, как должны меняться другие") — в реальности мы придумываем этот "один объект" и то, как он должен меняться (вначале в самых общих чертах, как, например, "уменьшение k"), а потом уж действительно в каком-то смысле "выводим" из инварианта, что делать с остальными объектами.

Конечно, можно придумывать разные инварианты, разные процессы получения одного и того же ответа из разных исходных "наборов" или по разным "путям" — и алгоритмы будут получаться разные.

Приведенный выше инвариант — отнюдь не "какой-нибудь", он не случаен и вообще не так прост. Это инвариант, описывающий так называемое "двоичное" возведение в степень — чрезвычайно быстрое и эффективное. Например, при $n = 1000$ в нашем алгоритме будет выполнено всего около 15 операций умножения (вы можете сами проверить это, а также — на досуге — найти связь между количеством операций умножения и двоичной записью числа n). По сравнению с тривиальным алгоритмом возведения в степень через цикл `n раз` (где таких операций

было бы 1000) наш алгоритм почти в 70 раз быстрее! И с ростом n эта разница в скорости только увеличивается.

Вернемся к учебнику.

Ну а теперь вернемся к учебнику. В рамках учебника изучение понятия инварианта цикла может происходить на двух уровнях.

Первый уровень — здраво осмысленный, наглядно-интуитивный. Просто полезно независимо от того, как мы используем инвариант и используем ли его вообще, записать статическое (неизменное) условие, описывающее взаимосвязь между меняющимися объектами (величинами). Это полезно для понимания того, что происходит в цикле, что будет, когда он несколько раз выполнится, и т.п. Это уровень, не содержащий никакой теории, вполне усваиваемый и достаточный для основной массы учеников.

Следующий, более высокий, уровень — использование инварианта как метода алгоритмизации при составлении алгоритмов. Здесь за рамками школы оставлен вопрос о том, как научиться придумывать инварианты. Но если инвариант задан или придуман по наитию (что, собственно, ничем не хуже, чем придумывание по наитию всего алгоритма), то его можно использовать для "вывода" тела цикла при составлении алгоритма, а также для проверки правильности всего алгоритма.

Этот второй уровень — придумывание сначала инварианта, а потом вывод из него алгоритма — кажется более сложным в основном из-за непривычности. Начинать составление алгоритма с придумывания инварианта можно, но для этого надо, чтобы голова немного "повернулась", да и культура нужна другая — уже не только алгоритмическая, но и логическая (математическая).

Именно из-за существенного задействования этой логической, математической составляющей мышления § 16 и кажется таким сложным. Да он и является, пожалуй, самым сложным в учебнике (вспомните диаграмму на с. 50). Здесь не обойдешься только здравым смыслом на уровне перемещения Робота по клеточному полю. Поэтому при преподавании в 7—8-х классах или в слабых классах § 16 можно пропускать.

Упражнения на инвариант цикла (это упр. 18, 19 на с. 139) выглядят так. Задан цикл, внутри которого написаны какие-то команды присваивания, а надо написать инвариант. Поскольку про сами циклы не говорится, что именно они считают, после выписывания инварианта школьники должны сказать, что же, собственно, этот цикл делает. До выписывания инварианта это непонятно (какие-то присваивания — мало ли что получится!). А когда инвариант написан, в него можно поставить условие окончания цикла — и получится ответ на вопрос, что именно считается в цикле.

Поскольку в этих упражнениях циклы ***уже*** написаны, а инвариант только еще надо придумать, то фактически это упражнения на первый уровень понимания инварианта. Инвариант как метод алгоритмизации, как метод составления алгоритмов остается за рамками учебника и за рамками упражнений. Инвариант в учебнике — это простенький прием "давайте опишем взаимосвязь в виде неизменного условия". И все.

16.4. Метод 4 — рекурсия

Последний — четвертый — метод этого параграфа (он помечен в учебнике звездочкой) называется "Рекурсия". Звездочка показывает и некоторую сложность в восприятии этого материала, и относительно меньшую важность рекурсии как метода алгоритмизации по сравнению с первыми тремя методами.

Еще одно замечание "вперед": если в случае с инвариантом кажется, что демонстрируется "ловкость рук, и никакого мошенства", то с рекурсией обычно вначале возникает полное ощущение того, что дурят нашего брата и все тут. Непонятно ни как вообще рекурсивный алгоритм может работать, ни что происходит при его выполнении.

По сравнению с учебником я переставлю материал и начну с содержательного примера. В учебнике сначала говорится, что надо быть осторожным, а потом уже — что это полезно. Я вначале продемонстрирую полезность, а потом уже покажу, что надо быть осторожным.

Задача, которая рассматривается в учебнике, выглядит так: где-то на клеточном поле внутри некоторой области, ограниченной закрашенными клетками, стоит Робот (*рис.* 2). Робот стоит в незакрашенной клетке, а "граница" области (т.е. закрашенные клетки) может быть устроена как угодно сложно. Известно только, что вся область у нас ограничена, т.е. по незакрашенным клеткам уйти "на бесконечность" нельзя. Задача состоит в том, чтобы всю эту незакрашенную область — и только ее — закрасить.

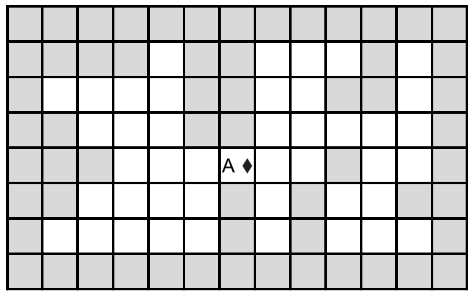


Рис. 2

МЕТОДИЧЕСКОЕ ОТСТУПЛЕНИЕ. Эту задачу очень легко проиллюстрировать на любом нормальном компьютере. Практически во всех графических редакторах есть такая возможность: когда на экране уже что-то нарисовано, есть какие-то линии, можно "ткнуть" мышкой в какую-нибудь область между линиями и "залить" ее определенной краской, определенным цветом. Причем "заливка" происходит только до границы области, до линии, как бы сложно она ни извивалась на экране.

Это в точности та же задача, только вместо клеток Робота здесь точки растра экрана (пиксели), а вместо границы из закрашенных клеток — граничная линия из точек экрана, отличающихся по цвету от точки внутри области, которую мы указали.

И ЕЩЕ ОДНО ОТСТУПЛЕНИЕ. Как всегда, чтобы лучше "прочувствовать", что дает применение метода, полезно сначала попробовать решить эту задачу "просто так", не читая учебник и не разбираясь с рекурсией. ***Если вы впервые читаете это пособие и еще не знакомы с рекурсией — отложите все книги и попробуйте просто решить задачу.*** Задайте ее ученикам (например, на дом), прежде чем вы приступите к изучению метода. Сравнение того, что при этом получится (у тех, кто все-таки умудрится решить задачу), с решением, приведенным в учебнике, — самый сильный аргумент в пользу изучения методов алгоритмизации.

распрямиться, и т.п. Этот исполнитель полезен при прохождении темы вспомогательных алгоритмов (т.е. еще до конструкций циклов и ветвления).

С другой стороны, по образцу этих задач можно предложить ученикам придумать своих исполнителей, посмотреть, быть может, они придумают что-то и интересное, и полезное.

Некоторые из таких исполнителей изображены на цветной вклейке в конце учебника: Двуног, взбирающийся по лестнице; Вездеход с локатором; Строитель — тот же Робот, но работающий на клеточной площадке в пространстве и умеющий перемещать "строительные кубики", и т.п.

Наконец, вы должны понимать, что мы ввели в учебник Робота и Чертежника потому, что с их помощью мы показали все, что хотели. Но, конечно, Робот — это не мировая константа, не число π. Исполнителей можно придумывать много. Вы, наверное, знакомы с исполнителем Муравей, который катает по клеточному полю кубики с буквами и которого использовал при преподавании информатики Г.А. Звенигородский. В системе "Роботландия" [Первин] этих исполнителей достаточно много и разных. В свердловском учебнике [Гейн] есть Чертежник-Вычислитель, чем-то похожий на Робота с хвостом. В нашем вузовском учебнике использовались "Редактор слова", Путник (похож на Робота, но имеет ориентацию) и целый ряд других исполнителей. Я уж не говорю о всем известной Черепашке в ЛОГО. Применение исполнителей как методический прием сейчас можно считать вполне распространенным и освоенным. Поэтому если вы в какой-то момент будете считать, что Робот в этом классе уже надоел, то вы можете сменить исполнителя, заменить его, скажем, на Путника, у которого есть ориентация. Замечу, что при этом многие задачи типа обхода препятствия можно будет записать, не повторяя четыре раза одно и то же с шагами в разные стороны (**вправо, вниз, вверх, влево**), а в виде цикла **4 раза** идти **вперед** и потом повернуться. Примеры вы можете найти в [ПдМ]. Алгоритмы при этом станут более компактными. Хотя сам Путник для восприятия несколько сложнее Робота.

И последнее замечание. Я очень вам рекомендую следующую задачу (ее нет в списке задач на повторение). Рассмотрим какой-нибудь алгоритм управления Роботом, например, алгоритмы А35—А37 из п. 9.14. Можно выбрать также и любые другие алгоритмы. Новая задача состоит в переделке этих алгоритмов таким образом, чтобы траектория движения Робота в ходе исполнения алгоритмов оказалась запомненной в виде литерной величины (по одной букве на каждое перемещение Робота). Кроме того, необходимо составить алгоритм, который, используя эту литерную величину, без каких-либо проверок вернет Робота обратно и повторит его движение вперед. Тогда при изучении материала про компиляцию, интерпретацию и кодирование алгоритмов из § 20—21 можно будет сослаться на решение этой задачи как на что-то уже пройденное и облегчить восприятие нового материала.

Лекция 9

Дальше мы переходим к главе 2, которая называется "Устройство ЭВМ" и состоит из четырех параграфов.

§ 17. Физические основы вычислительной техники

Здесь преследуются две основные цели — показать, как, с помощью каких физических процессов, можно: а) обрабатывать информацию и б) ее хранить, т.е. показать, как ***в принципе*** могут работать процессор (обработка информации, действия) и память (хранение информации, объекты) ЭВМ. Показать на чисто физическом уровне.

Естественно, показывается это на самых простых примерах хранения и обработки одного ***бита*** информации. Например, как собрать физическую схему, которая инвертирует бит (из 0 на входе делает 1 на выходе и наоборот). И как собрать физическую схему, которая обеспечит запоминание и хранение одного бита информации.

Этот материал более или менее традиционен. В старых учебниках применялись переключатели (электромагнитные реле), из которых, собственно, и были собраны самые первые ЭВМ. Мы использовали в качестве базового элемента более современные МОП-транзисторы (описав, как такой транзистор работает, но не разбирая, как именно его делают).

Если вы хотите объяснить, как сам транзистор работает, то имеется замечательная книга "Знакомьтесь: компьютер" [ЗК]. Эта книга замечательна по двум параметрам. Во-первых, она художественно сделана и легко читается, содержит массу разных занимательных историй. Ее можно просто с удовольствием читать на досуге, на ночь, лежа в кровати, и т.д. Во-вторых, из подобного рода занимательных книжек эта — единственная известная мне книга, в которой все изложено очень честно и которая полностью удовлетворяет нашему критерию "настоящести". Здесь нигде "не выплеснут ребенок". Все, что излагается, излагается подробно, содержательно и хорошо.

ЛИРИЧЕСКОЕ ОТСТУПЛЕНИЕ. Давайте я вам сразу расскажу про эту книжку все, что думаю, хотя остальное и не имеет отношения к данному параграфу. Когда мы только начинали писать учебник, у нас была идея сделать на нижней трети каждой страницы "подвальчики" с разными занимательными историями, интересными примерами, неформальными алгоритмами, шутками и пр. Чтобы этот материал, с одной стороны, тоже был, а с другой — чтобы он был явно отделен от изучаемого содержания курса.

Это — по разным причинам — не получилось. И в наших заготовках осталось довольно много примеров, которые мы хотели привести. Например, про самолет с обратной стреловидностью крыла (с крыльями, скошенными к носу, а не к хвосту). Такой самолет аэродинамически неустойчив, и человек без компьютера управлять им не может. Без компьютера такой самолет — все равно что обычный самолет без рулей: летит куда хочет, а то и просто валится. Были и другие интересные примеры, связанные с применениями компьютеров. А потом появилась книжка [ЗК], и мы с удивлением обнаружили в ней практически все примеры, которые хотели привести. И даже с изложением содержания применения компьютеров, например, схем управления указанным выше самолетом.

Я очень настоятельно рекомендую вам эту книгу — просто как необходимое дополнение к учебнику, чрезвычайно высококачественное, отвечающее всем требованиям нашего курса, плюс очень красочно и хорошо изданное и весьма занимательное.

Возвратимся к теме параграфа: в [ЗК] вы можете найти и описание (с картинками), как работает МОП-транзистор (вплоть до электронной и дырочной проводимости), и опи-

Задача 30. На поле Робота имеется область, “обнесенная забором” из стен. Известно, что Робот стоит вплотную у одной из стен “забора”. Надо определить, где стоит Робот, — внутри огороженного участка или снаружи. Вы видите — это обычная, простая формулировка в стиле всех задач про Робота. Однако сама задача уже вполне достойна сильных учеников.

Чем хороша эта задача? И в чем вообще отличительная черта задач для сильных? В том, что сразу не видно, как эту задачу решать. Глядя на картинку, на формулировку, нельзя сразу сказать, куда и как надо вести Робота, чтобы получить ответ.

На случай, если ученики окажутся недостаточно сильными и не смогут решить эту задачу и вам понадобится давать им наводящие указания, я вам сейчас в общих чертах обрисую, как ее можно решить. Для этого давайте вообразим себя Роботом. Если мы возьмемся левой рукой за стену и пойдем вдоль “забора”, все время касаясь его левой рукой и считая, сколько раз и куда (направо или налево) нам придется поворачивать, то при возвращении в исходную клетку “с другой стороны” (после обхода всего “забора”) у нас будет либо левых поворотов сделано на 4 (т.е. на один полный оборот) больше, чем правых, либо, наоборот, правых поворотов будет сделано на 4 больше, чем левых. Значит, при таком обходе мы один раз повернемся либо в одном направлении (через левое плечо, левых поворотов больше — при обходе “забора” снаружи), либо в другом (через правое плечо, правых поворотов больше — при обходе “забора” изнутри). Поэтому, посмотрев, каких поворотов было сделано больше, можно сказать, снаружи или изнутри огороженного участка мы находимся.

Конечно, это только идея решения, которую еще надо превратить в алгоритм, ведь у Робота нет ни рук, ни лица, он не умеет поворачиваться вокруг собственной оси ни налево, ни направо. Значит, все эти понятия придется имитировать с помощью величин алгоритмического языка, например, завести величину “ориентация” (т.е. куда направлено воображаемое “лицо” Робота) (кстати, для задания ориентации на плоскости можно использовать аналогию с часами: **0** — направление вверх, **3** — вправо, **6** — вниз, **9** — влево). Тогда поворот направо (по часовой стрелке) запишется как **ориентация:= mod(ориентация + 3,12)**, а поворот налево — как **ориентация:= mod(ориентация – 3,12)**.

Как, например, узнать, что Робот вернулся в исходную клетку? Можно считать по дороге суммарное смещение Робота от начальной клетки по горизонтали (по оси *X*) и по вертикали (по оси *Y*). При шаге вправо увеличивать смещение по *X* (**x:= x + 1**), а при шаге влево — уменьшать (**x:= x – 1**). Аналогично для шагов вверх и вниз (**y:= y + 1** и **y:= y – 1**). И если мы в какой-нибудь момент обнаружим, что **x = 0** и **y = 0**, то это и будет означать, что Робот в исходной клетке.

Придется также составить вспомогательный алгоритм “шаг вдоль стены”, который при повороте забора влево заставит Робота сделать два шага (*рис.* 5), при повороте забора вправо (конечное положение Робота на *рис.* 5) просто “повернет” Робота, оставив его на той же клетке (т.е. изменит значение величины “ориентация”). Другими словами, “шаг вдоль стены” должен так изменить ориентацию и положение Робота, чтобы воображаемая “левая рука” Робота сместилась к следующему одноклеточному участку стены.

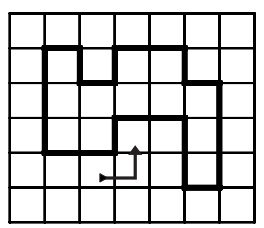


Рис. 5
Шаг “вдоль стены” при повороте “забора” влево

Задача 31 существенно сложнее, она недаром помечена звездочкой. Мы обычно называем ее задачей “о налоговом инспекторе”. Налоговый инспектор Робот подошел к земельному участку, огороженному неприступной стеной. Чтобы определить, правильно ли платится налог за землю, Робот должен подсчитать площадь земельного участка. Но, поскольку участок частный, а частная собственность неприкосновенна (это все не у нас, конечно, происходит, а в каком-то воображаемом правовом государстве), заходить внутрь участка даже налоговый инспектор права не имеет. Поэтому надо, как-то бродя снаружи вдоль стен участка, умудриться подсчитать площадь участка.

Эта задача намного сложнее, и даже идею ее решения, как это было в предыдущей задаче, я вам кратко изложить не могу. Точнее, одну идею, идею одного из решений, я могу изложить — “надо взять интеграл по контуру и, используя формулу Грина, получить площадь”. Но эта идея апеллирует к теоремам и понятиям дифференциальной геометрии и потому вряд ли будет полезна в обычной школе. И хотя на основе этой идеи можно написать короткий алгоритм и даже объяснить, почему он дает правильную формулу, придумать такой алгоритм чрезвычайно трудно.

Если эту задачу задать даже сильным ученикам, то скорее всего они будут мучиться, но вряд ли сами ее правильно решат. И это позволяет учителю сказать: “Видишь, тот же Робот и клеточное поле, а задача не решается. Значит, суть не в Роботе, а в том, умеешь ты решать задачи или не умеешь”.

И еще и еще раз повторю: в нашем курсе очень важна ценностная ориентация. Умение решать задачи должно быть поставлено выше технических знаний и навыков. Если этого не сделать, если школьники сидят и ждут, когда им разрешат сыграть на компьютере, когда им расскажут про операционную или файловую систему и т.п., то у них возникнет проблема “обманутых ожиданий”. Проблема несоответствия содержания курса их ожиданиям. Желательно с самого начала это погасить. Сказать, что курс содержательный, что мы будем учиться решать задачи. А чтобы не тратить время на изучение особенностей Бейсика или Паскаля, решать будем задачи про Робота на школьном алгоритмическом языке (на компьютерах — в системе КуМир).

Учитель, конечно, может преподавать и без Робота и школьного языка. Но тогда нужно будет их чем-то заменить, причем заменить так, чтобы не потерять алгоритмическую сложность решаемых задач, чтобы решение задач не оказалось заменено изучением чего-нибудь типа многочисленных замечательных возможностей команды **PRINT** в Бейсике.

В заключение хочу еще обратить ваше внимание на задачи 32, 33 и 34. Соответствующие исполнители реализованы в системе КуМир, и вы можете прямо на компьютере решать задачи по управлению этими исполнителями. Наибольший интерес у школьников обычно вызывает Двуног, состоящий из, естественно, двух ног и тяжелой головы на очень длинной шее. Когда голова наклонена, Двуног начинает медленно падать. В этот момент можно, скажем, выставить ногу, упасть на нее, распрямить шею, выставить вперед другую ногу, наклонить шею и опять начать падать. Типовая задача — “научить Двунога ходить”, т.е. составить соответствующий алгоритм, при выполнении которого Двуног пойдет. Сначала по ровному месту, потом по лестнице, потом в низкой пещере, где нельзя полностью

В чем сложность этой задачи? Посмотрите: в положении, изображенном на *рис.* 2, надо закрасить области и справа, и слева от Робота. Если мы закрасим клетку, в которой стоит Робот, и пойдем, например, направо, то скорее всего уже никогда не узнаем, что область слева первоначально входила в закрашиваемую зону, потому что не сможем отличить закрашенную клетку А от клеток границы.

Так же, как и с инвариантом, предлагаемое здесь рекурсивное решение в каком-то смысле “выводится” из доказательства того, что оно будет работать. Поэтому мы начнем с доказательства.

Итак, Робот стоит в некоторой клетке области. Клетки, которые он должен закрасить, — это клетки, до которых Робот может дойти, если пойдет только по незакрашенным клеткам. На старте — в общем случае — Робот может пойти в одну из четырех сторон (вниз, вверх, вправо, влево), поэтому множество клеток, до которых он может дойти из А по незакрашенным, — это сама клетка А, а также те клетки, до которых можно дойти из четырех клеток, прилегающих к А. Естественно, если эти четыре клетки не закрашены. Если же часть из них закрашена, то надо рассматривать только незакрашенные. Например, на *рис.* 2 таких клеток две — слева и справа от А.

Таким образом, задача закрашивания всех клеток, до которых можно дойти из А (по незакрашенным клеткам), сводится к подзадачам закрашивания всех клеток, до которых можно дойти из соседних с А незакрашенных клеток. Если для решения подзадач использовать вызов вспомогательного алгоритма “закрасить область”, закрашивающего все клетки, до которых можно дойти от той клетки, где стоит Робот, то получится алгоритм А89 со с. 135 учебника:

```
                                  (А 89)
алг  закрасить область
дано на поле Робота стен нет;
     число клеток, в которые Робот
     может попасть из исходного
     положения, двигаясь только по
     незакрашенным клеткам, конечно
надо закрашены все клетки, в которые
     Робот мог попасть из исходного
     положения по незакрашенным
     клеткам;
     Робот в исходном положении
нач
  если клетка не закрашена то
  | закрасить
  | вверх; закрасить область; вниз
  | вправо; закрасить область; влево
  | вниз; закрасить область; вверх
  | влево; закрасить область; вправо
  все
кон
```

Я думаю, вы уже заметили, что этот алгоритм вызывает сам себя (именно это и называется рекурсией, или рекурсивным вызовом алгоритма). Что алгоритм будет работать, доказывается приведенным выше рассуждением: все клетки, которые надо закрасить, — это клетка А и клетки незакрашенных областей, вплотную примыкающих к А с одного из четырех направлений.

Поскольку алгоритм вызывает сам себя, надо отдельно показывать и доказывать, что он не будет это делать вечно (не “зациклится”). Это можно сделать так: алгоритм сначала закрашивает незакрашенную клетку, а уж потом вызывает сам себя, поэтому перед каждым таким вызовом число незакрашенных клеток будет уменьшаться. Вначале число незакрашенных клеток, которые надо закрасить, ограничено, следовательно, рано или поздно вызовы алгоритмом самого себя прекратятся и, значит, выполнение алгоритма рано или поздно закончится.

Мы опять написали алгоритм, как-то логически показали, доказали, что он будет работать правильно и что все окончится. А как будет при этом Робот двигаться, куда он пойдет и в каком порядке будет красить клетки, этого мы пока не знаем. Согласитесь, что до § 16 нам и в голову не могло прийти, что можно составлять алгоритм, абсолютно не разбираясь с тем, что он будет ***делать***, что будет происходить при его ***выполнении***, как и куда будет двигаться Робот.

Итак, рекурсия как понятие — это вызов алгоритмом самого себя. Использование таких вызовов при составлении алгоритмов и есть существо рекурсии как метода алгоритмизации. Сложность здесь обычно только в одном — в восприятии. От этих вызовов алгоритмом самого себя голова может пойти кругом и возникнет либо ощущение порочного круга, либо просто непонимание.

Но давайте вспомним: раньше были основные и вспомогательные алгоритмы. Основной алгоритм вызывает вспомогательный алгоритм — это понятно. При вызове алгоритма в памяти отводится место, начинается выполнение алгоритма и т.д.

В случае рекурсии происходит все то же самое. При вызове из алгоритма “закрасить область” самого себя в памяти ЭВМ отводится место под ***еще один*** алгоритм “закрасить область” (принято говорить: под “второй”, “третий” и т.д. ***экземпляр*** алгоритма — см. *рис.* 3).

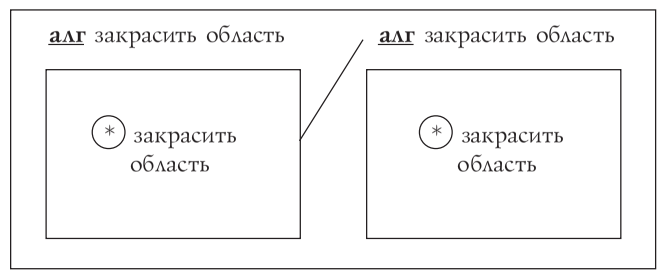


Рис. 3

То есть ЭВМ работает, как всегда, — при каждом вызове алгоритма “закрасить область”, сколько бы раз он перед этим уже ни вызывался, в памяти отводится ***новое место*** — и начинается выполнение вызванного алгоритма (например, 25-го экземпляра), а выполнение алгоритма, из которого произошел вызов (в нашем примере — выполнение 24-го экземпляра), приостанавливается на команде вызова до полного окончания выполнения вызванного — теперь надо говорить “экземпляра” — алгоритма. Происходят такие вызовы, вызовы, вызовы и вызовы все новых экземпляров алгоритма.

Можно представить себе каждый экземпляр алгоритма написанным на отдельном листке и кипу таких листков с одним и тем же текстом — текстом алгоритма “закрасить область”. Тогда работу ЭВМ можно описать так. При выполнении очередного вызова очередного экземпляра алгоритма ЭВМ выполняет команды по соответствующему листку, пока не доходит до вызова “закрасить область”. В этот момент ЭВМ “ставит галочку” напротив этого вызова, “откладывает листок”, берет ***новый*** листок с тем же текстом и начинает его выполнять с самого начала. Образуется та-

кая стопка отложенных листков (в информатике она называется “стеком”, потому что каждый следующий листок кладется сверху предыдущего). Если в какой-то момент очередной экземпляр алгоритма, очередной листок, удастся выполнить до конца (например, если клетка окажется закрашенной — тогда наш алгоритм ничего не делает), то в таком случае этот листок выбрасывается (т.е. этот экземпляр алгоритма стирается из памяти ЭВМ), из стопки берется последний отложенный листок — и ЭВМ продолжает его выполнение с места, отмеченного галочкой.

Это очень важно — представлять себе, ***как*** ЭВМ выполняет рекурсивные вызовы алгоритмов, чтобы понимать, как вообще работает рекурсия и что при этом происходит.

И здесь я плавно перейду ко второй части — рекурсия бывает не только полезна, но и опасна. Дело в том, что обычно, как и в примере выше, при составлении рекурсивного алгоритма у нас возникает “доказательство” его правильности и того, что он рано или поздно закончится. Но при этом не возникает никакого представления о том, какие действия и как будут выполняться. В этом и состоит опасность. Ведь формальной правильности и завершаемости недостаточно. Что значит “выполнение алгоритма рано или поздно закончится”? — ведь алгоритм может оказаться ужасающе неэффективным. Вряд ли нас устроит доказательство того, что алгоритм “когда-нибудь” кончится, если реально для его выполнения понадобятся сотни лет работы на самых совершенных компьютерах.

Для демонстрации этой опасности, для демонстрации того, что очень простой рекурсивный алгоритм может оказаться ужасающе неэффективным, в учебнике используется рекурсивный алгоритм А88, вычисляющий (уже который раз!) n-й элемент последовательности Фибоначчи (с. 134 учебника). Этот алгоритм записан в точности по определению последовательности Фибоначчи: $\Phi(n) = \Phi(n-1) + \Phi(n-2)$. Именно поэтому он гораздо проще всех наших предыдущих алгоритмов для последовательности Фибоначчи. Но как он будет выполняться? Пусть, для примера, мы стали с его помощью вычислять $\Phi(45)$. Это значит — смотрите на текст алгоритма, — что выполнятся вызовы $\Phi(44)$ и $\Phi(43)$, т.е. наш алгоритм Ф будет дважды вызван рекурсивно, но уже с другими аргументами. Соответственно, алгоритм $\Phi(45)$ будет “отложен” и начнет выполняться, скажем, $\Phi(44)$. При выполнении $\Phi(44)$ надо будет вычислить (рекурсивно вызвать) $\Phi(43)$ и $\Phi(42)$. При вычислении $\Phi(43)$ — вызвать $\Phi(42)$ и $\Phi(41)$ и т.д. (*рис.* 4).

Вы видите, что $\Phi(42)$ вызывается уже три раза (в разных частях картинки). Если рисовать картинку дальше, то к моменту, когда мы дойдем до $\Phi(1)$ и $\Phi(2)$ (которые вычисляются явно, т.е. не порождают новых рекурсивных вызовов), у нас на картинке будет уже больше миллиарда разных $\Phi(\ldots)$.

ЛИРИЧЕСКОЕ ОТСТУПЛЕНИЕ. Вы помните старинную задачу про мудреца, который попросил на первую клетку шахматной доски положить одно зерно, на следующую — два, на следующую — четыре и т.д.? На каждую следующую — в два раза больше, чем на предыдущую. Получаются степени 2 (на последнюю клетку надо положить 2^{63}), а степени двойки чрезвычайно быстро растут.

У нас почти то же самое — каждый следующий уровень на *рис.* 4 содержит вначале вдвое больше вызовов Ф, чем предыдущий (более высокий) уровень.

Таким образом, если мы начнем вычислять по приведенному рекурсивному алгоритму $\Phi(45)$, то будет выполнено свыше миллиарда операций сложения. А ведь в рекуррентном алгоритме А82 этих операций было всего 45. Значит, рекурсивный алгоритм будет выполняться ***в несколько десятков миллионов раз*** медленнее! И это не считая затрат времени на постоянное “откладывание” алгоритмов, затрат памяти ЭВМ при отведении места под вызываемые алгоритмы и т.п.

Вы видите: такая простейшая задача, такой простой алгоритм — и какая жуткая неэффективность. Поэтому, повторю еще раз, рекурсией надо пользоваться с осторожностью. Недостаточно просто написать рекурсивный алгоритм. Надо представлять себе, как он будет выполняться, — не возникнет ли в процессе выполнения эффекта удвоения (или утроения, или учетверения) объема работы, объема вычислений.

Если рекурсивный алгоритм вызывает сам себя только один раз (в одном месте), то, как правило, это неопасно. Аналогичный *рис.* 4 рисунок будет содержать линейную цепочку вызовов — и никакого удвоения или утроения не произойдет. Но если алгоритм вызывает себя несколько раз (алгоритм А88 вызывает себя дважды, алгоритм А89 — четыре раза), то это потенциально очень опасно. И надо отдельно разбираться и понимать, как алгоритм будет выполняться и не начнет ли он помногу раз вычислять одно и то же.

Итак, рекурсия — это несколько экзотический, потенциально опасный, но иногда полезный метод алгоритмизации. Существуют задачи, которые рекурсивно решить легко, а без рекурсии — очень трудно. Закраска области (на поле Робота или на экране) — одна из таких задач. Есть и другие. Попробуйте, например, составить алгоритм обхода конем шахматной доски (конь должен побывать в каждой клетке ровно по одному разу) без использования и с использованием рекурсии — и, как говорится, “почувствуйте разницу”.

В целом рекурсия обычно применима и полезна, когда после первого хода, или шага, или еще чего-то мы сводим исходную задачу к другой ***точно такой же*** задаче (подзадаче), но с другими данными. Именно это мы проделали в задаче закраски области. Задачи, которые легко свести к другим точно таким же, но с другими данными, как правило, легко и просто записываются рекурсивно. Но, повторю еще раз, простота записи не означает в данном случае простоты и эффективности работы алгоритма. Это надо смотреть отдельно. Поэтому, хотя рекурсия и позволяет очень легко записывать такие алгоритмы, использовать ее надо аккуратно и с осторожностью.

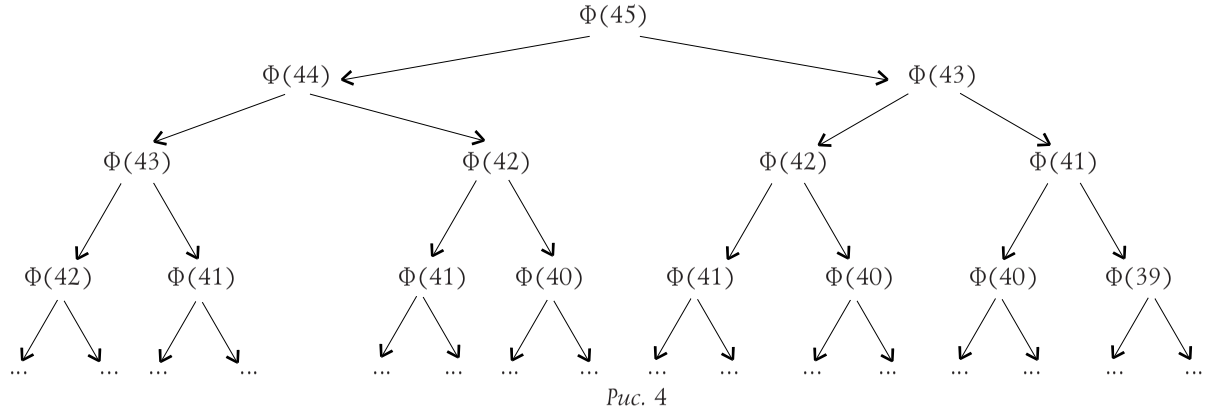


Рис. 4

ЛИРИЧЕСКОЕ ОТСТУПЛЕНИЕ. В заключение я еще скажу несколько слов о нашем личном отношении к рекурсии. Мы включили этот метод в учебник, потому что существует такое течение в информатике, такая достаточно большая группа ученых, которые считают рекурсию основным и главным методом информатики. Считают, что даже циклы надо реализовать через рекурсивные вызовы алгоритмов, а конструкций циклов в языке быть не должно.

Базой для этого течения является математическая теория, называемая аппаратом рекурсивных функций, в которой доказывается, что таким образом можно подсчитать все, что угодно.

Соответственно, имеются языки программирования, построенные на том, что именно так и надо работать, языки, в которых нет циклов, в которых циклы имитируются через рекурсию. Кстати, небезызвестный язык ЛОГО (если отвлечься от черепашьей графики и посмотреть на сам язык программирования) — один из таких языков. Цикла **пока** в нем уже нет, а если мы хотим повторять какие-то действия, пока будет выполнено некоторое условие, то это надо делать через **если** и рекурсивные вызовы.

Поскольку это течение, считающее рекурсию одной из самых важных и базовых вещей в информатике, достаточно представительно, особенно среди математиков, мы решили этот метод в учебнике изложить — хотя бы со звездочкой.

Кроме того, когда мы обсуждали включение рекурсии в школьный курс с Андреем Петровичем Ершовым, то он сказал, что, на его взгляд, это очень важная составляющая алгоритмической культуры и в каком-то виде рекурсия в школьном курсе должна быть представлена обязательно. И мы, конечно, постарались это мнение учесть.

Но, повторю, сами мы считаем рекурсию просто одним из методов алгоритмизации — методом, безусловно, полезным, хотя и с достаточно ограниченной областью применения. И полагаем, что пользоваться им надо осторожно и аккуратно.

На этом § 16, а с ним и вся первая глава заканчиваются. На самом деле если вы теперь посмотрите на первую главу целиком, то увидите, что состоит она из двух разных разделов: изложение собственно алгоритмического языка заканчивается § 15, а в § 16 содержится совершенно другой материал — методы алгоритмизации (хотя параграф назван не так, а попроще — “составление циклических алгоритмов”). В реальности это отдельный раздел информатики, но в учебнике он не выделен.

В § 16 изложены четыре метода алгоритмизации. Эти методы различны и по сложности, и по применимости. На школьном уровне наиболее применим, по-видимому, метод рекуррентных соотношений. В школе (и в математике, и в физике) достаточно много задач, решаемых этим методом. Следующим идет метод однопроходных алгоритмов, применимый к огромному количеству типовых задач информатики. Все эти задачи вычисления максимальных, минимальных и среднеарифметических величин, как и практически любая другая задача, вычисления чего-то по таблице, могут быть решены этим методом и за один проход. Содержательное применение метода использования инварианта цикла осталось за рамками учебника как несколько более сложное. А на рекурсию вы можете смотреть как на достаточно редкий, экзотический метод, применимый к некоторым специфическим задачам (и именно поэтому она отмечена звездочкой).

Более глубокое и содержательное изложение этих, а также некоторых других методов алгоритмизации можно найти в нашем вузовском учебнике [ПдМ]. Поскольку учитель должен владеть предметом глубже, чем изложено в школьном учебнике, я вам настоятельно рекомендую потратить свое время и разобраться в методах алгоритмизации более глубоко. Хотя это требует усилий и некоторого уровня математической культуры.

Упражнения на повторение к главе 1

Первый раздел заканчивается так называемыми “упражнениями на повторение” (с. 142 и далее). На самом деле эти “упражнения на повторение” никакими упражнениями на повторение не являются. Это просто набор упражнений, которые можно решать, пройдя всю первую главу. Среди них довольно много задач, взятых из предыдущих учебников Андрея Петровича Ершова, — задач математического плана, где надо что-то вычислить, но произвести при этом не более какого-то заданного числа операций.

Единственное мое общее замечание сводится к тому, что это не есть задачи на повторение — это просто набор задач. Вы вполне можете использовать их в процессе изучения первого раздела, выбирая и давая задачи сильным ученикам.

Кроме того, я хочу обратить ваше внимание на задачи, которые мне кажутся наиболее полезными и важными. Это задачи 30 и 31 на с. 145.

Предположим, у вас в классе сидит какой-нибудь умный мальчик и говорит: “Что мы все какой-то ерундой занимаемся, все Робота гоняем. Когда же мы наконец займемся делом?”

Вы помните, я говорил, что наш курс построен на том, что надо научить решать задачи. Совершенно не важно — с этой точки зрения, — умеет ли ученик нажимать на кнопки. (Конечно, если курс компьютерный, то надо уметь это делать, но даже в этом случае умение нажимать на кнопки, знание КуМира, Бейсика, Паскаля, операционной системы не входят в основные ***цели*** курса.)

Посмотрите, сколько мы уже всего разного — и очень содержательного — прошли, а про то, как работать на компьютерах, ни слова не сказали. Поэтому для изучения нашего курса чрезвычайно важно создать правильную ценностную ориентацию, чтобы школьник понимал, что умение решать задачи важнее, чем знание тех или иных операторов и скорость нажатия на кнопки. Умение думать значительно важнее знания каких-то технических деталей. Чтобы эту ценностную ориентацию создать, ученикам надо предлагать задачи, достойные их уровня, иначе ничего не получится. Если сильному ученику долго задавать простые задачи вроде “довести Робота до стены”, то, конечно, он будет считать это ерундой и игрушками для маленьких.

Поэтому сильным ученикам надо предлагать достойные их задачи — это с одной стороны. А с другой стороны, задачи должны быть на поле Робота, чтобы показать, что дело не в Роботе и не в формулировке. Надо просто уметь решать задачи.

И здесь очень полезны задачи 30 и 31.

МОДЕРНИЗАЦИЯ КОМПЬЮТЕРНОГО КЛАССА

Сжатый диск в Windows 95

Продолжение. Начало на с. 1

то, что требовалось, и прежде всего, конечно, пакет Microsoft Office. Никаких особых проблем здесь не встречается, единственное затруднение — это разобраться, что тебе действительно нужно из состава этого пакета, а что нет. Тут, собственно, и раздумывать долго не пришлось бы, а просто брать все подряд, целиком весь пакет, если бы...

Если бы винчестер был емкостью в несколько гигабайт. Тогда — да, можно пойти на полную установку, но у меня ситуация иная: действительно, гигабайта четыре, может, и наберется, да только на всех 12 компьютерах, вместе взятых! А так, по отдельности, — где-то винчестеры по 350, где-то — по 430, а на двух машинах пусть даже и по 500 мегабайт — все равно, по нынешним меркам этого едва хватает. К тому же диск C, на который обычно производится установка, составляет хотя и большую, но все-таки какую-то часть этого объема — 255 Мб[1]. И, между прочим, около пятидесяти мегабайт здесь занято уже с самого начала, даже если никаких других программ еще нет, — самой системой. Так что в моем распоряжении всего лишь чуть более двухсот мегабайт на каждом рабочем месте. Причем это еще только в том случае, когда система — оригинальная Windows 95. А я вообще-то уже перешел на ее обновленный вариант OSR2, включающий в себя некоторые дополнительные компоненты, среди которых самым значительным, конечно, является Internet Explorer[2]. Таким образом, на диске C свободного места остается еще на 30 Мб меньше, т.е. 170 мегабайт на все про все.

Надо уплотнять диск

Этого, конечно, очень мало. Обязательно надо уплотнять диск, причем сделать это удобно именно тогда, когда на нем, кроме системы, еще ничего нет. Я уже рассказывал[3], как производится уплотнение при работе в среде MS DOS, теперь пришла пора коротко описать ту же процедуру, но для среды Windows 95. Идеология прежняя: реальный диск C будет переименован какой-нибудь другой латинской буквой (я, например, выберу букву G) и на нем вместо множества файлов будет в конечном итоге создан один огромный файл — почти на весь объем диска в 255 мегабайт. Внутри него, этого файла-гиганта, информация из всех отдельных маленьких первоначальных будет записана на долговременное хранение, но в специальном, очень плотном, как бы архивном формате, заняв в общей сложности гораздо меньше места. Когда мне содержимое того или иного из первоначальных файлов потребуется, пусть это будет, например, текст, или рисунок, или даже программа, то только он один и будет разархивирован с диска в оперативную память для работы. Или, наоборот, если я создам новый документ и захочу сохранить его на уплотненном диске, он будет добавлен в архив, естественно, в сжатом виде.

Все это уплотнение-разуплотнение производится автоматически, без каких-либо особых указаний с моей стороны, самой системой, причем очень быстро, практически мгновенно. При этом я вообще не буду обращаться напрямую к реальному диску G, даже могу и вовсе не знать о его существовании. Почему? Да потому, что для моего удобства этот самый огромный файл-архив, точнее говоря, его содержимое, будет представлено мне на экране не в виде огромного файла, а в образе привычного (на самом деле фиктивного) диска C, только вдвое примерно большего объема — не 255, а около 500 мегабайт. Я буду видеть все свои файлы с указанием их номинального размера в байтах, совсем не задумываясь о том, где они там находятся в архиве и сколько в нем реально занимают места. Для меня это будут просто файлы на диске C. Если я их стану копировать, скажем, на дискету, то на нее они запишутся в своем настоящем, неуплотненном, виде. Если, наоборот, захочу переписать что-нибудь с дискеты, то снова обращусь к диску C, хотя реально информация будет сжата и помещена в архивный файл на диске G.

Такой подход мне очень по душе, и единственно, чего я могу опасаться, — это что в какой-то прекрасный день отлаженная схема даст сбой, файлы перестанут читаться и как следствие вся система "рухнет". Ну что же, и такое бывает. А что, есть какая-нибудь альтернатива? Как говорится, не от хорошей жизни я иду на уплотнение. Да и страхи, как показывает мой многолетний опыт, необоснованны. Правда, случилась у меня дважды такая катастрофа — один раз на работе (еще в DOS), другой раз дома (уже в 95-м). Но виноват в обоих случаях я сам: был невнимателен к сообщениям системы и собственными руками губил все дело. Последний раз — совсем недавно, нынешней зимой; и по горячим следам расскажу, как это произошло. История, кстати, поучительная.

Не та версия программы

Начну издалека: в декабре месяце я приобрел модем. Модем внутренний, надо, значит, вставлять его в разъем материнской платы, следовательно, снимать корпус с системного блока. Дело привычное: на работе, в компьютерном классе, то на одной машине что-то сломается, то на другой. А вот дома как собрал я компьютер из комплектующих в прошлом году, так до сих пор и не было повода заглядывать внутрь, поскольку все работало нормально. Но тут волей-неволей пришлось. И раз уж так, то заодно я еще одну вещь решил попробовать. Дай-ка, думаю, процессор "разгоню", многие так делают — и ничего, хорошо получается. Он у меня имеет официальную частоту 166 Мгц, а я попробую выставить побольше.

Сказано — сделано. Побольше можно получить двумя способами: либо частоту самой платы поднять с положенных 66 мегагерц до 75 или даже до 83 (по ее паспорту это разрешается), либо коэффициент умножения для процессора увеличить, вместо значения 2,5 взяв, например, 3,0. И то и другое делается с помощью специальных перемычек-джамперов на плате. И будет у меня тогда, допустим, чтобы сильно не рисковать, $75 \times 2{,}5 = 188$ Мгц, или $66 \times 3{,}0 = 198$ Мгц вместо стандартной комбинации $66 \times 2{,}5 = 166$ Мгц. Риска в общем-то никакого и нет, ничего не сгорит, в крайнем случае компьютер просто зависнет из-за перегрева процессора, тогда я назад вернусь. Между прочим, первый вариант даже предпочтительнее второго, поскольку хоть 188 и меньше 198, но зато тут не только один лишь процессор побыстрее заработает, а и вся остальная периферия, связанная с платой, и память, и видеоплата, и винчестер.

С этого моего решения и потянулась цепочка, приведшая к аварии. Впрочем, еще ничего не предвещало какой-либо беды (в смысле работы с уплотненным диском); ее бы и не случилось при спокойном состоянии духа. Но меня охватил азарт: модем отошел в сторону, было уже забыто, зачем открывался компьютер. Выставив частоту платы на 75, я включил питание и сразу усмотрел в табличке с параметрами системы, которая выводится на экран при загрузке, желаемые 188 Мгц. Но долго радоваться не пришлось, поскольку стоило лишь появиться заставке Windows 95, как машина остановилась, сообщив напоследок о какой-то фатальной ошибке. Ага, подумал я, раз с Windows что-то не ладится, то не попробовать ли работу в MS DOS, причем не в режиме эмуляции, а ради чистоты эксперимента в старом добром 6.22, загрузившись с системной дискеты, которую я на всякий случай храню еще с тех времен. Это было вторым опасным шагом.

Дело в том, что один из трех логических дисков — диск E, где я держал все свои статьи, таблицы и тому подобные документы, как раз был сжатым. Причем уплотнение производилось, естественно, еще до всех этих попыток разгона, программой-компрессором DriveSpace в версии для Windows 95. А на дискете-то присутствовала более ранняя ее версия, из системы MS DOS 6.22. И что же получилось? Старый компрессор, вступив в работу при загрузке с дискеты, обнаружил на винчестере тот самый архивный файл[4], о котором я говорил, и, как ему и положено, постарался информацию о множестве хранящихся там уплотненных документов представить в виде фиктивного диска большого объема. Но из-за различия в версиях ему это не удалась сделать. По-видимому, сама структура файла, создаваемого новым компрессором, не такова, с какой может работать старый компрессор. Вот если бы наоборот: диск был уплотнен старой версией, а работать с ним довелось бы новой, ведь именно такую ситуацию следует считать нормальной, — то все было бы в порядке.

Поэтому DriveSpace вывел на экран сообщение о том, что он не может присоединить сжатый том и предлагает вызвать на помощь программу Scandisk, которая пусть сначала проверит его. Хорошо, пусть. Такая программа на системной дискете у меня тоже имелась, и я запустил ее. Но при этом совсем не учел, что ведь и она не из состава Windows 95, а из DOS 6.22! Как будто так и надо, она приступает к проверке и сообщает, что, по ее мнению, файл сжатого тома поврежден, и предлагает его исправить. Якобы что-то там напутано в таблице размещения файлов (на самом деле ничего там не напутано, а просто сделано по-новому, как — старая версия программы не понимает). И даже здесь еще не поздно было повернуть назад, правда, это уже в последний раз.

Scandisk — программа очень осторожная и никогда не осмелится на несанкционированные действия. И, конечно, она запрашивает моего разрешения на такое, можно сказать, хирургическое вмешательство. Причем из соображений безопасности мне даже предоставляется возможность зафиксировать на отдельной чистой дискете те изменения, которые будут внесены в архивный файл, чтобы в случае чего вернуть все в прежнее состояние. Конечно же, мне нужно было согласиться на это и выбрать вариант **Undo** в диалоговом окне программы, а не **Ok**, который был предложен по умолчанию. Но я, чисто механически нажав Enter, отрезал себе пути к отступлению.

О лицензионной привилегии

Не вникая в детали, сообщу, что после сделанных исправлений файл сжатого тома перестал читаться как старым, так и новым компрессором. Собственно, файл-то — вот он, можно было копировать его на другой логический диск, можно было дать ему другое имя. Одного только с ним никак не получалось — представить его содержимое в виде совокупности отдельных файлов, чтобы можно было с ними работать (как говорят, присоединить сжатый диск). А ведь это все были мои статьи за несколько лет, мои таблицы, рисунки, упражнения; да, кстати, и не только мои, а всей семьи — одним словом, труды. Уж как я только не пробовал добраться до них, пусть бы даже не до всех, но хотя бы до некоторых. Никакие утилиты не помогали, никакой диск-доктор. Было от чего прийти в отчаяние.

Дошло до того, что я уже вручную, на низком уровне, через нортоновский Disk Editor, байт за байтом просматривал информацию, записанную в сжатом файле. И, надо сказать, многое понял. Этот файл имеет структуру как у настоящего диска — загрузочная запись, таблица размещения файлов (FAT — *File Allocation Table*), корневой каталог и т.д. Вот в них-то и содержались неверные сведения. А сами исходные файлы никуда не делись, я их видел, хотя и в уплотненном формате, непригодном к работе. Файлы есть, а доступа к ним нет, потому что структура этого как бы диска нарушена. Все равно как если бы на железной дороге было перепутано расписание: поезда есть, а движение невозможно.

Мне очень хотелось восстановить порядок в структуре. Недели две-три (не подряд, конечно, с перерывами) я провел за экраном, пытаясь нащупать ее закономерности. Всех своих знакомых расспросил, все свои книжки просмотрел. И так пробовал изменить загрузочную информацию, и этак. Пока не понял, что без специальной технической документации гадать тут можно долго и надо бы все-таки обратиться к службам технической поддержки. В московское представительство Microsoft, наконец!

Набирая телефон фирмы, я уже предполагал, что разговаривать со мной станут, только если у меня Windows 95 лицензионный, а не пиратский. И был готов удостоверить это соответствующим документом — сертификатом подлинности с моим личным номером. После улаживания формальной стороны дела меня соединили с инженером, которому я (по телефону же) изложил проблему. Он внимательно слушал, а я во время рассказа все радовался про себя — вот, мол, как хорошо, все по-честному, не зря были потрачены 40 долларов на фирменный дистрибутив.

Однако время идет, я все рассказываю и рассказываю, а он все молчит и молчит. Через несколько минут возникло поначалу смут-

Окончание на с. 14

[1] Я считаю необходимым разбивать один жесткий диск на три логических, чтобы на C держать приложения и саму операционную систему, D предоставлять учащимся для их текстов, рисунков, программ и других упражнений, а E использовать как место, где хранятся дистрибутивы. Не все, конечно, но некоторые, часто требуемые, по крайней мере — сам Windows 95. На каждом рабочем месте диск C имеет размер 255 Мб, диск D — 45 Мб, а все остальное приходится на E (от 50 до 200 Мб).

[2] Казалось бы, зачем он мне, если класс не подключен к глобальной сети? А затем, что и в локальной можно (и нужно!) работать по Интернет-технологии, — организовывая так называемый Интранет.

[3] В предыдущей серии статей 1997 г.

[4] Этот файл имеет имя DBLSPACE.000, и его еще называют файлом сжатого тома, или по-английски CVF — *compressed volume file*.

Сжатый диск в Windows 95

Окончание. См. с. 1, 13

ное, а потом все более определенное ощущение, что собеседник не очень-то разбирается в данном вопросе. Наше общение длилось с полчаса, инженер куда-то пропадал на длительные периоды (листал справочники, что ли) и в конце концов стал клонить к тому, что структура загрузочного сектора у сжатого файла — это довольно сложная вещь.

Да не такая уж сложная, отвечаю. В принципе я сам могу разобраться и прошу только кое-какую техническую документацию. А он — нет, говорит, нельзя, это наше "ноу хау". Я даже рассмеялся. Ну не может быть такого, думаю, просто причину ищет, чтобы отфутболить. Вот алгоритм сжатия информации действительно может являться секретом, а таблица размещения файлов — какое же здесь "ноу хау"?

Я ему свое, он мне свое. Я говорю: соедините меня с вашим начальством, пусть дадут мне другого инженера; он говорит: зачем же так резко, давайте пошлем запрос в Европу — может, там нам помогут. Ну хорошо, я человек уступчивый, запрос так запрос. Через два дня пришел смехотворный ответ: в Европе подумали и пришли к выводу, что для восстановления порядка надо воспользоваться программой Scandisk. Той самой, с которой все и началось. Круг замкнулся. Мне стало ясно, что с этой стороны помощи ждать нечего.

Так все и пропало. Не совсем, конечно, пропало, поскольку файл DBLSPACE.000 все-таки оставлен мною до лучших времен — скопирован на другой логический диск (на D) и лежит там пока что без употребления. Может, все-таки и сам разберусь когда-нибудь. А диск E пришлось форматировать. После такой неудачи уже и разгонять процессор что-то расхотелось, к тому же оказалось, что вся партия модемов попалась бракованная. В общем, началась полоса неудач в моей жизни. Но дело это уже прошлое, и вспомнил я о нем так уж, к слову, — показать, что да, действительно, аварии с уплотнением бывают. Только и самому не надо быть рассеянным.

Процесс уплотнения

Но если на домашнем своем компьютере я могу обойтись и без уплотнения — три гигабайта на винчестере позволяют об этом не думать, то в классе, как уже было сказано, другого пути нет. Сжимать диск все равно рано или поздно, а придется. Так лучше уж не ждать, когда он будет заполнен до отказа и станет страшно неповоротлив, а проделать эту необходимую операцию прямо на старте, как только установил систему. На всех машинах по очереди. Лучшего момента для этого, чем сейчас, не найти.

Чтобы сжать диск, я запускаю соответствующую служебную программу:

Пуск ⇒ Программы ⇒ Стандартные ⇒ Служебные ⇒ Сжатие данных.

На экране, как обычно, появляется серия из нескольких диалоговых окон, и вот первое.

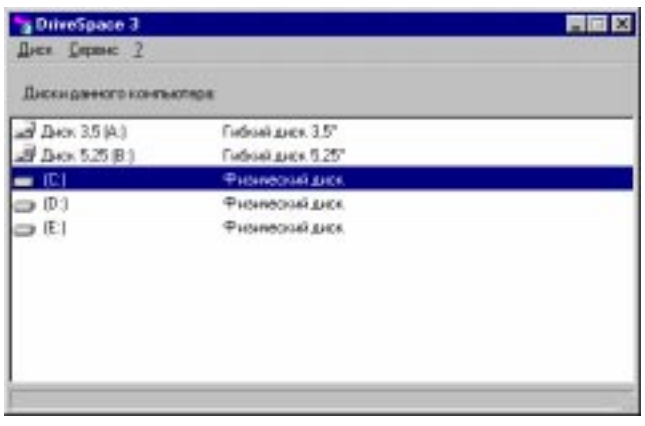


Рис. 1. С каким диском будем работать?

Сначала я сожму диск C, а потом, когда все закончится, еще раз запущу программу для следующего, для D; третий же — E — уплотнять нет смысла, ведь там хранится дистрибутив, и так уже уплотненный до предела. Переведя синюю полоску в нужное положение, выбираю из меню **Диск** команду **Уплотнить**. И получаю предсказание относительно результата начинаемой операции.

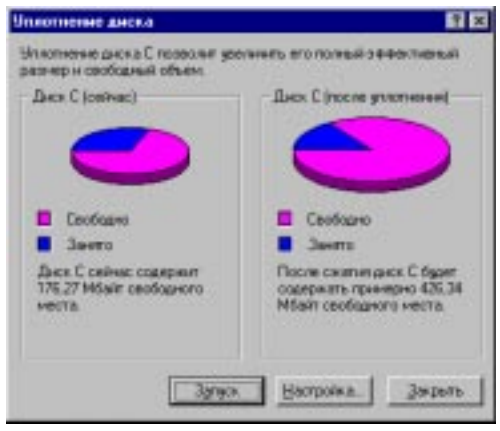


Рис. 2. Что было, что будет...

Вот это да! Свободных четыреста мегабайт вместо ста семидесяти. Очень хорошо. Можно бы и согласиться, но что мне мешает зайти в **Настройку**, хотя бы посмотреть: а какие параметры уплотнения взяты по умолчанию и нельзя ли что-нибудь еще улучшить?

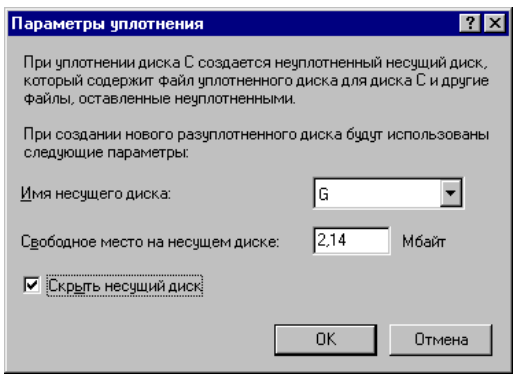


Рис. 3. Зачем-то два мегабайта оставляют несжатыми

Буква для диска сначала была предложена F — первая же свободная после C, D и E, только мне больше нравится G, потому что на одном из компьютеров в классе буква F уже занята под CD-ROM, и, чтобы уж всюду было одинаково, я раскрываю список (черный треугольничек справа) и выбираю подходящую. Далее, зачем же оставлять два с лишним мегабайта неуплотненными? Это излишняя роскошь, и вместо указанного числа не попробовать ли сразу ввести ноль?

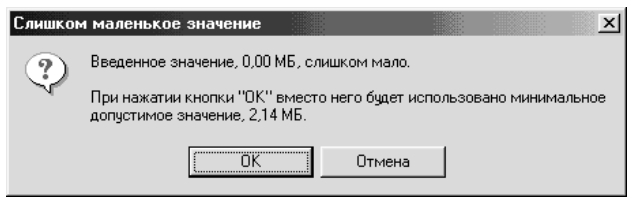


Рис. 4. Сколько-то места надо оставить для технических нужд

Ну, нельзя так нельзя. А вот уж галочку внизу, которая тоже поначалу отсутствовала, поставить, безусловно, следует, чтобы в окне **Моего компьютера** отображался один только фиктивный диск C, с которым, как мне будет представляться, я и работаю, а реального G во избежание путаницы пусть совсем не будет видно.

После всех этих приготовлений, возвратившись в третье окно, я нажимаю наконец **Ok**, и процесс, как говорится, пошел. Впрочем, нет, будет еще одна заминка — мне предложат сделать загрузочную дискету, так, на всякий случай, вдруг что-нибудь непредвиденное произойдет. Чтобы было с чего запустить компьютер. Но у меня такая дискета давным-давно уже заготовлена, и хотя лишняя не помешает, я все-таки отказываюсь, чтобы побыстрее перейти к делу.

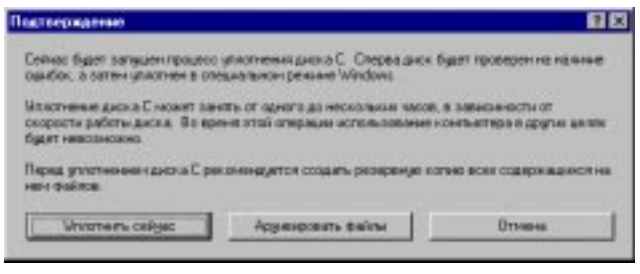


Рис. 5. Еще не поздно передумать

Следует последнее решительное предупреждение: мол, это может занять очень много времени, да, кроме того, еще файлы, бывает, пропадают по дороге и вообще, может, не будем? На самом деле ничего страшного: времени потребуется минут сорок, ну, может, час (в зависимости от того, насколько много уже записано на диске), а файлов каких-нибудь уникальных пока что нет, архивировать для резерва на дискеты ничего не нужно. Давайте уж начнем уплотнять прямо сейчас!

Ну и начинается. Сначала довольно долго идет проверка диска на наличие сбойных участков, только вряд ли они должны быть — ведь совсем недавно, при установке системы, диск форматировался. Потом, когда проверка закончится, компьютер будет предложено перезапустить — и начнутся главные дела: само уплотнение, затем дефрагментация, потом переопределение размера и т.д. Я в этих действиях активного участия не принимаю, только смотрю, как на экране крутятся стилизованные диски, разбрасываются и вновь собираются вместе кусочки файлов. Заканчивается все снова перезагрузкой — и вот она, финальная картина. В окне **Моего компьютера** по-прежнему диски C, D и E (а никакого G не видно). Все выглядит, как было раньше. Только если щелкнуть правой клавишей мыши по диску C и посмотреть его свойства, то объем окажется возросшим, как было обещано, вдвое.

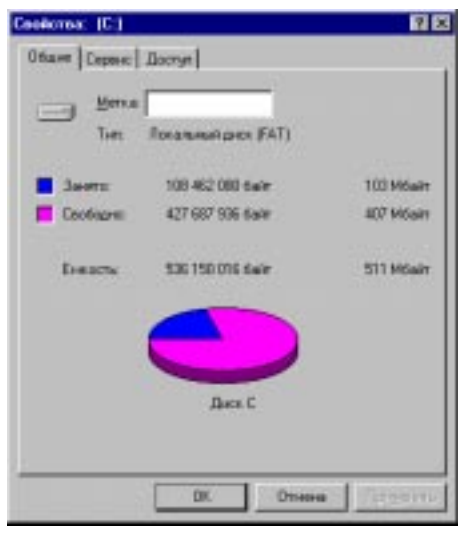


Рис. 6. С таким диском можно жить!

Был диск на 255 Мб, из которых около восьмидесяти было занято системой, а теперь у меня как бы диск на 511 Мб. Правда, занято на нем уже почему-то 103 Мб вместо 80; наверное, какие-нибудь новые файлы, нужные для компрессора, добавились. Но даже если и так, все равно выигрыш огромный. Осталось сделать совсем немного, кое-какие последние штрихи добавить к почти готовой картине.

А именно — надо еще раз запустить программу DriveSpace и уже во втором ее меню, **Сервис**, выбрать пункт **Настройка**.

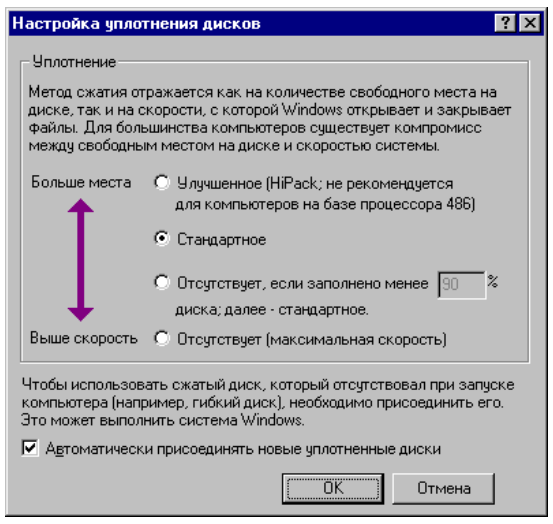


Рис. 7. Это уже лишнее

Тут менять вариант "стандартное" на что-то другое без особой нужды не рекомендуется, а вот галочку внизу лучше убрать. Ее отсутствие позволит компрессору каждый раз, когда я буду ставить ту или иную трех- или пятидюймовую дискету, не беспокоиться по поводу того, сжата она или не сжата. Потому что ну кому, скажите, придет в голову заниматься таким пустым делом, как уплотнять дискеты, — чего же компрессору зря силы тратить?

И это все. Последняя перезагрузка — и конец заботам. Можно приступать к установке на уплотненный уже диск пакета Microsoft Office, ради чего все и затевалось. Но об этом в следующий раз — в последнем выпуске, относящемся к этой серии.

Разноцветные области

С.М. ОКУЛОВ

Дано прямоугольное клеточное поле $M \times N$ клеток. Каждая клетка поля окрашена в один из шести цветов, причем верхняя левая и нижняя правая клетки имеют различные цвета. Две клетки одного цвета, имеющие общую сторону, считаются принадлежащими одной области. Таким образом, поле разбивается на некоторое количество областей.

Правила игры. Вначале первый игрок находится в области, содержащей левую верхнюю клетку, второй — в области, содержащей правую нижнюю клетку. Игроки ходят по очереди. Делая ход, игрок перекрашивает свою область в любой из шести цветов.

В результате к области игрока присоединяются все прилегающие к ней области выбранного цвета, если такие имеются. Если после очередного хода окажется, что области игроков соприкасаются, игра заканчивается. Можно считать, что проигрывает тот, кто не может сделать ход, но суть задачи не в этом.

Требуется написать программу, которая определяет минимально возможное число ходов, за которые игра может завершиться.

Формат входных данных. Цвета пронумерованы от 1 до 6. Первая строка входного файла содержит размеры поля ($1 \le M, N \le 50$). Далее следует описание поля — M строк по N цифр (от 1 до 6) в каждой (без пробелов). Первая цифра соответствует цвету левой верхней клетки игрового поля. Количество областей не превосходит 50.

Формат выходных данных. В выходной файл необходимо вывести искомое количество ходов.

Пример файлов входных и выходных данных.

INPUT.TXT	OUTPUT.TXT
4 3 1 2 2 2 2 1 1 4 3 1 3 2	3

Решение. Сведем задачу к некоторой графовой модели. Будем рассматривать идею решения на примере, приведенном в формулировке задачи. Преобразуем исходную матрицу (`T`) в матрицу областей (`Ob`), определим цвет каждой области (массив `C`) и подсчитаем количество областей (`k`). Для нашего примера массив `Ob` будет выглядеть следующим образом.

```
1 2 2
2 2 3
4 5 6
4 7 8
```

Массив `C` — (`1, 2, 1, 1, 4, 3, 3, 2`), `k` равно 8. Затем по матрице построим матрицу смежности графа (`G`), определяющего связи между областями.

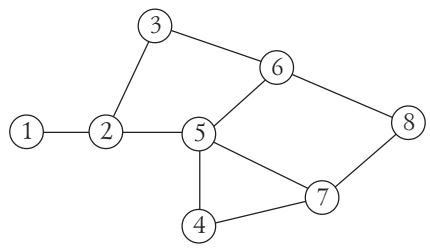


Рис. 1. Граф, определяющий связи областей (для нашего примера)

```
0 1 0 0 0 0 0 0
1 0 1 1 1 0 0 0
0 1 0 0 0 1 0 0
0 1 0 0 1 0 1 0
0 1 0 1 0 1 1 0
0 0 1 0 1 0 0 1
0 0 0 1 1 0 0 1
0 0 0 0 0 1 1 0
```

Рис. 2. Матрица связности графа

После этих замечаний решение задачи сводится к поиску кратчайшего пути, точнее, количества ребер в кратчайшем пути между начальной и конечной вершинами. Для этого можно использовать алгоритм Дейкстры или обход графа в ширину. Схема обхода графа в ширину приведена на рисунке. Рядом с вершинами в квадратных скобках указан уровень просмотра этой вершины. Ответ задачи — значение уровня конечной вершины минус единица.

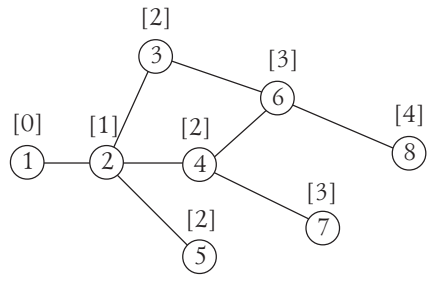


Рис. 3. Последовательность просмотра вершин графа

Приведем фрагменты реализации.

```
Procedure MakeG;
 Var i,j:Integer;
Begin
 FillChar(Ob,SizeOf(Ob),0);
 FillChar(G,SizeOf(G),0);
 k:=0;
 For i:=1 To N Do
 For j:=1 To M Do
 If Ob[i,j]=0 Then Begin
    Inc(k);C[k]:=T[i,j];
 Obl(i,j,k);{процедура заполнения связной
             области значением метки k}
                   End;
 NumObl:=k;{NumObl — глобальная переменная,
           определяющая количество областей}
{формируем матрицу G с помощью рекурсивной
 процедуры Rec}
 For i:=1 To N Do
   For j:=1 To M Do If Ob[i,j]<>0
                    Then Rec(i,j,Ob[i,j]);
End;
```

В процедуре `Obl` реализуется обычная схема поиска выхода из лабиринта.

```
Procedure Obl(x,y,k:Integer);
 Var t:Integer;
 Begin
   If (T[x,y]<>C[k]) Or (Ob[x,y]<>0) Then Exit
   Else Begin
          Ob[x,y]:=k;
          For t:=1 To 4 Do Obl(x+Px[t],y+Py[t],k);
        End;
 End;
```

В данной рекурсивной процедуре используются массивы констант `Px` и `Py`, которые имеют вид:

```
Px:Array[1..4] Of ShortInt=(1,0,-1,0);
Py:Array[1..4] Of ShortInt=(0,-1,0,1);
```

При формировании матрицы `G` используется алгоритм обхода связной области, т.е. области, заполненной одним значением метки в матрице `Ob`.

```
Procedure Rec(x,y,k:Integer);
Var t:Integer;
Begin
 If Ob[x,y]=0 Then Exit;
 If (T[x,y]<>C[k]) Then
  Begin G[k,Ob[x,y]]:=1;G[Ob[x,y],k]:=1;End
 Else Begin
      Ob[x,y]:=0;
       For t:=1 To 4 Do Rec(x+Px[t],y+Py[t]);
      End;
End;

Procedure MakeA;
 Var Og,Nnew:Array[1..MaxOb{максимальное
      количество областей}] of Byte;
 yk1,yk2,i:integer;{указатели (записи,
      чтения) для работы с очередью}
 Begin
  FillChar(Og,SizeOf(S),0);
  FillChar(Nnew,SizeOf(Nnew),0);
  yk1:=1;Og[yk1]:=1;Nnew[1]:=0;yk2:=0;
  While yk2<yk1 Do Begin
    Inc(yk2);
    {выбираем очередной элемент из очереди}
    For i:=1 To NumObl Do
    If (G[Og[yk2],i]=1) And (Nnew[i]=0) Then
    Begin
      Inc(yk1);Og[yk1]:=i;
      Nnew[i]:=Nnew[Og[yk2]]+1;
    End;
                   End;
  {Nnew[NumObl]-1 — ответ}
 End;
```

ЗАДАЧИ

ПРИГЛАШАЕМ НА КОНФЕРЕНЦИИ

Наступает весна. Для учащихся это время "собирать камни" – подводить итоги года. И поэтому все компьютерные конференции для школьников проводят весной – в марте.

Две из них ориентированы на старшеклассников и проводятся в крупнейших московских вузах:

1. **13—14 марта 1999 г.** Московский государственный инженерно-физический институт проводит конференцию-конкурс научных работ учащихся старших классов "Юниор-99", в которой будут следующие секции: математика, физика, информатика, науки о Земле и космосе, науки об окружающей среде. **Тел. 324-91-15.**

2. **16—19 марта 1999 г.** Московский государственный технический университет им. Н.Э. Баумана проводит научную конференцию молодых исследователей "Шаг в будущее. Москва" в части прикладного программирования. **Тел. 263-61-98.**

   Вышеперечисленные мероприятия проводятся в рамках Международного научного и инженерного смотра школьников (International Science and Engineering Fair, ISEF), генеральным спонсором которого является компания Intel.

3. **22—24 марта 1999 г.** Московский дом научно-технического творчества молодежи проводит научно-практическую конференцию учащихся "Поиск-99", где будет работать секция "Новые информационные технологии" по следующим направлениям: прикладное программирование, компьютерные развивающие среды, компьютерные сети и web-технологии, компьютерная графика, анимация и мультимедиа, издание и верстка. На конференции будут работать Интернет-кафе для участников. **Тел. 954-00-58.**

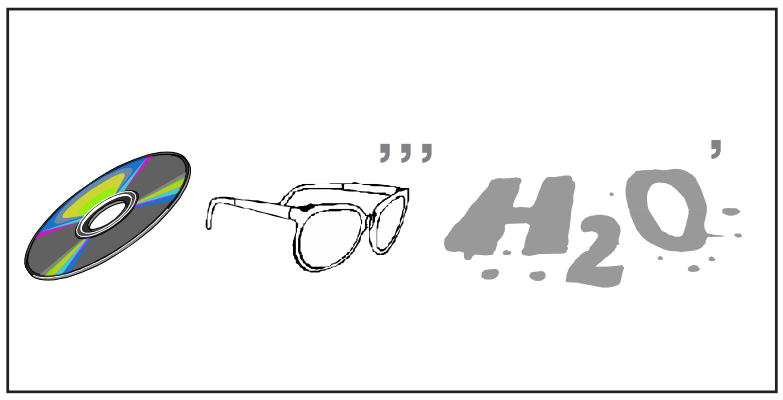
4. **21—26 марта 1999 г.** центр НТТУ "Эврика" г. Обнинска Калужской обл. проводит российскую открытую конференцию учащихся "Юность. Наука. Культура", где будут рассматриваться творческие работы в области компьютерных технологий, в образовании, науке, технике и искусстве, а также программные продукты (системное и прикладное программирование). **Тел. (08439) 4-27-62.**

Все указанные конференции проводятся на долевых условиях и предусматривают оргвзнос.

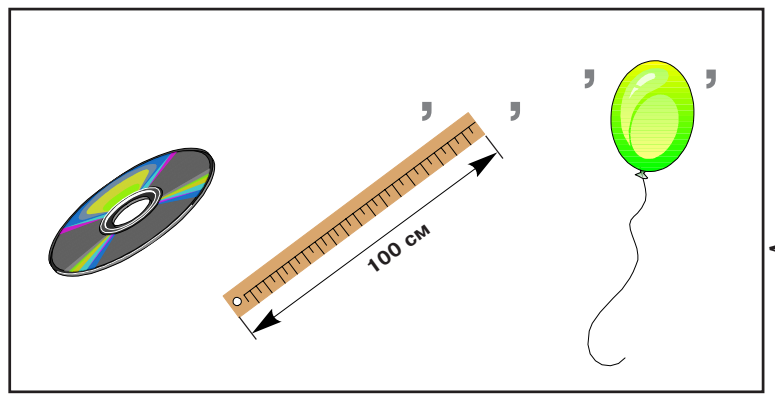
Желаем вам творческих успехов в этих мероприятиях.

Подготовил ***А.Г. Вышеславцев***

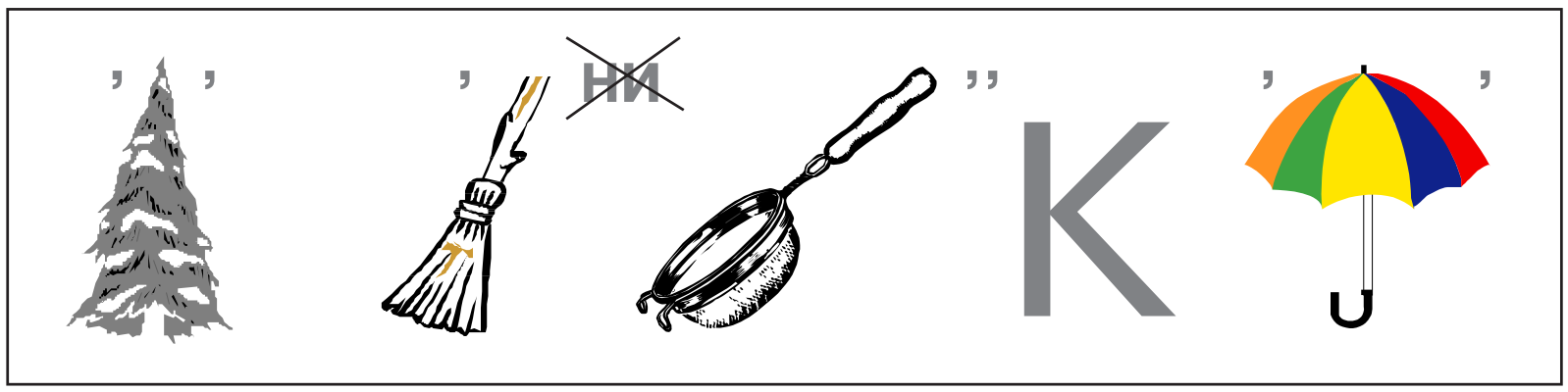
Ребусы по информатике



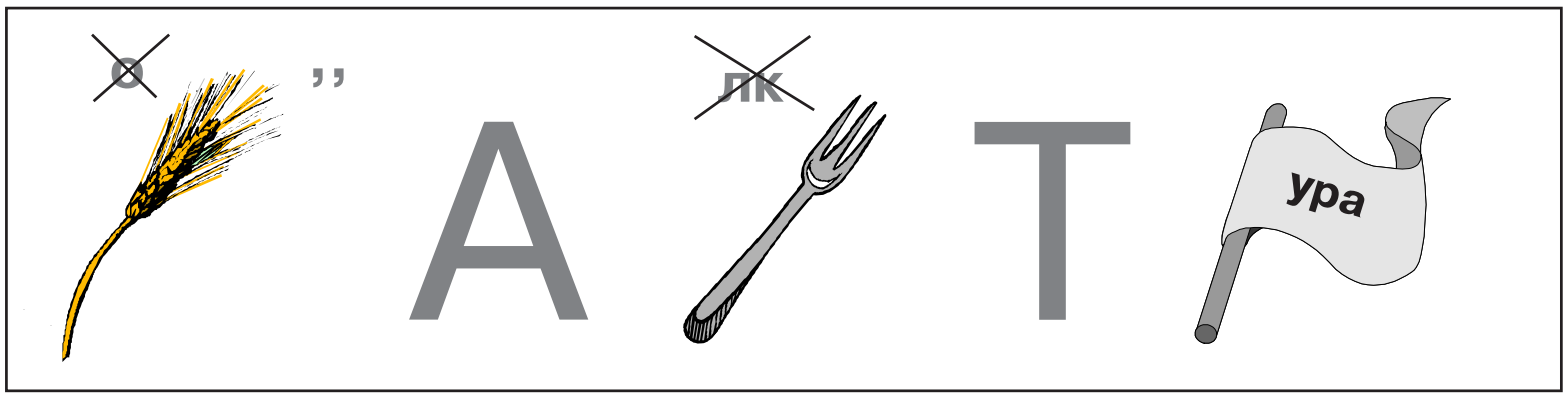
Дисковод



Дискета



Лексикон



Клавиатура

Кто больше?

Эту популярную игру можно использовать на викторинах, конкурсах, конкурсах капитанов и пр. Суть игры проста: выбирается некоторое (достаточно “богатое”) слово и требуется сконструировать из его букв как можно больше слов.

Например:

1. Из слова **КИЛОБАЙТ** можно составить следующие слова:

кол, кит, кот, йота, ил, лоб, лот, лай, байт, бит, бита, бок, бак, бот, бал, бой, блик, тол, тик, блок.

2. РЕДАКТОР:

рак, рок, река, ректор, реактор, род, еда, едок, дека, дар, док, ар, ад, кот, кета, крот, кадр, кедр, код, тор, трек, ток, рот, ода, корт, кора, кед, кредо, кратер, рокер, трак, орда.

3. КЛАВИАТУРА:

кара, кит, кар, лак, лавка, лира, лук, ар, аут, арка, атака, аул, вилка, вал, вар, вата, втулка, ил, ива, Икар, тик, тир, тур, тара, трак, трал, рука, рак, вика, лава, трава, валик, рукав, катар.

4. ПРОГРАММИРОВАНИЕ:

программа, паром, повар, пир, пора, пена, пиво, пени, паром, пар, погром, перо, прорва, право, пирог, помор, пони, рог, ром, ров, рама, рига, рана, репа, рампа, роман, орган, опора, овин, овен, оправа, орава, игра, ива, грамм, гора, горе, гром, грива, гимн, грим, гарем, гомон, горн, говор, грива, аргон, ар, ангар, мир, мор, марево, мрамор, мера, мена, мина, мама, мерин, море, ворон, маргарин, вар, вера, вампир, враг, вагон, ворона, вор, вина, вена, нива, нерпа, нерв, норма, нора, негр, норов.

5. ИНФОРМАТИКА:

инок, Икар, икона, норка, норма, нота, нитка, форма, фора, фирма, факт, финт, фантик, форт, факир, фронт, формат, фара, фата, роман, рамка, романтик, рама, рот, рота, рак, рана, ром, тон, тор, тик, трон, тара, танк, ток, тмин, тариф, торф, том, тир, тина, март, мрак, мир, матка, марка, мор, манка, мак, миф, мина, кора, катар, карман, корм, кара, кант, карат, кино, катион, коран, комар, кофта, корма, кит, картина, кот, ком, кон, кран, крона, ар, аорта, амфора.

Вот еще несколько подходящих слов:

1. Кодирование.
2. Исполнитель.
3. Интерпретатор.
4. Монитор.
5. Отладка.
6. Микропроцессор.
7. Фортран.
8. Электроника.

Материалы рубрики подготовил В.Г. Федоринов

ВНЕКЛАССНАЯ РАБОТА ПО ИНФОРМАТИКЕ

Гл. редактор
Е.Б.Докшицкая
Зам. гл. редактора
С.Л.Островский
Редакция:
Л.Н.Картвелишвили,
Ю.А.Соколинский,
Н.Л.Беленькая,
Н.П.Медведева
Дизайн и компьютерная верстка:
Н.И.Пронская
Корректоры:
Е.Л.Володина,
С.М.Подберезина

Отпечатано с готовых диапозитивов редакции в типографии “ПРЕССА”, 125865, Москва, ул. Правды, 24

Тираж 7000 экз.
Заказ №

121165, Москва, Киевская, 24
тел. 249 4896
Отдел рекламы
тел. 240 1041

ИНДЕКС ПОДПИСКИ
для индивидуальных подписчиков 32291
для предприятий и организаций 32591
комплекта приложений 32744

Internet: infosef@glasnet.ru
Fidonet: 2:5020/69.32
WWW: http://www.1september.ru