В.М. Пестриков

А.Н. Маслобоев

О.К. Федоров

Программирование в системе Turbo Pascal 7.0

Учебное пособие

Санкт-Петербург

2002

Содержание

Введение.

Целью настоящего пособия является привить пользователям персональных компьютеров навыки основ программирования на алгоритмическом языке Паскаль. Созданный в начале 70-х годов признанным классиком программирования Никлаусом Виртом, этот язык был назван в честь французского ученого Блеза Паскаля (1623-1662). Великий ученый, которого современники называли французским Архимедом, вошел в историю не только как автор научных трудов, охватывающих самые различные области человеческого знания – от философии до математики, но и как изобретатель арифметической машины - первого в мире механического счетного устройства.

Изобретение Блеза Паскаля положило начало тому процессу, который привел в конечном счете к появлению современной вычислительной техники, ставшей одним из определяющих факторов научно-технического прогресса. Без компьютеров ныне вообще немыслимо нормальное существование и развитие цивилизованного общества. Исходя из вышесказанного ясно, что название языка было выбрано не случайно. Паскаль был задуман как образцовый язык, который должен определенным образом формировать мышление программистов, помогать им почувствовать законы программирования, его красоту.

Первоначально язык Паскаль разрабатывался прежде всего как язык, предназначенный для эффективного обучению программированию, и успешно справлялся с этой задачей. Например, в США Паскаль был объявлен официальным языком программирования для учащихся средних школ, которые намерены специализироваться в области вычислительной техники и программирования в американских университетах. Но с течением времени Паскаль вышел за чисто учебные рамки и стал равноправным и популярным языком программирования.

В 80-е годы позиции Паскаля еще более упрочились в связи с появлением версий языка, предназначенных для персональных компьютеров. Язык стал использоваться не только как средство обучения студентов и школьников, но и широко стал применяться как рабочий инструмент пользователей. Возникло целое семейство языков Паскаль, и ведущее место в этом семействе занял язык Турбо Паскаль, разработанный программистами американской фирмы Borland. На протяжении ряда лет (1983-1992) фирмой Borland был создан ряд новых, более совершенных версий языка, и в настоящее время Турбо Паскаль представляет собой мощную систему программирования, включающую универсальную интегрированную среду, в которую «погружен» язык. Эта среда значительно упрощает и облегчает процесс создания программ, и в то же время

предоставляет пользователю ряд новых, дополнительных возможностей (использование средств объектно-ориентированного программирования, работа с графикой и звуком и другие). В любой момент времени пользователь может запросить помощь, и на экране компьютера появится информация о режимах работы, командах и операторах языка и т.д.

Хотя система программирования Турбо Паскаль создавалась как приложение операционной системы MS DOS, она успешно работает и на компьютерах, на которых установлена операционная система Windows и продолжает широко использоваться как в учебных целях, так и для решения практических задач.

Авторы данного пособия поставили перед собой задачу по возможности просто и доступно изложить материал, необходимый для самостоятельного изучения Турбо Паскаля и написания программ в этой среде. Для овладения материалом, изложенным в пособии не требуется предварительная подготовка в области программирования, и читателям желательно иметь знания и навыки на уровне начинающего пользователя персонального компьютера.

Запуск системы Турбо Паскаль.

Система программирования Турбо Паскаль представляет собой комплекс, содержащий ряд файлов, но как и в любом программном комплексе, в ней имеется один головной файл, запускающий ее на выполнение. Этот файл называется TURBO.EXE. Процедура запуска системы зависит от того, в какой операционной системе или программной оболочке работает пользователь, поэтому рассмотрим следующие три варианта.

1. Запуск Турбо Паскаля из операционной системы MS DOS.

   Для этого необходимо перейти в каталог, содержащий файл TURBO.EXE, набрать в командной строке TURBO.EXE и затем нажать клавишу **Enter**.

2. Запуск Турбо Паскаля из программы-оболочки Norton Commander.

   Вывести в активную панель Norton Commander каталог, содержащий файл TURBO.EXE, установить на этот файл курсор и затем нажать клавишу **Enter** или дважды щелкнуть имя файла мышью.

3. Запуск Турбо Паскаля из операционной системы Windows.

Как и большинство других операций в ОС Windows, запуск файла на выполнение можно осуществить несколькими способами.

а) Открыть папку, содержащую файл TURBO.EXE, и запустить файл на выполнение двойным щелчком мыши.

б) Если система программирования Турбо Паскаль “прописана” в главном меню, открыть меню нажатием кнопки “Пуск”, найти в меню соответствующий пункт и один раз щелкнуть его мышью.

в) Если на рабочем столе Windows для файла TURBO.EXE создан ярлык, для запуска файла дважды щелкнуть ярлык мышью.

После успешного запуска системы программирования Вы увидите на экране компьютера исходный экран системы . Под управлением ОС Windows система программирования может работать либо в полноэкранном режиме, либо в оконном, занимая только часть экрана компьютера. Для того, чтобы выбрать удобный для пользователя режим работы следует отредактировать ярлык файла TURBO.EXE. Вообще, при работе в Windows такой ярлык рекомендуется создать, так как иначе Турбо Паскаль будет запускаться только в режиме MS DOS, в котором для пользователя недоступны возможности ОС Windows.

Для редактирования ярлыка нужно щелкнуть его правой кнопкой мыши и в открывшемся контекстном меню выбрать пункт «Свойства», щелкнув его мышью. В открывшемся диалоговом окне на вкладке «Программа» установим флажок «Закрывать окно по завершении сеанса работы». Тогда , при выходе из системы программирования ее окно будет закрываться автоматически. Затем щелкнем на той же вкладке экранную кнопку «Дополнительно» и в окне дополнительных настроек программы уберем флажок «Режим MS DOS» , после чего можно будет работать в Турбо Паскале, не выходя из Windows. Наконец , перейдем на вкладку «Экран» и устанавливаем переключатель в положение «Полноэкранный» или «Оконный» в зависимости от того, в каком режиме Вам удобно работать.

Теперь система программирования настроена и можно начинать с ней работать.

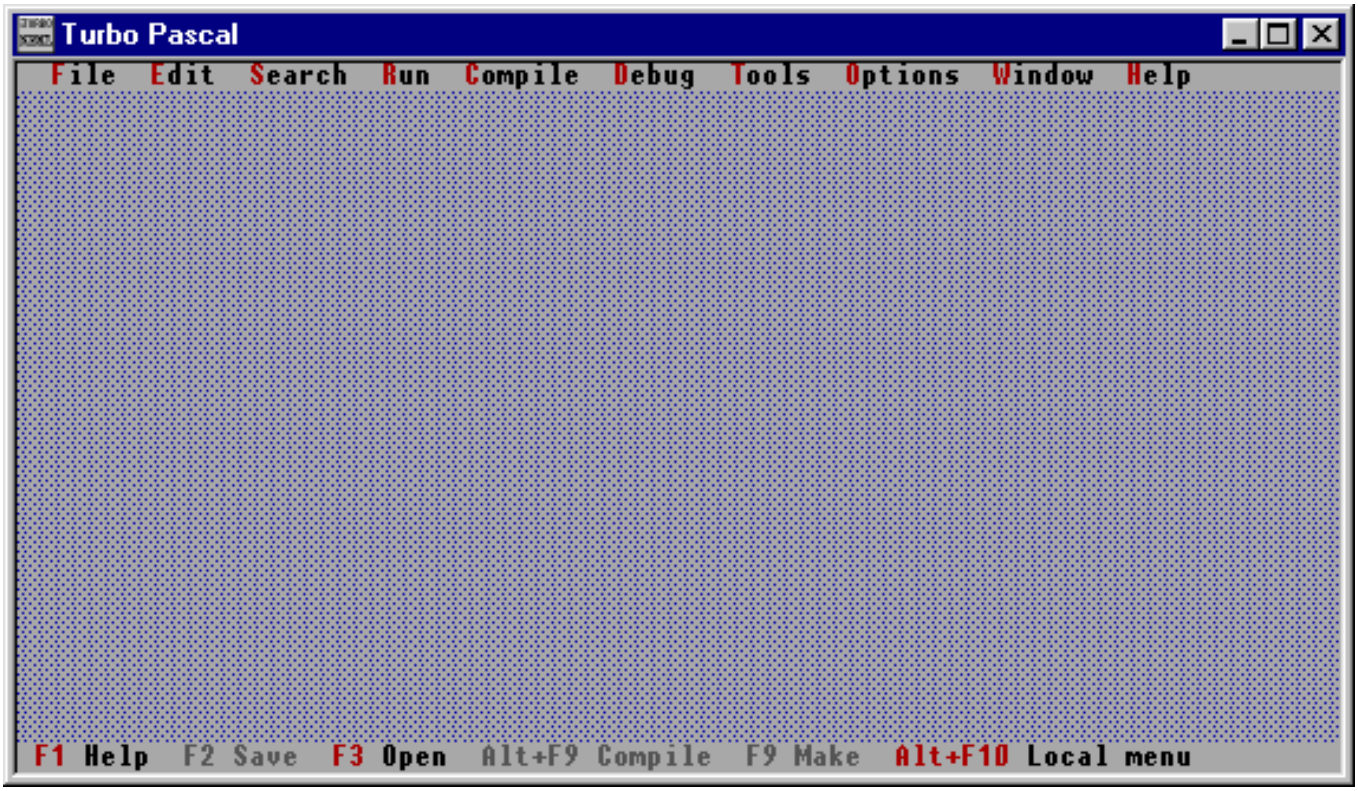


Рис 1. Исходный экран системы программирования

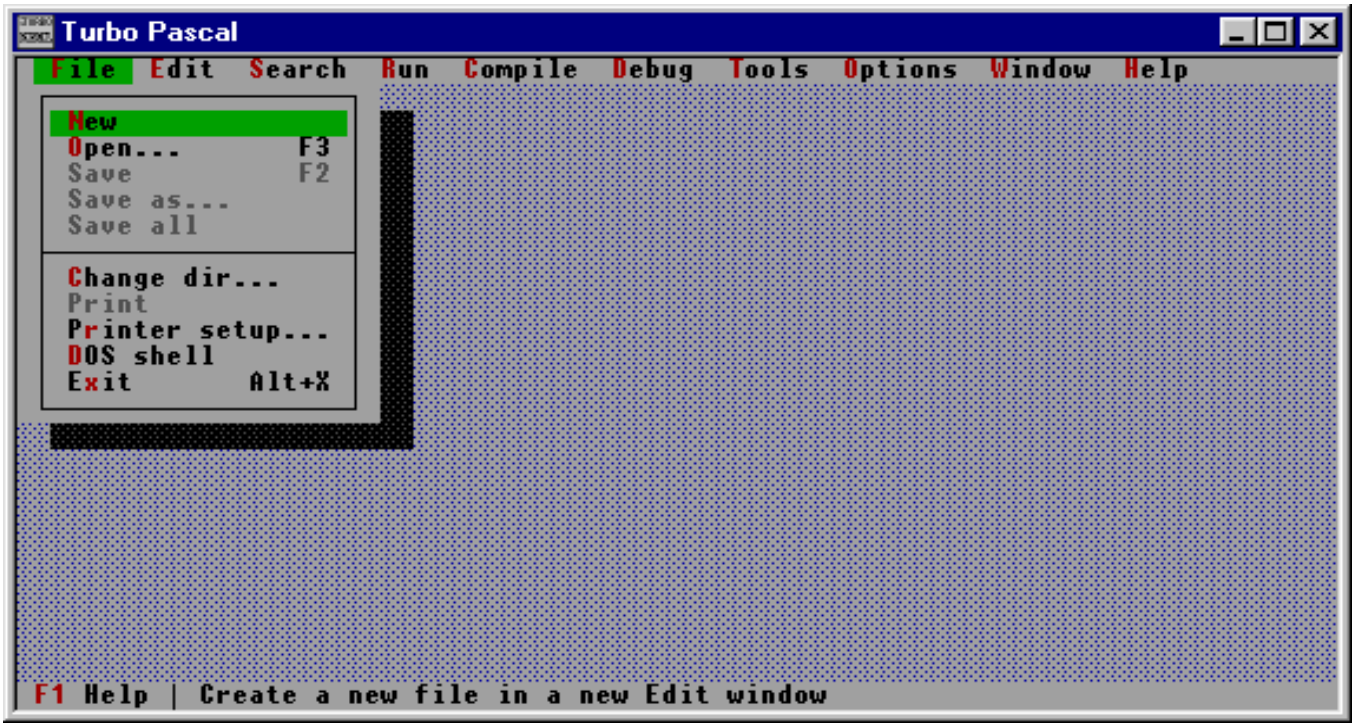


Рис.2 Экран системы программирования с активизированной строкой меню.

Верхняя строка исходного экрана называется строкой меню и содержит десять разделов. Для того, чтобы активизировать строку меню (т.е. привести ее в рабочее состояние) необходимо нажать клавиши **F10** и **Enter** или щелкнуть строку меню мышью. После этого один из разделов меню будет находиться в раскрытом состоянии (будут видны все пункты данного раздела). На рис.2 изображен экран системы Турбо Паскаль после активизации меню (раскрыт раздел **File**).

Данный раздел содержит команды, используемые для операций с файлами: создания новых файлов, открытия ранее созданных, сохранения изменений в файлах, их переименования, смены каталога, в который по умолчанию записываются файлы, а также для выхода из системы программирования.

Для того, чтобы создать программу на языке Паскаль необходимо создать новый файл, в котором будет записан текст данной программы. Создание файла производится выполнением команды **File→New** (для этого нужно щелкнуть мышью пункт **New** в разделе **File** или установить курсор на этот пункт стрелками клавиатуры и нажать клавишу **Enter**. После этого на экране появится окно вновь созданного файла , в котором можно вводить текст программы на языке Паскаль. Этот файл по умолчанию получит имя “Noname00.pas”(см. рис.3). Следующие созданные Вами файлы получат соответственно имена “Noname01.pas”, “Noname02.pas” и т.д.

Так, как такие имена файлов не несут информации о содержании записанных в них программ (словосочетание “Noname” в переводе с английского означает “без имени”) то желательно давать файлам какие–либо осмысленные имена. Например файл, содержащий программу , складывающую два числа, можно назвать “ Summa2.pas”. Имена файлов могут содержать буквы латинского алфавита и цифры. Так как система Турбо Паскаль является приложением операционной системы MS DOS, то необходимо помнить о следующем ограничении: имена файлов с программами на Паскале(как и все прочие имена файлов в MS DOS) не могут содержать более 8 символов (не считая 3 символов, зарезервированных для расширения имени).

Файл, создаваемый в системе программирования Турбо Паскаль должен иметь расширение pas. Причем, если пользователь работает в программе-оболочке Norton Commander, то оболочку можно настроить таким образом , чтобы файл с расширением pas автоматически обрабатывался системой Турбо Паскаль при двойном щелчке мышью на нем или нажатии клавиши **Enter**. Для этого нужно в меню Norton Commander выбрать раздел «Команды», а в нем пункт «Обработка расширений». Для добавления в список расширений нового нажимаем функциональную клавишу **F6** и в диалоговом окне «Изменение

обработки расширений» указываем тип расширения (в данном случае – pas) и путь к файлу TURBO.EXE на Вашем компьютере. Теперь Вам не нужно будет специально запускать систему программирования для работы с программами, написанными на Паскале.

Переименование программы производится с помощью команды **File→Save as …** (Файл →Сохранить как). С помощью этого пункта меню задается также каталог, в котором будет сохранен данный файл. При этом открывается диалоговое окно следующего вида (см. рис. 4). Вообще, если команда меню в Турбо Паскале заканчивается многоточием, то при ее выполнении открывается диалоговое окно.

Это диалоговое окно содержит следующие элементы: текстовое поле, в которое можно вводить новое имя файла; окно со списком файлов, содержащих программы на Паскале, в том каталоге, куда по умолчанию записывается файл; стандартные кнопки **OK**, **Cancel**(Отмена) и **Help**(Помощь), а также информационную строку содержащую сведения о текущем каталоге и о выделенном файле (один файл в каталоге всегда выделен другим цветом). Перемещение между элементами диалогового окна можно производить с помощью клавиши **Tab** или с помощью мыши. Перемещение внутри элементов производится с помощью стрелок управления курсором (← → ↑ ↓) или мышью.

Если каталог, заданный по умолчанию, устраивает пользователя, то ему необходимо после ввода имени сохраняемого файла только щелкнуть 2 раза мышью кнопку **OK** или выделить эту кнопку клавишей **Tab** и нажать **Enter**, после чего диалоговое окно закроется и файл сохранится в текущем каталоге под указанным именем. Если же файл нужно сохранить в другом каталоге, то в этот каталог необходимо перейти. Для того, чтобы перейти в надкаталог (на один уровень вверх) необходимо выделить в каталоге **..**(признак надкаталога) и нажать **Enter** или щелкнуть 2 раза мышью. Для перехода в подкаталог (на один уровень вниз) необходимо выделить в каталоге имя этого подкаталога и нажать **Enter** или щелкнуть 2 раза мышью. Затем в текстовом поле вводим имя сохраняемого файла. Можно вводить имя диска, на котором будет сохранен файл, каталога в котором он будет сохранен и самого файла и непосредственно в текстовом поле, но для начинающего пользователя это представляет определенные сложности, т. к. при этом нужно правильно указать путь к каталогу и файлу.

Если пользователь не будет сохранять данный файл (например, если он был создан только в учебных целях) то нужно щелкнуть кнопку **Cancel**. Если в процессе сохранения файла возникли какие-либо сложности, то пользователям, знающим английский язык, можно воспользоваться кнопкой **Help**. После ее

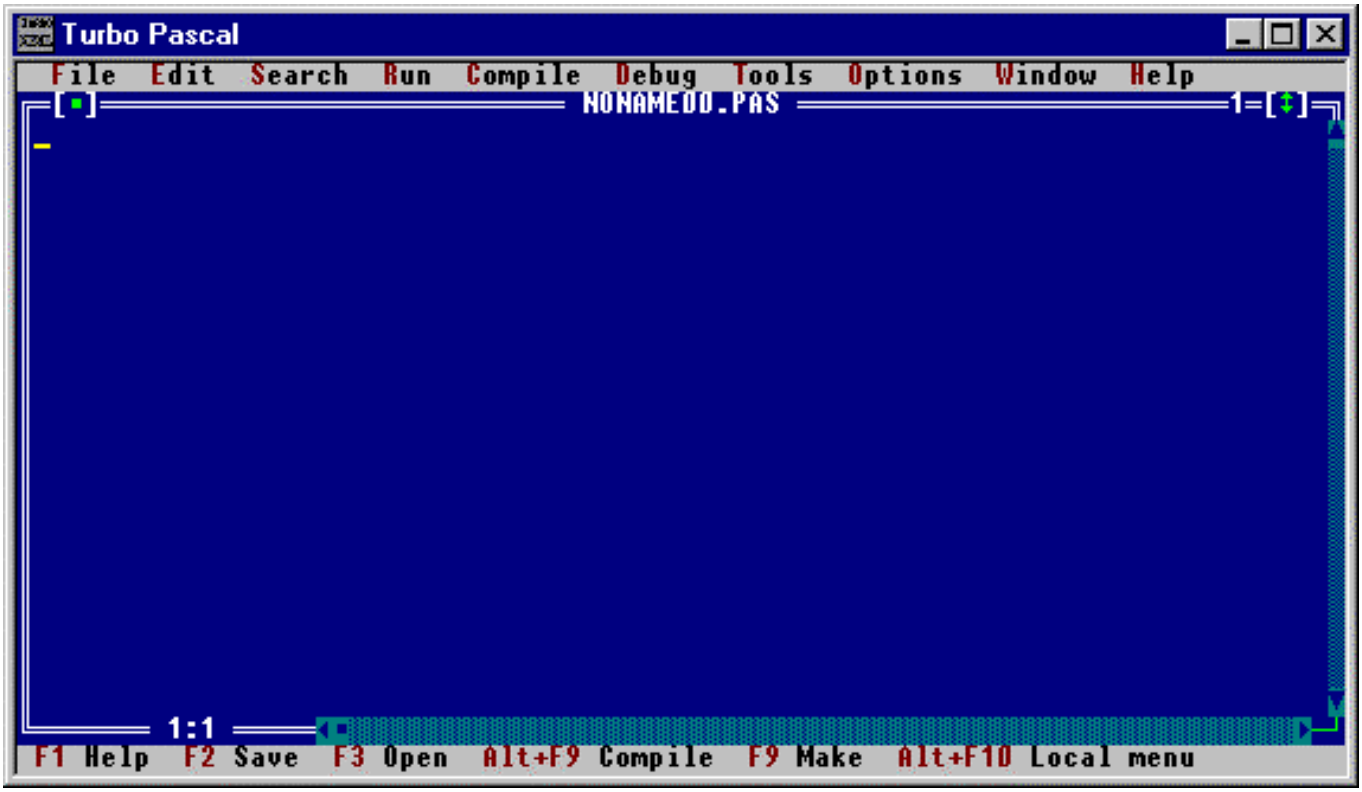


Рис.3 Окно программы в системе Турбо Паскаль

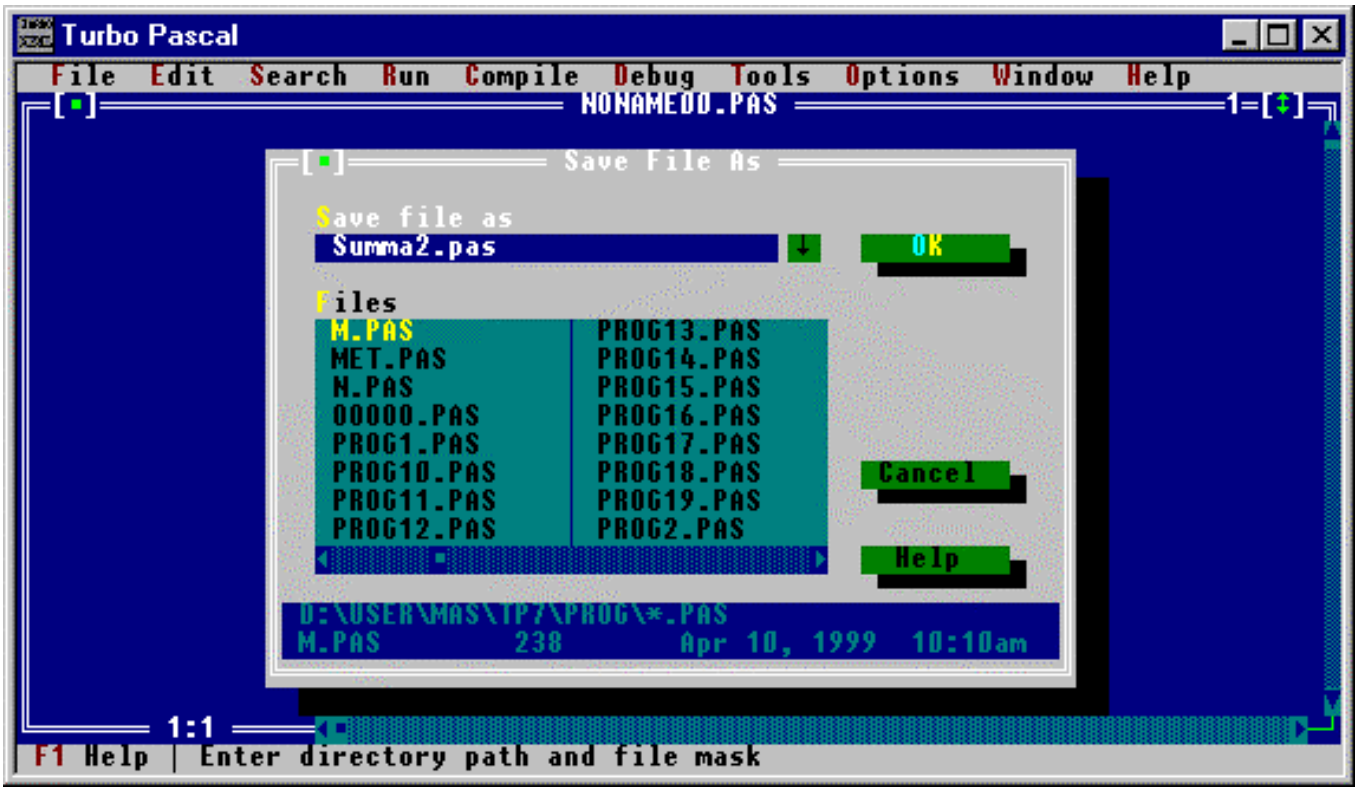


Рис. 4 .Диалоговое окно “Сохранение файла”

нажатия открывается окно с контекстной подсказкой (контекстный – относящийся к данному разделу меню). Такое окно показано на рис.5.

Сохранить файл в нужном каталоге под нужным именем лучше в самом начале работы над программой, т. к. в дальнейшем при необходимости срочно прервать работу над программой можно быстро сохранить ее текст, выполнив команду **File→Save**. При этом все изменения будут внесены в файл автоматически. Теперь, перед тем, как приступить к созданию первой программы на языке Паскаль, пользователю необходимо получить представление об алфавите этого языка и структуре программы на языке Паскаль.

Алфавит языка Паскаль.

Подобно естественным языкам, таким, как русский, английский, французский и другие, которые люди используют при общении друг с другом, языки программирования, на которых человек общается с компьютером, имеют свой алфавит. В данном случае алфавит – это набор букв, цифр и других символов, используемых при написании программ.

Алфавит языка Паскаль включает в себя :

1. Буквы латинского алфавита от **A** до **Z**.

2. Цифры от **0** до **9**.

3. Специальные символы:

а) одиночные

+ - * / = < >
[ ] , ( ) : ;
^ . @ { } $ #

б) парные

<= >= := ..
(* *) (. .)

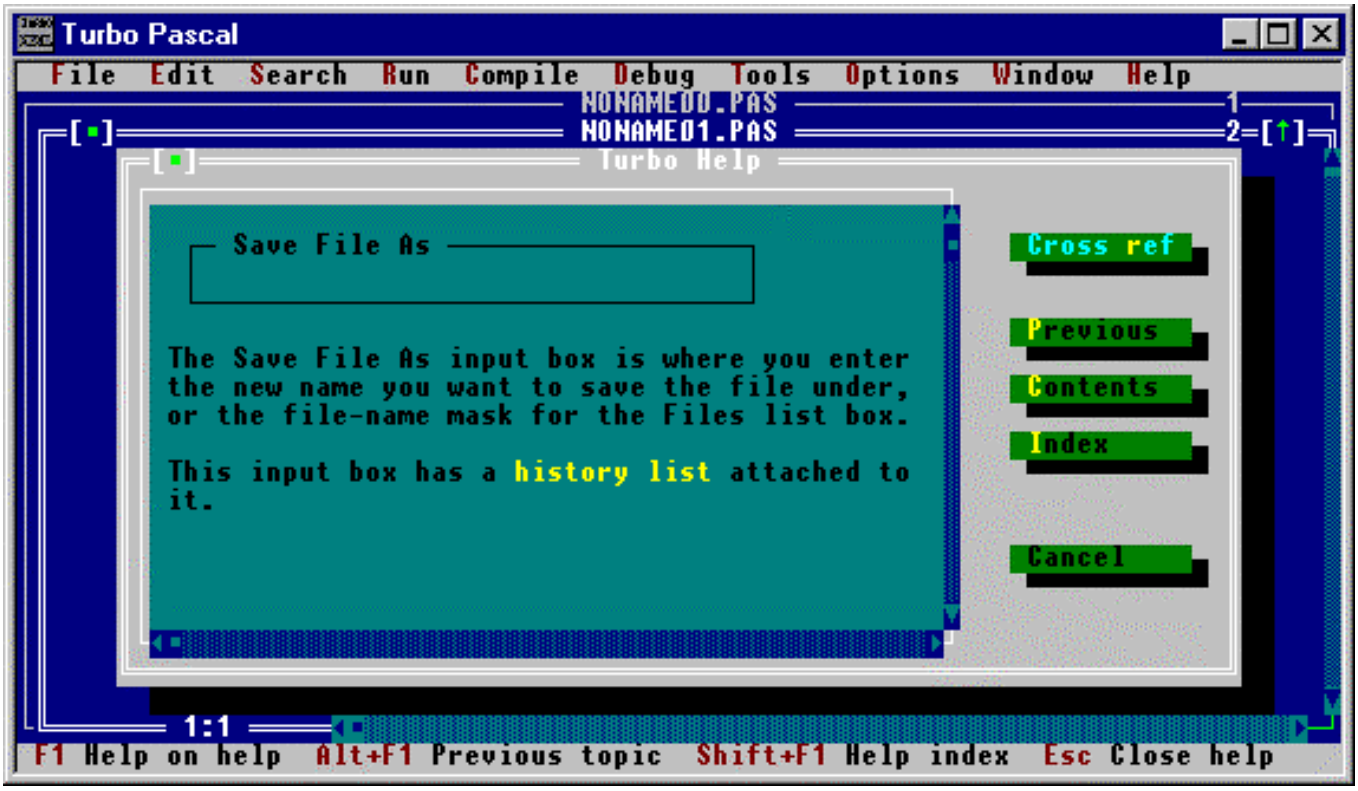


Рис.5 Окно содержания справочной системы и окно контекстной помощи

Наряду с цифрами, буквами и специальными символами Паскаль содержит ряд служебных слов, значения которых заранее определены и не могут изменяться пользователем. К таким зарезервированным словам относятся :

and	**if**	**repeat**
asm	**implementation**	**set**
array	**in**	**shl**
begin	**inherited**	**shr**
case	**inline**	**string**
const	**interface**	**then**
constructor	**label**	**to**
destructor	**library**	**type**
div	**mod**	**unit**
do	**nil**	**until**
downto	**not**	**uses**
else	**object**	**var**
end	**of**	**while**
exports	**or**	**with**
file	**packed**	**xor**
for	**procedure**	
function	**program**	
goto	**record**	

Использование букв русского алфавита допускается только в комментариях (пояснениях к программе, не влияющих на ход ее выполнения) и в строковых константах и переменных.

Структура программы на языке Паскаль.

Программа представляет собой последовательность действий, выполняемых в процессе решения поставленной задачи . Эти действия описываются в виде операторов языка Паскаль, которые и являются основными элементами, из которых складывается программа. При этом операторы работают с различными величинами: постоянными (константами) и переменными. Эти величины перед тем, как они будут использованы в программе, должны быть соответствующим образом описаны в программе. Программа также может иметь заголовок (хотя его наличие и не обязательно).

Поэтому в общем виде программа на языке Паскаль состоит из следующих разделов:

1. Заголовок программы (заголовок может быть произвольным, но для того, чтобы легче было ориентироваться в имеющихся программах, рекомендуется давать программе заголовок, совпадающий с именем файла , в котором она хранится).

2. Раздел объявлений (в нем описываются константы и переменные).

3. Раздел операторов.

Первая программа на Паскале.

Теперь, когда пользователь имеет общее представление о том, как должна выглядеть программа на языке Паскаль, можно приступить к созданию первой программы на этом языке. Эта программа должна будет вывести на экран Вашего компьютера какой-либо осмысленный текст, например, такой: «Моя первая программа». Создадим файл для новой программы командой **File→New** , дадим ему имя "**pervprog.pas**" с помощью команды **File→Save as ...** введем текст программы с клавиатуры компьютера по строкам (рис. 6) Ввод очередной строки завершается нажатием клавиши **Enter**. После нажатия этой клавиши курсор, имеющий форму горизонтальной черты, переходит на следующую строку и можно ее набирать. Разберем подробно текст данной программы. Сделать это будет несложно, так как программа занимает всего четыре строки.

Программа имеет заголовок, который обязательно должен начинаться со служебного слова **program**. После пробела вводится собственный заголовок программы (в данном случае заголовок – **pervprog**). Раздел объявлений в данной программе отсутствует, так как в ней не задействована ни одна переменная или константа. Раздел операторов обязательно начинается со служебного слова **begin**(начало). В самом разделе в данной программе содержится единственный оператор. Это оператор вывода, который состоит из служебного слова **writeln** и списка выводимой информации, заключенного в скобки. Эта информация в данном случае состоит из одного элемента – текста, расположенного между апострофами(одиночными кавычками). Такие тексты в Паскале называются строками. Строка может содержать любые символы (включая буквы русского алфавита) кроме апострофа. При выполнении оператора эта строка выводится на экран компьютера, причем ограничивающие ее апострофы не выводятся, а затем курсор перемещается на следующую строку. Раздел операторов обязательно заканчивается служебным словом **end** (конец), после которого ставится точка – признак конца программы.

Когда ввод текста программы завершен и программа записана в долговременную память компьютера, ее можно запускать на выполнение. Для запуска программы используем пункт **Run** в одноименном разделе меню. После выполнения программы экран компьютера "моргнул" и вернулся в исходное состояние, то есть результатов проделанной работы мы не увидели. Для просмотра результатов следует воспользоваться командой **Output**(Вывод) из раздела меню **Debug**. Тогда на экране компьютера ниже текста программы появится дополнительное окно(прямоугольная область) с результатами работы (Рис. 7). В верхней правой части окна видна цифра 2 – порядковый номер этого окна. Номер 1 имеет окно с текстом программы. Итак, мы вывели требуемый текст на экран компьютера .

Если результаты необходимо просмотреть в полноэкранном режиме, а не в окне, то необходимо выполнить пункт **User screen** (экран пользователя) из того же раздела меню **Debug**. Еще проще развернуть окно с результатами работы во весь экран, щелкнув мышью стрелку в верхней части окна. Для того, чтобы вернуться к экрану с исходным текстом программы следует нажать любую алфавитно-цифровую клавишу.

Окно с результатами работы программы необязательно должно быть “черно-белым”. Текст может выводиться различными цветами , а также может использоваться цветной фон . Для этого необходимо использовать модуль **Crt,** входящий в состав системы программирования. Модулем называется стандартная бибиотека системы Турбо Паскаль. По умолчанию при запуске системы в оперативную память загружается только модуль **System.** Для подключения других модулей необходимо дать специальную команду. В частности, для подключения модуля **Crt** первой командой программы, находящейся сразу после заголовка , должна быть команда **Uses Crt**. Цвет символов задается с помощью команды **TextColor**. После служебного слова **TextColor** в скобках указывается цвет символов. Всего в Турбо Паскале используется 16 стандартных цветов. Вот их названия:

Black – черный

Blue - синий

Green – зеленый

Cyan - бирюзовый

Red - красный

Magenta - фиолетовый

Brown – коричневый

LightGray – светло-серый

DarkGray – темно-серый

LightBlue - голубой

LightGreen – зеленый

LightCyan – светло-бирюзовый

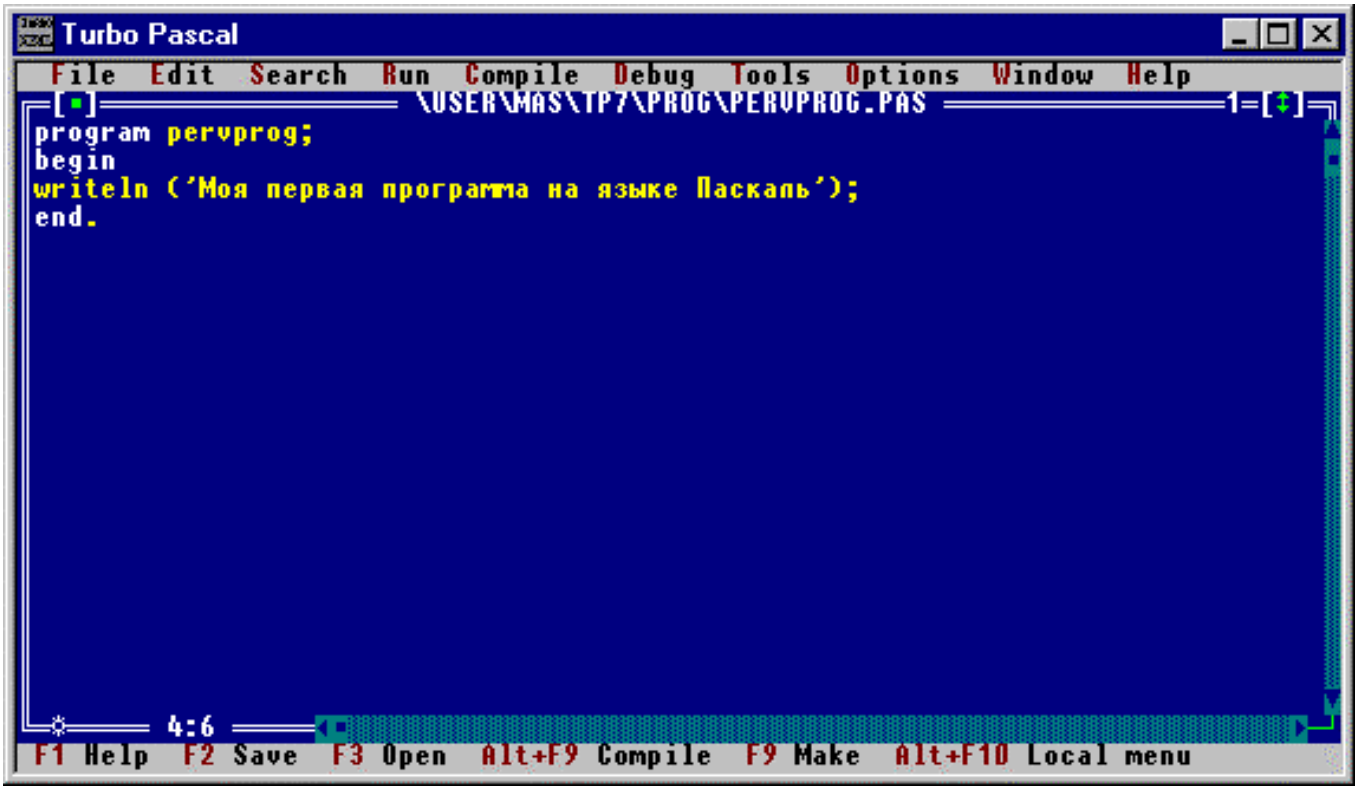


Рис 6. Первая программа на Паскале

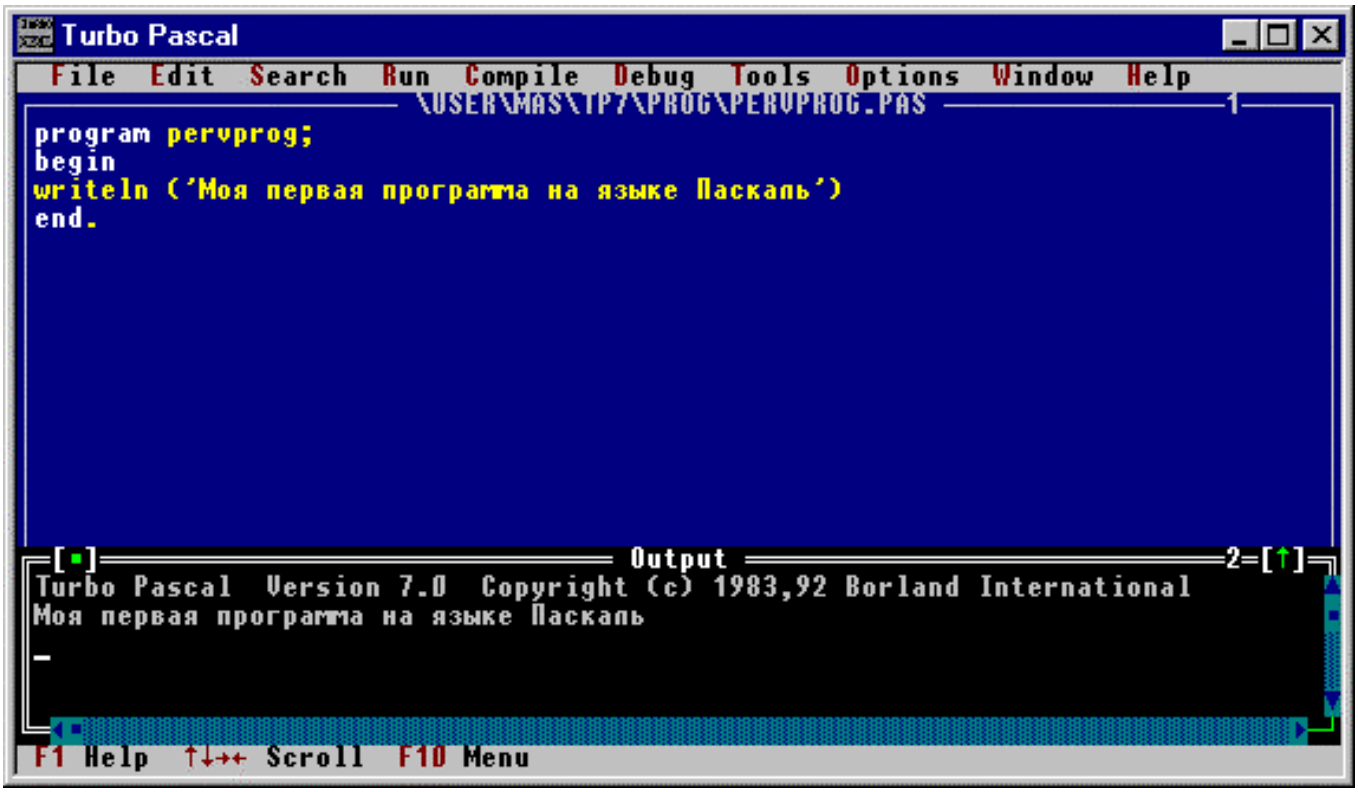


Рис. 7 Результаты работы первой программы

LightRed – светло-красный **LightMagenta** – светло-фиолетовый

Yellow – желтый **White** – белый

Для задания цвета фона используется команда **TextBackground**. Формат ее аналогичен команде **TextColor**, но эта команда позволяет использовать только 8 цветов:

Black **Red**

Blue **Magenta**

Green **Brown**

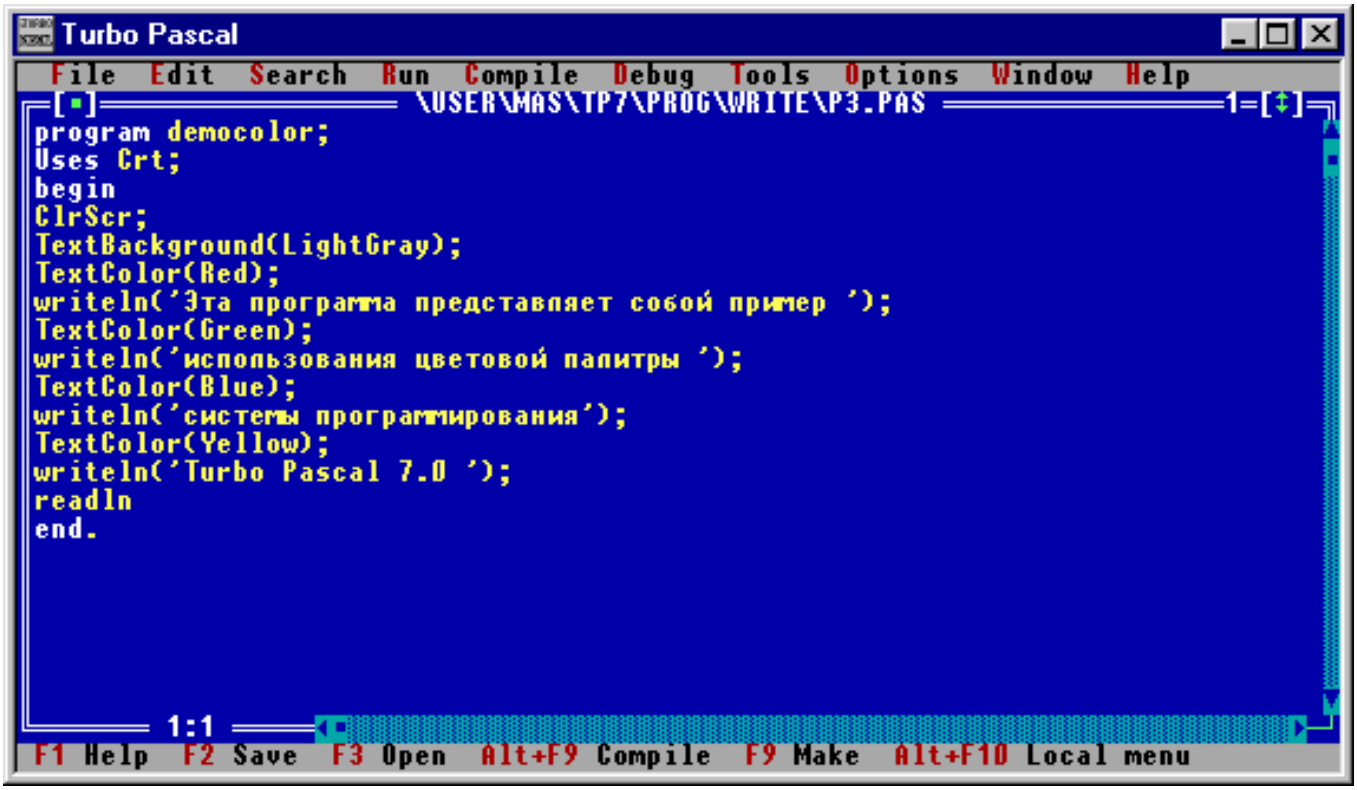
Cyan **LightGray**

Составим программу, которая выводит на светло-сером фоне следующий текст(см. рис.8):

Эта программа представляет собой
пример использования
цветовой палитры
системы программирования Turbo Pascal 7.0

Первая строка этого текста будет выведена красным цветом, вторая – зеленым, третья – синим, а четвертая – желтым. Обратите внимание, что операторы в программе отделяются друг от друга точкой с запятой. В начале программы для очистки экрана дается команда **ClrScr**. Эта команда также работает только в том случае, если в программе подключен модуль **Crt.** Далее, командой **TextBackground** задается фон текста. Перед выводом очередной строки текста командой **TextColor** предварительно указывается ее цвет, а затем сам этот текст выводится оператором **writeln**.

Помимо вышеуказанных возможностей модуль **Crt** позволяет также устанавливать курсор в указанное программистом место на экране, а также создавать различные аудиоэффекты.



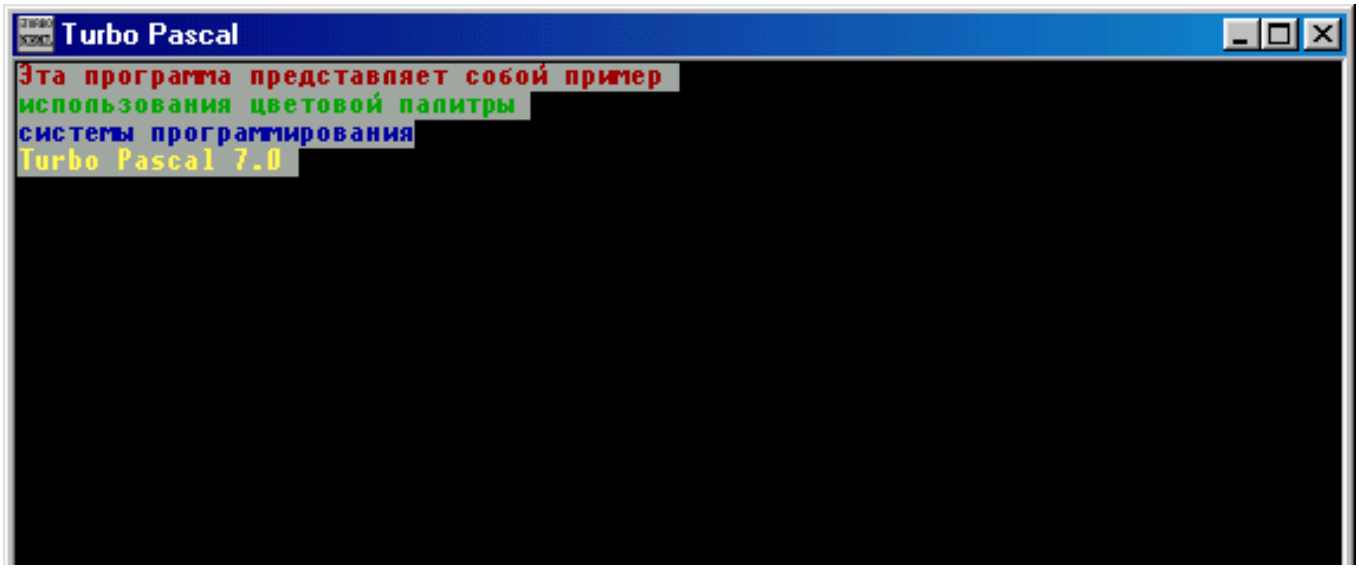


Рис. 8 Программа, демонстрирующая возможности модуля Crt и результат ее работы.

Программы линейной структуры.

Программы, рассмотренные в предыдущем разделе осуществляли вывод информации на экран компьютера. Теперь мы перейдем к задачам, в которых осуществляется не только ввод или вывод информации, но и ее ***обработка*.** Простейшим примером такой обработки информации являются арифметические вычисления, в ходе которых по имеющимся исходным данным подсчитывается результат. Составим программу, которая складывает два числа и выводит их сумму на экран компьютера. Для того, чтобы сложить эти числа , необходимо сперва ***присвоить*** их каким-либо переменным (например, **a** и **b**), а эти переменные предварительно описать. Затем следует присвоить результат арифметической операции сложения третьей переменной (**c)** и вывести результат на экран компьютера.

Итак, программа (см. рис. 9) будет выглядеть следующим образом : после заголовка программы – **sum2** идет раздел описания переменных, который обязательно должен начинаться со служебного слова **var** (сокращение от английского variable – переменная). После служебного слова через запятую перечисляются сами переменные, используемые в программе и после двоеточия указывается тип этих переменных. В данном случае для упрощения задачи будем складывать только целые числа, поэтому все переменные будут относиться к одному типу – integer (то есть – целые). В разделе описания должны быть описаны все три переменные, используемые в данной программе – **a,b** и **c.** Затем, в основной части программы переменным **a** и **b** должны быть присвоены значения с помощью оператора присваивания.

Общий вид оператора присваивания в языке Паскаль

x:=y

где **x –** имя переменной, которой присваивается значение;

y – присваиваемая величина, которая может быть числом или арифметическим выражением . Значение арифметического выражения в процессе выполнения программы вычисляется и присваивается стоящей в левой части оператора переменной.

В операторе присваивания после имени переменной обязательно должно стоять двоеточие. Если пропустить в записи оператора двоеточие, то получится не оператор присваивания, а операция сравнения. (Операции сравнения мы будем изучать позже в разделе «Условные операторы»). Отличие этих двух записей можно наглядно проиллюстрировать на следующем примере:

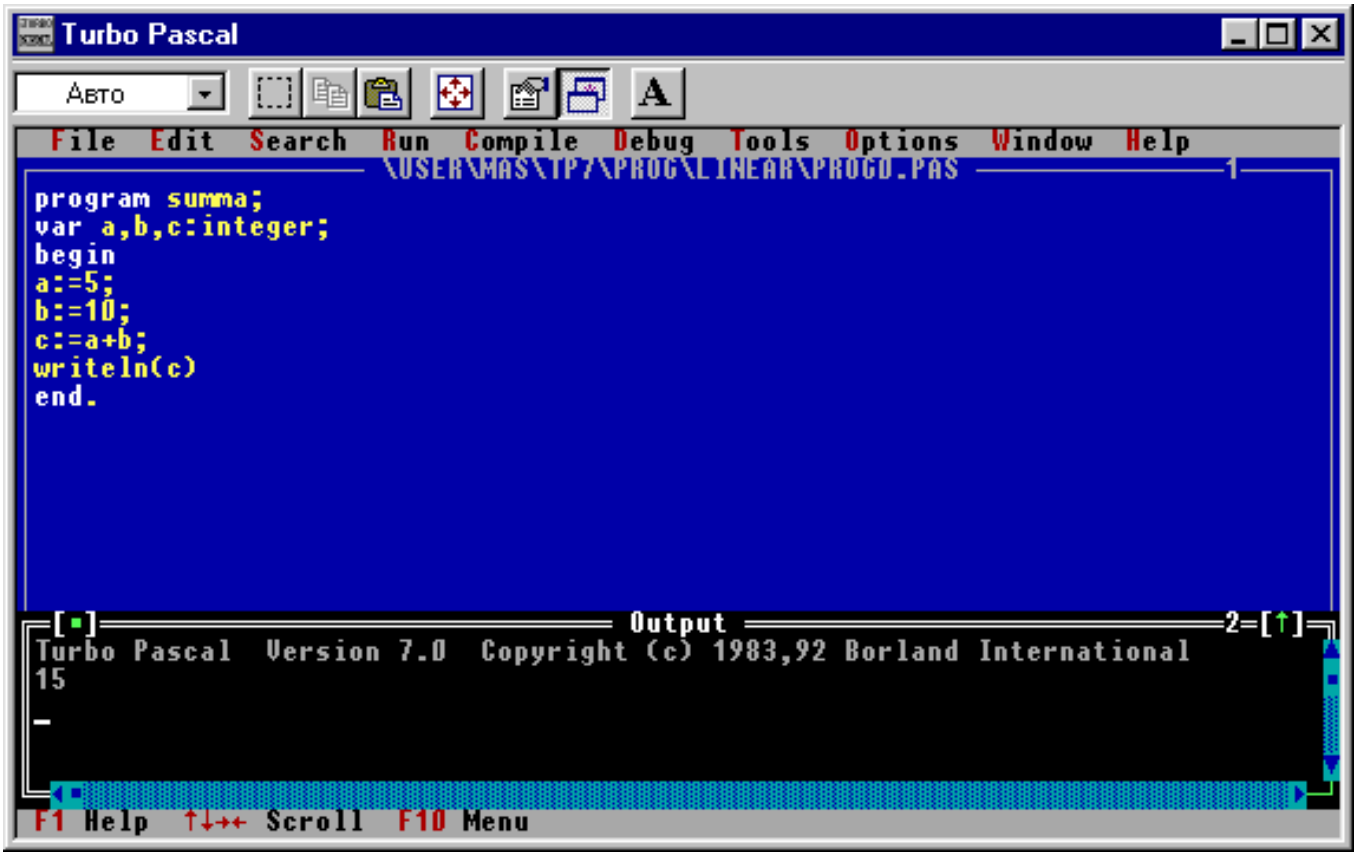


Рис 9. Программа, складывающая два числа и результат ее работы.

запись **x=x+5** является неверной;

запись **x:=x+5** в Паскале является вполне допустимой. Эта запись означает, что переменной **x** присваивается новое значение, которое больше чем старое значение на **5**. Например, если до выполнения операции присваивания значение **x** было равно **3**, то после ее выполнения значение **x** станет равным **8**.

В нашей программе мы присвоим переменной **c** значение выражения **a+b**, то есть в переменной **c** будет содержаться теперь значение суммы двух чисел. Теперь осталось вывести это значение на экран компьютера, что мы сделаем с помощью уже знакомого нам оператора writeln. Следует, однако, учесть, что когда с помощью данного оператора выводится значение переменной, то после служебного слова writeln указывается ее имя, и оно, в отличие от текста, не заключается в кавычки. Поэтому оператор вывода в программе будет записан следующим образом:

writeln(c);

В результате выполнения программы на экран будет выведен результат – число 15.

Эта программа (как и предыдущие) состоит из операторов, которые выполняются последовательно друг за другом. Такие программы называются программами ***линейной структуры***.

Теперь попробуем усовершенствовать программу таким образом, чтобы она складывала любые два произвольных числа, которые пользователь введет с клавиатуры компьютера. Для этого, наряду с уже знакомыми нам операторами, нужно будет использовать также оператор ввода **readln**. Общий вид оператора **readln** похож на оператор **writeln**: он состоит из служебного слова **readln** и списка ввода, заключенного в скобки. В данной программе будут использованы два оператора **readln**, каждый из которых в списке ввода содержит по одной переменной:

readln(a);
readln(b);

Переменным a и b и будут присвоены численные значения, введенные с клавиатуры. Теперь программу можно запускать на выполнение . После запуска программы появится пользовательский экран . Далее, можно вводить числовые значения. Для ввода какого-либо числа необходимо набрать его на

клавиатуре, а затем нажать клавишу **Enter**. После ввода второго числа на экране появится результат работы программы – сумма двух чисел.

Такая программа будет вполне работоспособна, но неудобна для пользователя, так как после ее запуска непосвященному человеку при виде «черного» экрана непонятно, что же нужно дальше делать. Поэтому дополним программу (см. рис. 10). Перед операторами ввода поставим операторы вывода, которые будут выводить информацию, подсказывающие пользователю дальнейший ход его действий. Например, таким образом:

writeln ('введите 2 числа');
writeln ('после ввода каждого числа нажимайте клавишу Enter');

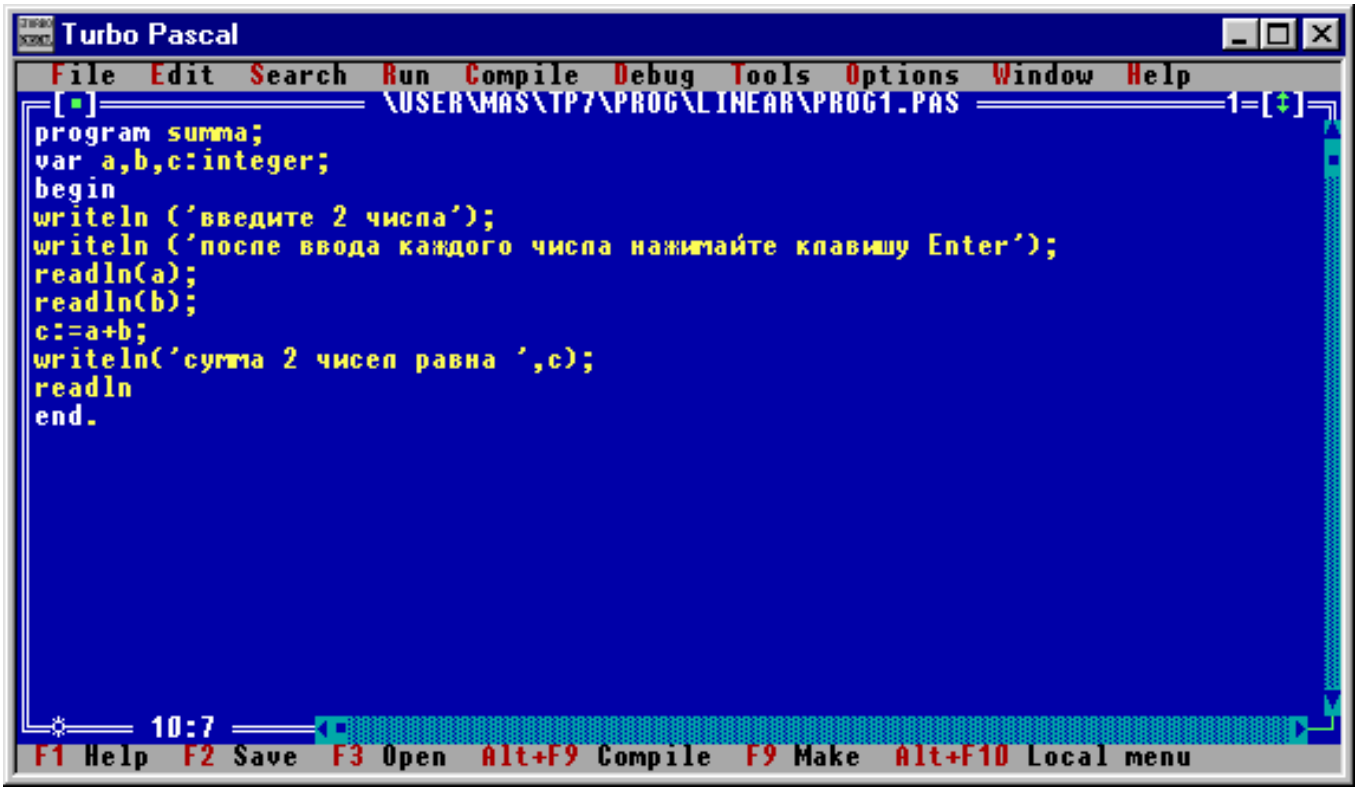
Теперь программа не только правильно работает, но и обеспечивает для пользователя ***дружественный интерфейс***, то есть удобный способ общения человека с программой.

В конец программы внесены еще 2 дополнения. В список вывода оператора **writeln** вставлен еще один элемент – текст, поясняющий полученный результат. Кроме этого, после оператора вывода добавлен еще один «пустой» оператор ввода **readln.** Этот оператор приостанавливает окончание работы программы до тех пор, пока не будет нажата клавиша **Enter**, то есть у пользователя появляется возможность сразу ознакомиться с результатами выполнения программы, которые теперь не исчезают с экрана. (Не нужно будет потом специально открывать раздел меню **Debug**)

В вышеописанной программе использовалась только одна арифметическая операция – сложение. Но в языке Паскаль, естественно , используются и другие арифметические операции. Вот их перечень

+	сложение
-	вычитание
*	умножение
/	деление
div	целочисленное деление
mod	нахождение остатка от целочисленного деления

Первые 4 операции выполняются так же, как в обычной арифметике. Единственное, на что следует обратить внимание: знаки умножения и деления отличаются по написанию (вместо принятых в математике знаков × или · используется * , вместо **:** используется /). Кроме того знак умножения между сомножителямя нельзя опускать, как это часто делается в математике. Что же



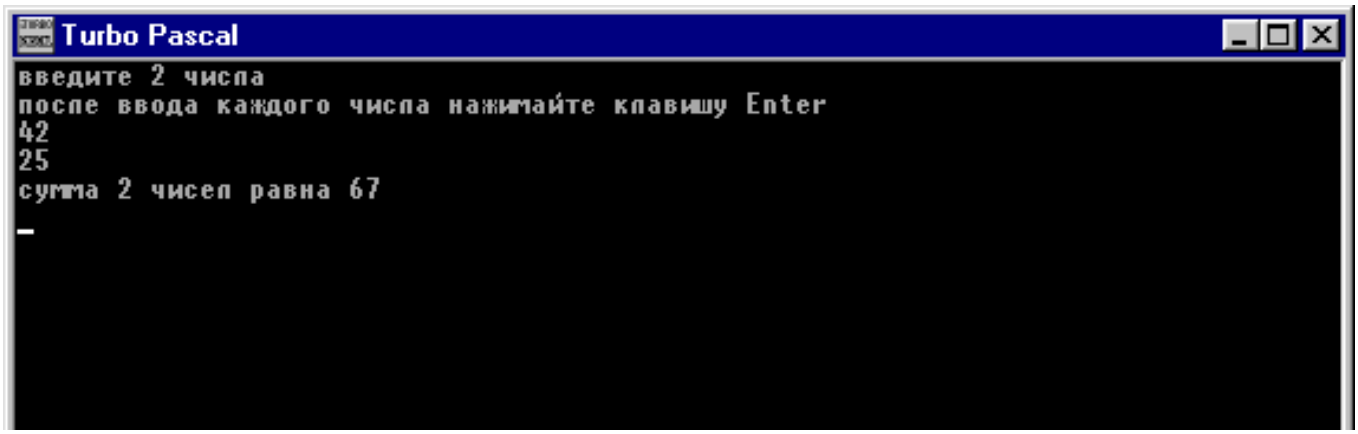


Рис 10. Усовершенствованная программа сложения двух чисел и результат работы программы сложения.

касается целочисленного деления, то оно выполняется как обычное, но затем отбрасывается дробная часть получившегося при делении числа. Операции **div** и **mod** выполняются только над переменными и константами целого типа.

Пример использования действий **div** и **mod** :

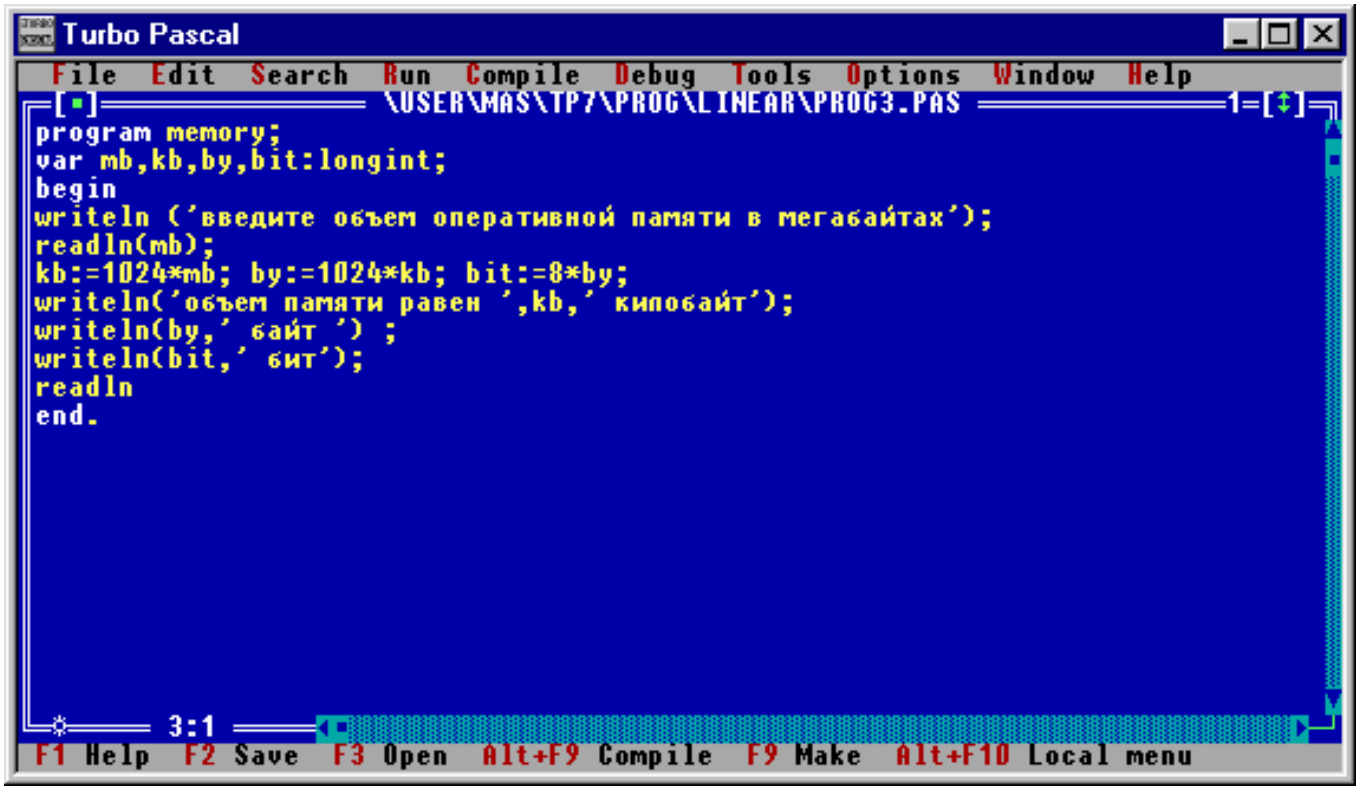
значение выражения 91 **div** 8 равно 11
значение выражения 91 **mod** 8 равно 3

При вычислении значения арифметического выражения сперва выполняются действия, обладающие более высоким ***приоритетом***. К ним относятся умножение,деление, целочисленное деление и нахождение остатка. Сложение и вычитание имеют более низкий приоритет. Если в выражении имеются действия с одинаковым приоритетом, то они выполняются слева направо по ходу выражения. Если необходимо изменить порядок действий в выражении, то следует использовать круглые скобки. Действия, заключенные в скобки обладают наиболее высоким приоритетом, то есть выполняются в первую очередь.

Составим программу, которая пересчитывает объем оперативной памяти компьютера, выраженный в мегабайтах, в килобайты, байты и биты (см. рис. 11). Между этими единицами измерения информации существуют следующие соотношения :

1 мегабайт = 1024 килобайта
1 килобайт = 1024 байта
1 байт = 8 бит

В программе будут использованы 4 переменные : **mb** – для объема в мегабайтах, **kb** – для килобайт, **by** – для байт, **bit** – для бит. При составлении данной программы можно столкнуться со следующей проблемой: результаты работы программы будут выражены достаточно большими числами, которые нельзя обработать с помощью переменных типа integer. Этот тип используется для представления чисел в диапазоне от –32768 до 32767. Поэтому для описания переменных используем другой целочисленный тип данных – **longint**, с помощью которого можно работать с числами в диапазоне от –2147483648 до 2147483647. Тогда программа будет выглядеть следующим образом:



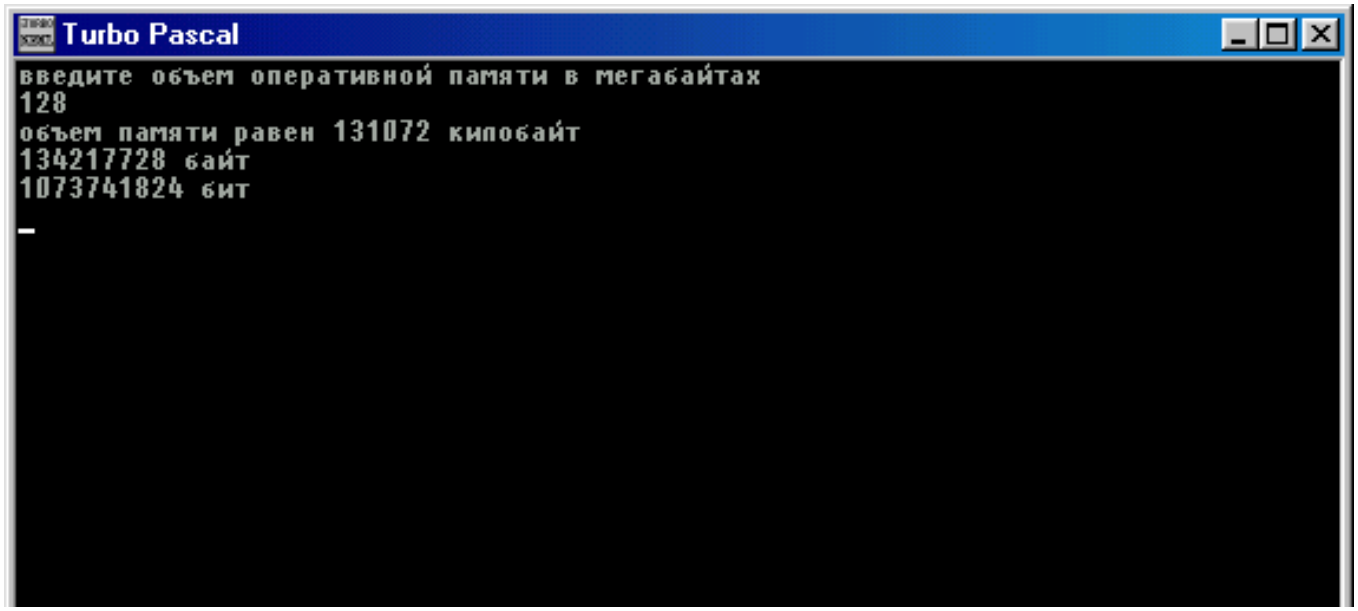


Рис. 11. Программа, подсчитывающая объем оперативной памяти компьютера и результат ее работы.

Для решения ряда задач, в которых используются только целые числа, удобно бывает использовать действия div и mod. В качестве примера составим программу для работы банкомата (см. рис. 12) Банкомат должен выдавать клиенту требуемую сумму, но при этом расходовать наименьшее количество купюр. Пусть в банкомате имеются купюры по 100, 50 и 10 рублей. Тогда ***алгоритм*** решения данной задачи (то есть последовательность действий,необходимая для ее решения) сводится к следующему. В начале программа предлагает клиенту ввести требуемую сумму. (Для удобства работы банкомата попросим ввести сумму кратную 10). Затем эта сумма делится на 100. Результат целочисленного деления и будет минимальным количеством банкнот по 100 рублей. Остаток от деления разделим на 50 и получим соответственно минимальное требуемое количество банкнот по 50 рублей. Наконец, получившийся остаток при делении на 50 разделим на 10 и получим количество банкнот по 10 рублей.

При составлении программы нам понадобятся следующие переменные: **sum** – вводимая клиентом сумма, **k100** – количество купюр по 100 рублей, **k50** – количество купюр по 50 рублей, **k10** – количество купюр по 10 рублей, **vspom** – вспомогательная переменная, в которой содержится остаток, получающийся при делении. Программа будет выглядеть следующим образом:

Редактирование текста встроенным редактором системы ТурбоПаскаль.

Обратите внимание, что в тексте программы «банкомат» три строчки, содержащие оператор **writeln** почти идентичны друг другу. Поэтому для ускорения и облегчения ввода текста можно воспользоваться возможностями встроенного редактора системы. После того как мы набрали первую из трех строчек, два раза скопируем ее, после чего останется внести в скопированные строчки минимальные изменения. Для этого нужно будет сперва выделить первую строчку. Это можно сделать как с помощью клавиатуры, так и с помощью мыши. Если мы используем клавиатуру, то нажатием клавиши «Home» устанавливаем курсор в начало выделяемой строки. Затем выделяем строку: удерживая нажатой клавишу **Shift**, нажимаем нужное количество раз клавишу «стрелка вправо», после чего вся строка будет выделена серым цветом. Еще быстрее можно выделить строку с помощью мыши. Это делается методом протягивания (то есть указатель мыши перемещается по выделяемой строке при нажатой левой клавише мыши). Когда строка выделена, открываем раздел меню **Edit** (редактированиие текста) . В разделе выбираем пункт **Copy**

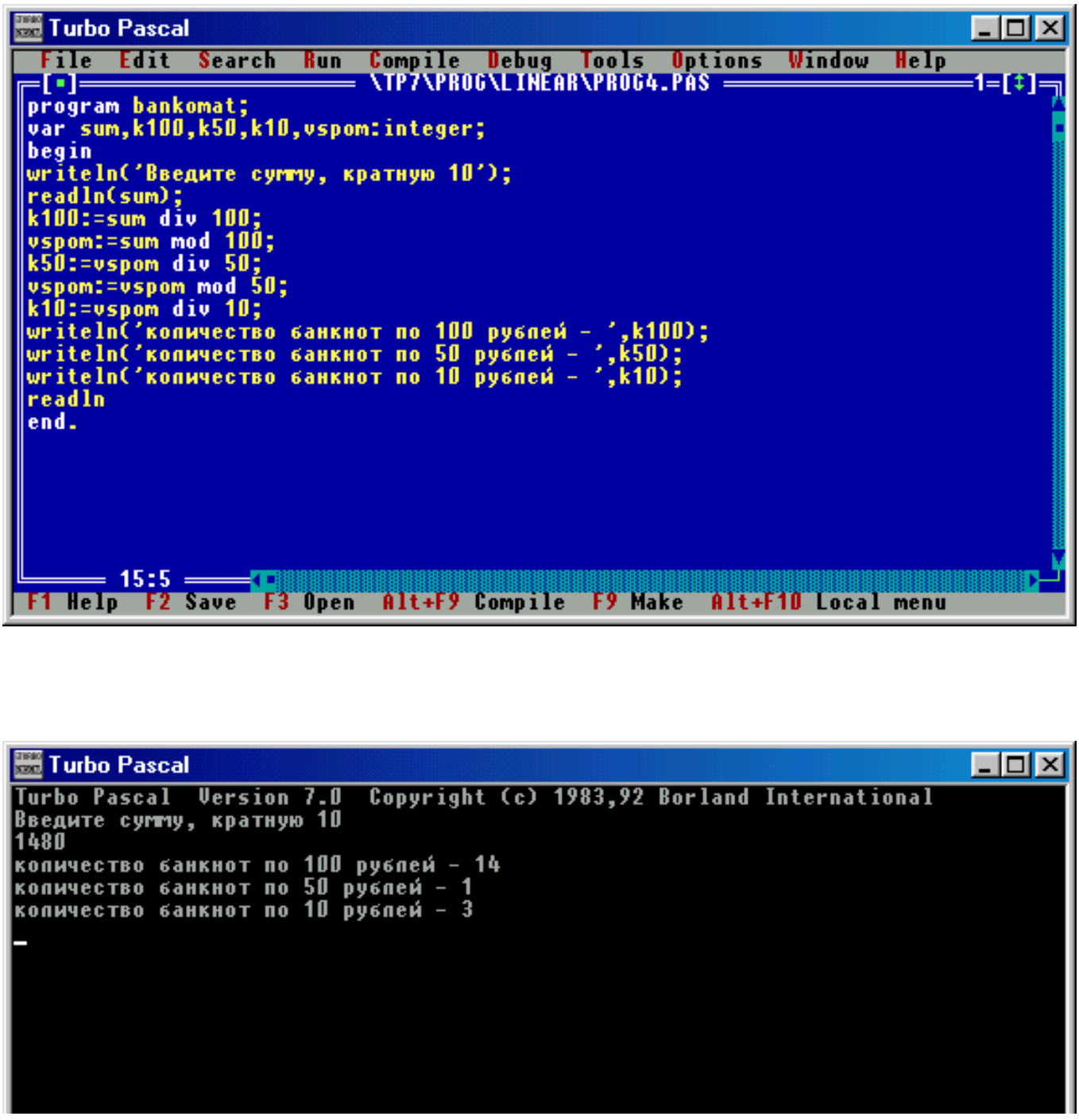


Рис. 12. Программа «банкомат» и результаты ее работы.

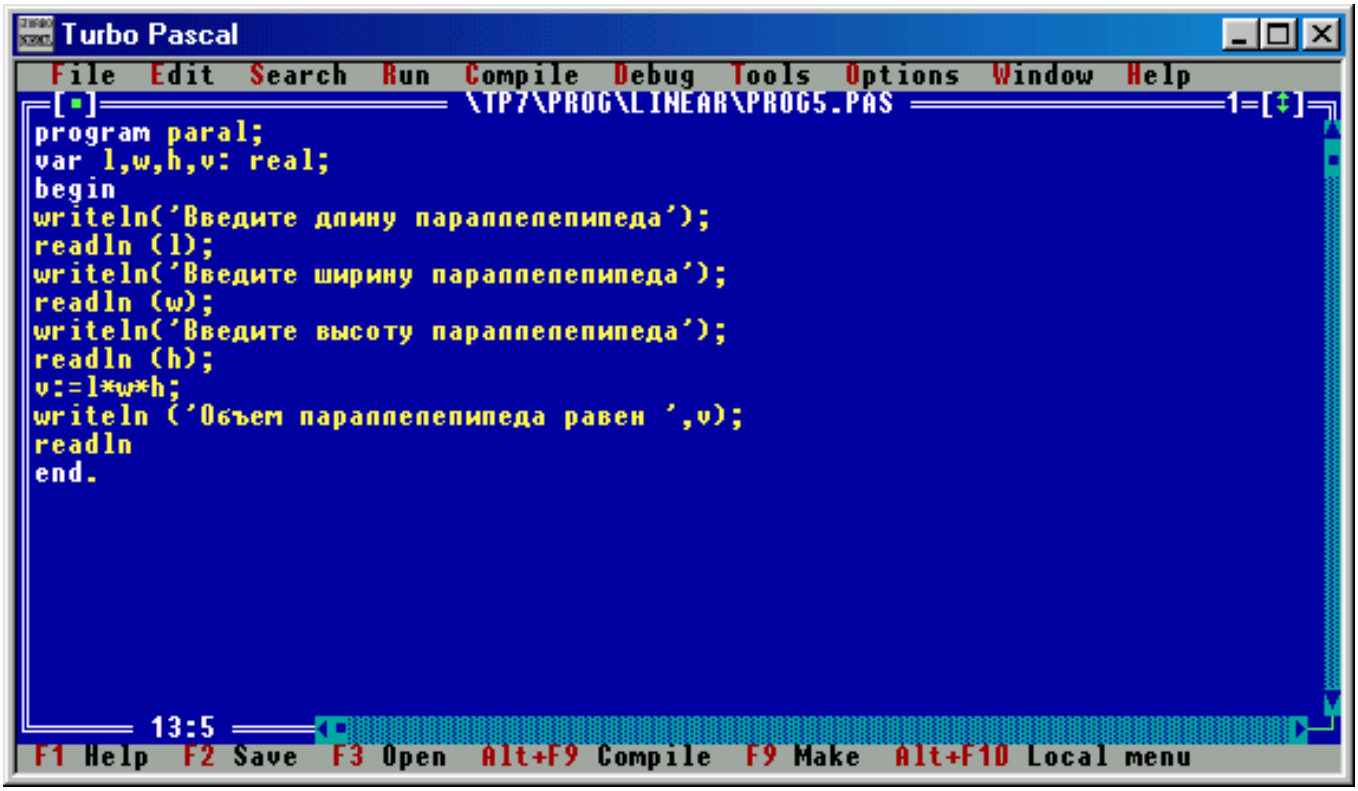
(копировать), после чего содержимое данной строки будет скопировано в специальную область памяти компьютера, называемую буфером. Затем нажатием клавиши **Enter** перемещаем курсор на строчку ниже . После этого снова открываем раздел меню **Edit** и выбираем в нем команду **Paste** (вставить), после чего хранящаяся в буфере строка будет скопирована туда, где установлен курсор. Затем спускаем курсор еще на одну строку и еще раз даем команду **Paste**. Таким образом под строкой появятся две ее копии, которые необходимо будет немного откорректировать.

С помощью команд из раздела меню **Edit** можно не только копировать какой-либо фрагмент текста (это может быть не только строка, но и группа строк), но и перемещать его. Для этого нужно выделить фрагмент так, как это было описано выше (если выделяется несколько строк, то после выделения одной строки для выделения следующих нужно несколько раз нажать клавишу «стрелка вниз») а затем использовать команду **Cut** (вырезать), после чего выделенный фрагмент временно исчезнет из программы и будет хранится в буфере до тех пор, пока его не вставят в другое место текста командой **Paste**. Необходимо при этом помнить, что буфер является одноместным, то есть при копировании или перемещении в него какой-либо информации, та информация, что раньше в нем хранилась, теряется. Поэтому перед тем, как что-либо записывать в буфер, полезно просмотреть его содержимое командой **show clipboard** (просмотр буфера). Наконец, при необходимости удалить фрагмент текста его также выделяют и затем дают команду **Clear** (очистить). Удалить одну строку можно и нажав сочетание клавиш **Ctrl + Y**.

Если в процессе редактирования Вы сделали ошибку, то можно отменить последнее изменение в тексте с помощью команды **Undo**. Если же Вы решили повторить отмененное действие, то это можно сделать командой **Redo**.

Использование вещественных чисел.

Естественно, что задачи, решаемые с помощью ТурбоПаскаля не ограничиваются такими, в которых при расчетах используются только целые числа. Для того , чтобы можно было использовать переменные, которые могут принимать вещественные значения, следует в разделе описания переменных описать их с помощью типа real. В качестве примера рассмотрим задачу вычисления объема параллелепипеда, где его высота, длина и ширина – вещественные числа (см. рис. 13). В программе, написанной для решения данной задачи используются 4 переменные . Все они описаны как вещественные. Это **l** –длина параллелепипеда, **w** – его ширина, **h** – высота и **v** – искомый объем. Исходные данные вводятся с помощью операторов **readln**, а



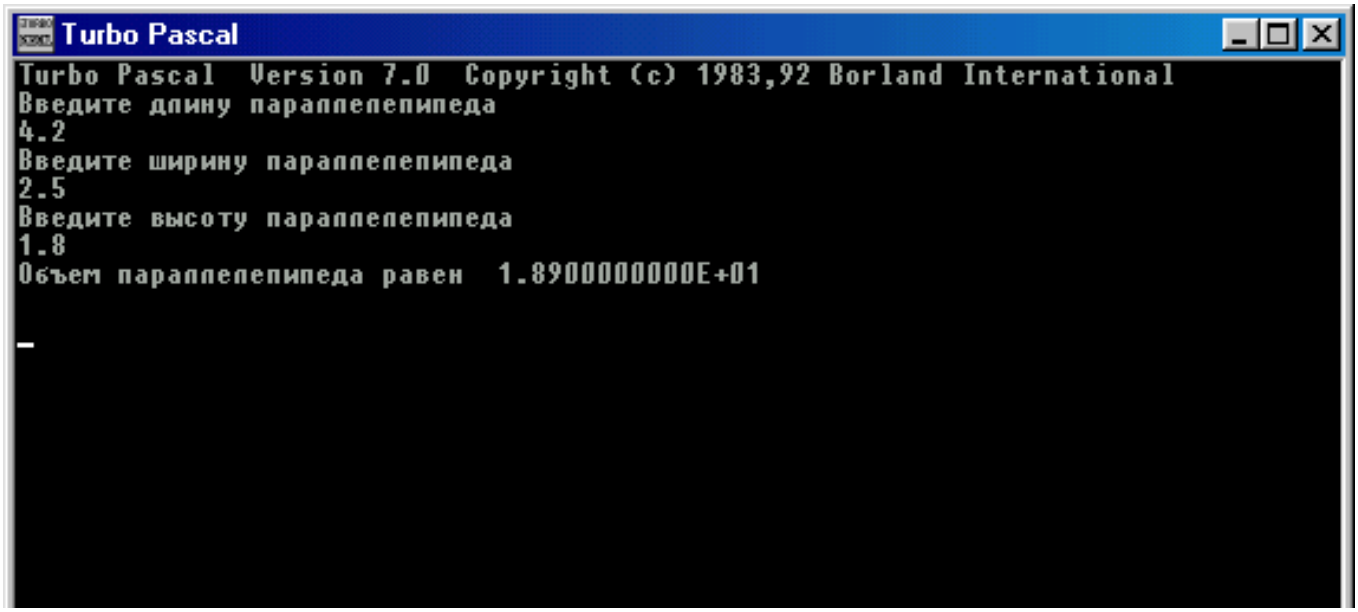


Рис. 13. Программа вычисления объема параллелепипеда и результат ее работы.

затем вычисляется произведение этих 3 величин и присваивается переменной **w**, значение которой выводится оператором **writeln.** При вводе в компьютер исходных данных целая часть от дробной отделяется не запятой, а точкой.

При длине параллелепипеда, равной 4.2, ширине – 2.5, и высоте – 1.8 получим следующий результат, показанный на рис.15.

Результат выведен в виде с плавающей точкой, то есть число представлено в виде произведения двух чисел мантиссы (десятичная дробь, расположенная слева) и характеристики (целое число, находящееся справа от буквы Е). Получившийся результат расшифровывается как 1.89×10^{1}. Именно в виде с плавающей точкой в Паскале по умолчанию представляются все вещественные числа. Такой вид удобен для представления больших чисел, но не подходит нам в данном случае. В языке Паскаль предусмотрена возможность форматировать выводимые результаты, то есть определять их внешний вид. Для этого в операторе вывода после имени переменной нужно указать после двоеточия общее количество символов, отводимое под данную переменную, а затем количество цифр в дробной части. Не нужно забывать при этом, что одну позицию нужно оставить для точки. Например, если в вышеприведенной программе оператор вывода изменить следующим образом:

writeln (‘Объем параллелепипеда равен’, v:6:2)

то под значение переменной будет использовано 6 символов, в том числе 3 символа для целой части, 1 – для точки и 2 для дробной части. Тогда при тех же исходных данных результат будет выведен в следующем виде:

Объем параллелепипеда равен 18.90

то есть результат будет представлен в виде, удобном для восприятия человеком.

Условные операторы.

При составлении программ часто возникает необходимость выполнять тот или иной вариант действий в зависимости от выполнения или невыполнения некоторого условия. Для этого в программировании используются условные операторы. В языке Паскаль используется два условных оператора – оператор **If**, который позволяет выбрать один из двух возможных вариантов действий и оператор **Case**, позволяющий выбрать один из многих возможных вариантов. В качестве условий, истинность или

ложность которых проверяется, часто используются операции сравнения. В языке Паскаль используется 6 таких операций:

< - меньше > - больше = - равно

< = - меньше или равно > = - больше или равно < > - не равно

Написание последних 3 операций отличается от принятого в математике где соответственно используются обозначения $\leq$, $\geq$ и $\neq$, но так как клавиш с подобными обозначениями нет на клавиатуре компьютера, то используются вышеуказанные сочетания из двух символов.

Результатом операции сравнения может быть величина, которая принимает одно из двух возможных значений: «истинно» (если указанное соотношение действительно выполняется) и «ложно» (если соотношение не выполняется). По-английски «истинно» и «ложно» пишется соответственно “True” и “False”. Константы и переменные, значениями которых может быть “True” или “False” называются логическими или булевскими константами и переменными. Для их описания в языке Паскаль существует специальный тип – boolean. Пример описания логической переменной:

```
var logic:boolean;
```

Над величинами логического типа можно производить логические операции подобно тому как над данными целого или вещественного типа производятся арифметические операции. Но результатом логических операций могут быть только логические величины, то есть те же “True” и “False”. Всего в Паскале используется 4 логических операции: “Not” («Нет» -логическое отрицание), “And” («И» – логическое умножение), “Or”(«Или» – логическое сложение), “Xor”(«Исключающее Или»). Эти операции подразделяются на унарные, то есть такие, которые производятся только над одной величиной, и бинарные, то есть те которые производятся сразу над двумя величинами. К первому типу операций относится “Not”, ко второму – все остальные.

При выполнении логических операций соблюдается, как и для арифметических операций, определенный приоритет. Наиболее высоким приоритетом обладает операция “Not”, то есть она выполняется в первую очередь. Далее выполняется операция “And”. Самый низкий приоритет имеют операции “Or” и “Xor”.

Подобно тому как для числовых величин существует таблица умножения, для логических величин также существует таблица, в которой указаны результаты логических операций при различных исходных данных.

Такая таблица называется таблицей истинности. Ниже приведена такая таблица для унарных и бинарных операций.

Таблица1. Унарные операции.

X	Not(X)
False	True
True	False

Из данной таблицы видно, что в результате данной операции исходное значение величины изменяется на противоположное.

Таблица 2. Бинарные операции.

X	Y	X and Y	X or Y	X xor Y
False	False	False	False	False
False	True	False	True	True
True	False	False	True	True
True	True	True	True	False

Из вышеприведенной таблицы видно, что для операции “and” значение будет равно “True” только тогда, когда обе исходных величины имеют такое же значение. Для операции “Or” значение будет “True”, если хотя бы одна из исходных величин имеет такое значение. Для операции “Xor” значение будет “True”, если значения исходных величин не совпадают.

Логические операции используют при составлении программ, в которых требуется проверить сразу несколько условий.

Оператор If.

Общий вид оператора If следующий:

If условие
then вариант 1
else вариант 2

служебные слова **if**, **then** и **else** в переводе с английского означают соответственно **если**, **то** , **иначе**. Оператор действует следующим образом: сначала проверяется, выполняется ли условие, находящееся после слова if. Если это условие истинно, то осуществляется первый вариант действий, в противном случае – второй.

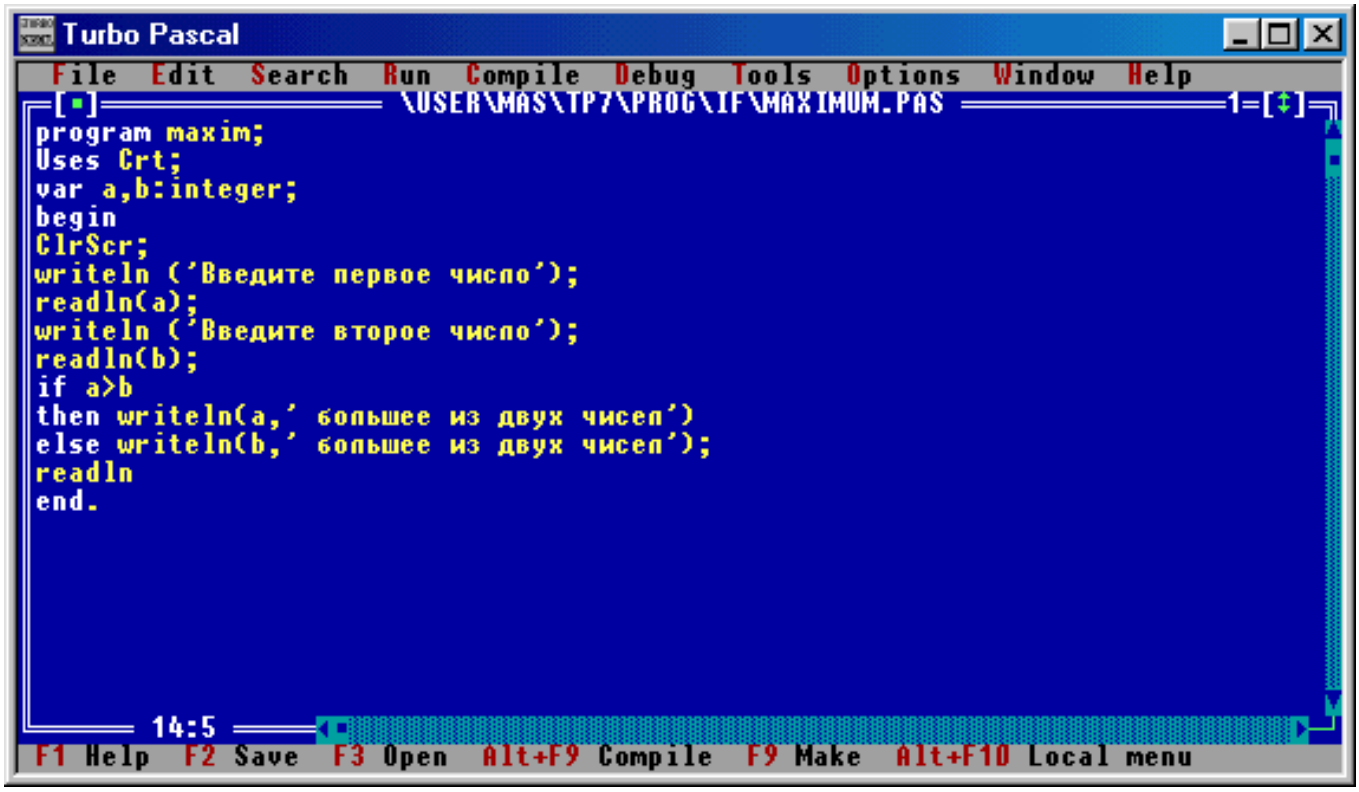
Классическим примером использования оператора **If** является программа, определяющее максимальное из двух введенных чисел. (см. рис. 14).

В данной программе вводятся два числа **a** и **b** , затем с помощью условного оператора проверяется их соотношение и если первое число больше, то оно выводится на экран , а в противном случае выводится второе число. Обратите внимание на то, что хотя конструкция **If ...Then...Else** расположена на трех строчках, между ними не ставится точка с запятой, так как вся эта конструкция представляет собой единый условный оператор.

С помощью оператора **If** несложно составить программу-тест, проверяющую знания пользователя. Например, приведенная ниже программа (см. рис.15) проверяет , знает ли пользователь дату первого полета человека в космос. Для этого введенная пользователем дата сравнивается с правильной и на экран выдается соответствующее сообщение.

Наряду с обычным условным оператором в языке Паскаль существует и сокращенный условный оператор. В таком операторе, если условие, содержащееся после **If** , истинно, то выполняется оператор после **then**, а в противном случае не выполняется никаких действий.

Как в обычных , так и в сокращенных условных операторах содержащиеся в них внутренние операторы могут быть простыми и составными. Составным оператором называется группа операторов, которая заключена между словами **begin** и **end.** Сейчас мы как раз и рассмотрим программу, которая использует данные возможности (рис. 16). Это программа



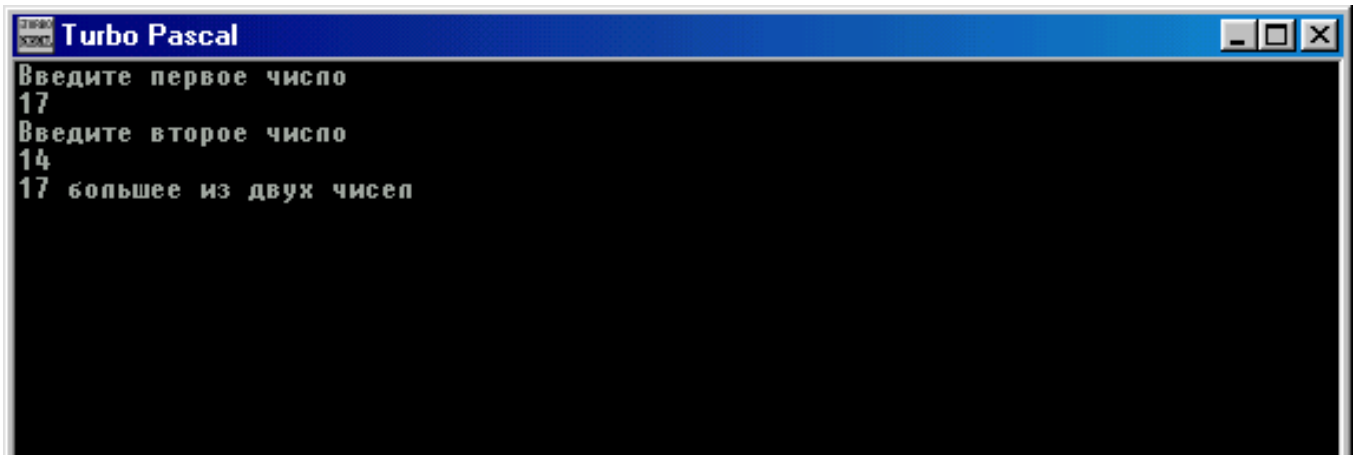
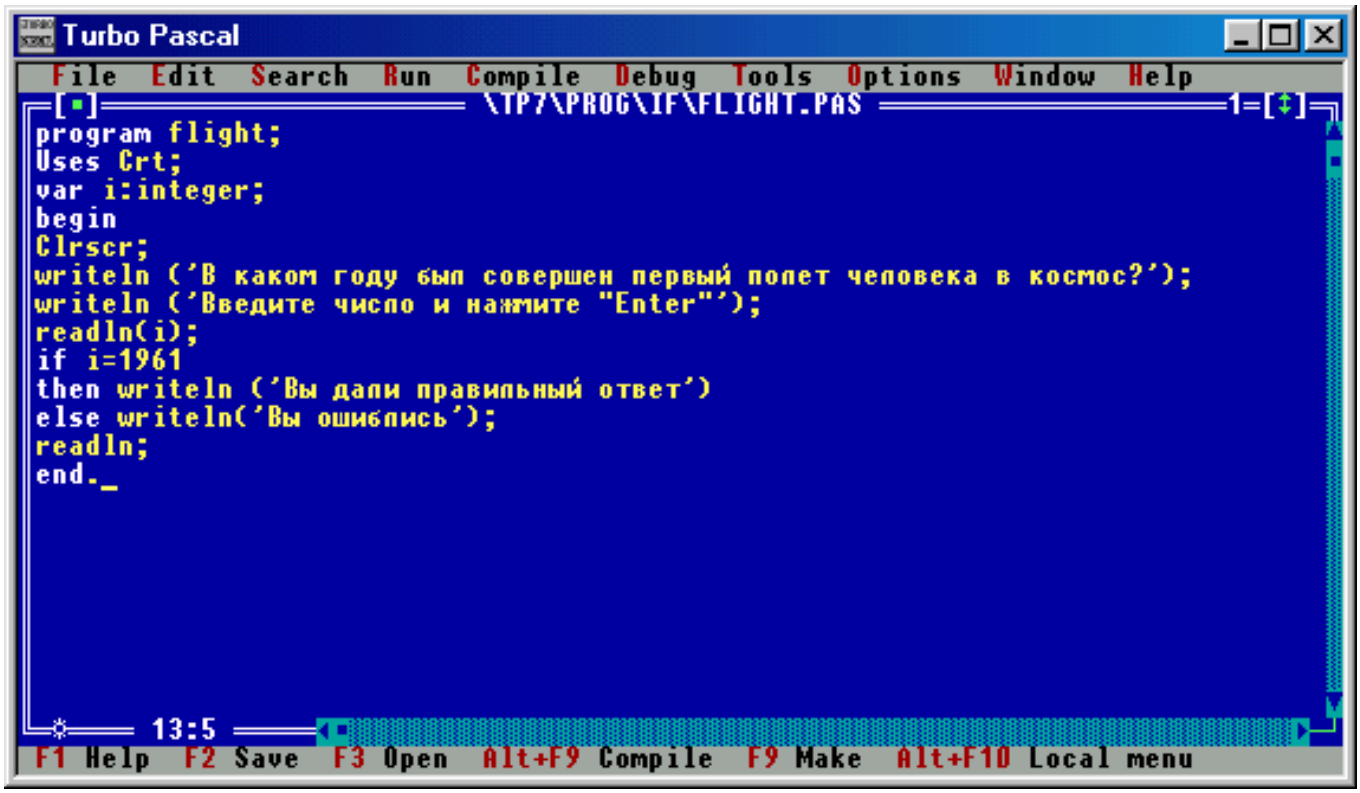


Рис.14. Программа определения большего из двух чисел и результаты ее работы.



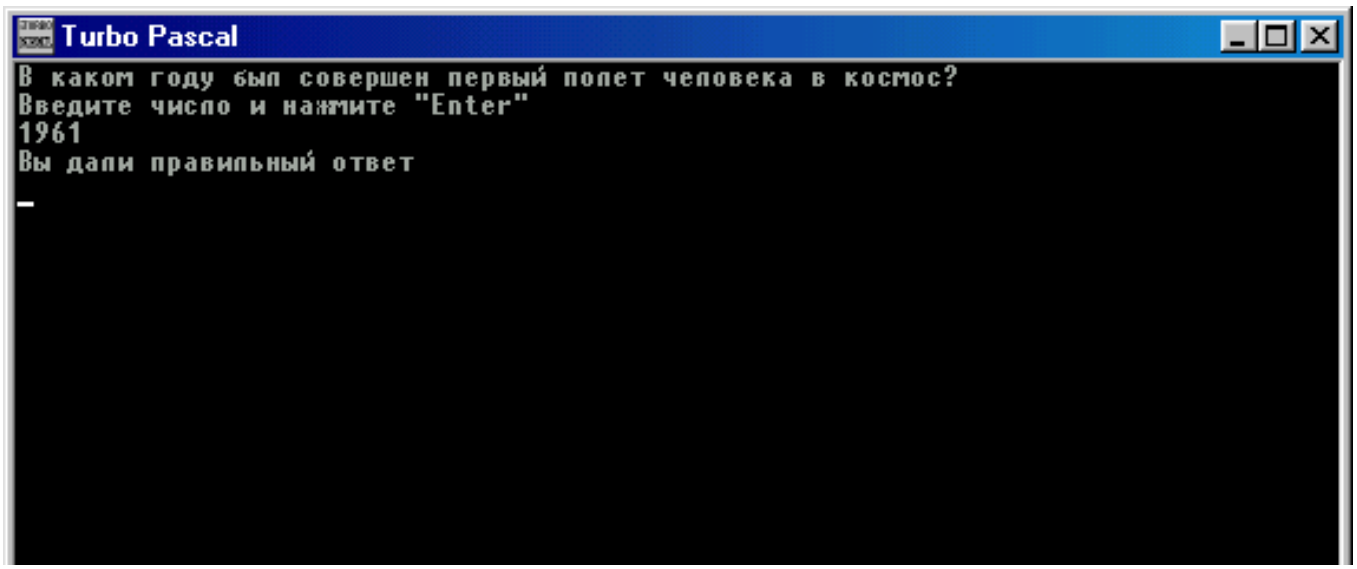


Рис. 15. Программа –тест и результаты ее работы

виртуального экзамена на знание истории освоения космоса. Пользователь должен отвечать на вопросы, которые появляются на зкране компьютера. Экзаменуемый, правильно ответивший на все вопросы получает за экзамен 5 баллов. За каждый неправильный ответ оценка снижается на 1 балл. Поэтому минимальная оценка за экзамен (при неправильном ответе на все 3 вопроса) составит двойку.

В программе для ввода ответов используется переменная **i** , очередное значение которой каждый раз сравнивается с эталонным. Данные по успеваемости хранятся в переменной **n**. На рис.16 приведен также результат работы программы.

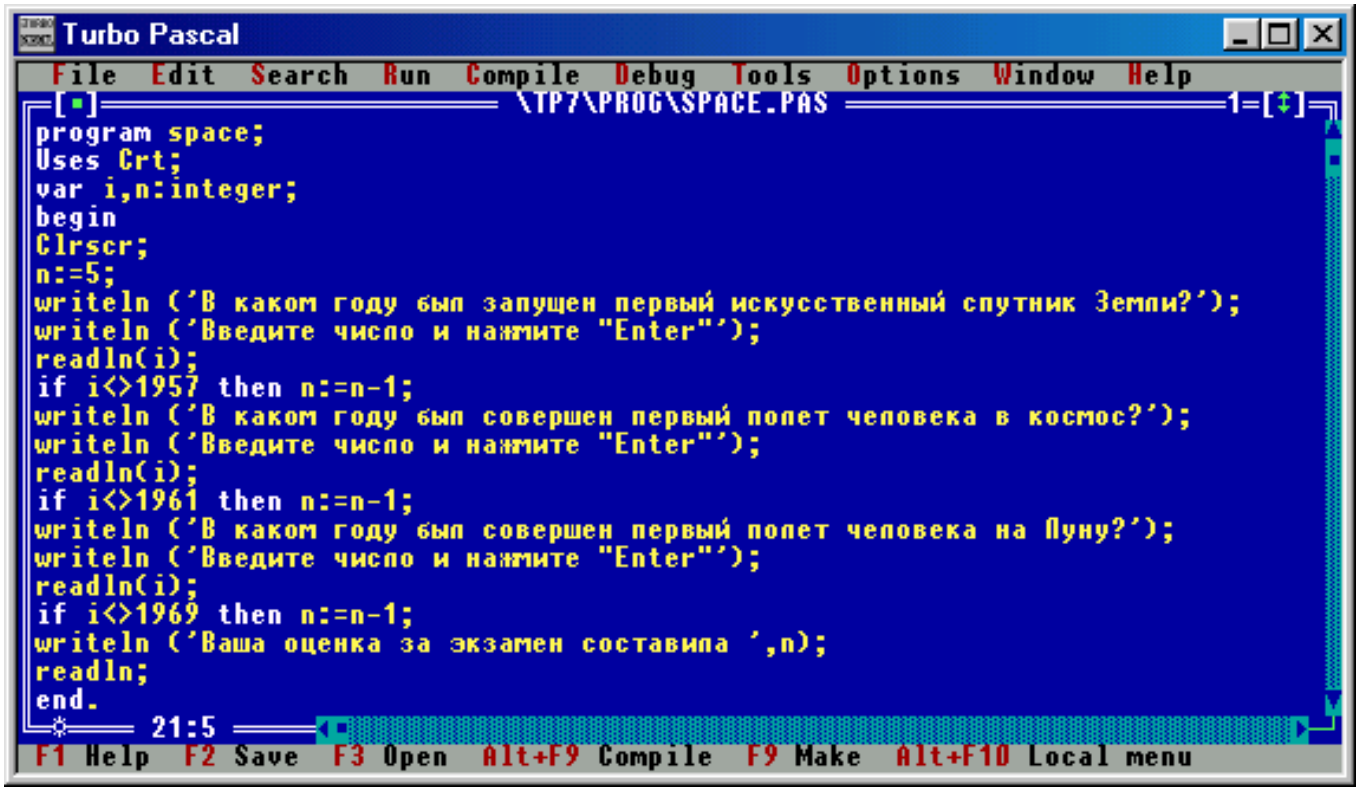
Оператор Case.

Оператор Case используется для выбора одного из нескольких возможных вариантов дальнейших действий. Общий вид оператора:

Case селектор **of**
Значение 1: Вариант1;
Значение 2: Вариант2;
.....
Значение n: Вариант N
Else Вариант N+1

где **Case**, **of** , **Else** – служебные слова языка Паскаль. Словосочетание **Case of** переводится на русский как «В случае если». Селектором называется переменная целого или символьного типа (о символьных переменных мы поговорим более подробно в разделе «Работа с символами и строками»). В зависимости от значения данной переменной выполняется соответствующий этому значению вариант действий. Этот вариант указывается после двоеточия, отделяющего его от значения. Если переменная-селектор не принимает ни одно из перечисленных в операторе значений, то выполняется альтернативный вариант, который указан после слова **Else.** Часто используется и сокращенный вариант этого оператора (без слова **Else**).

В качестве примера рассмотрим программу, которая выводит на экран компьютера строку текста тем или иным цветом в зависимости от введенной цифры (см. рис. 17).



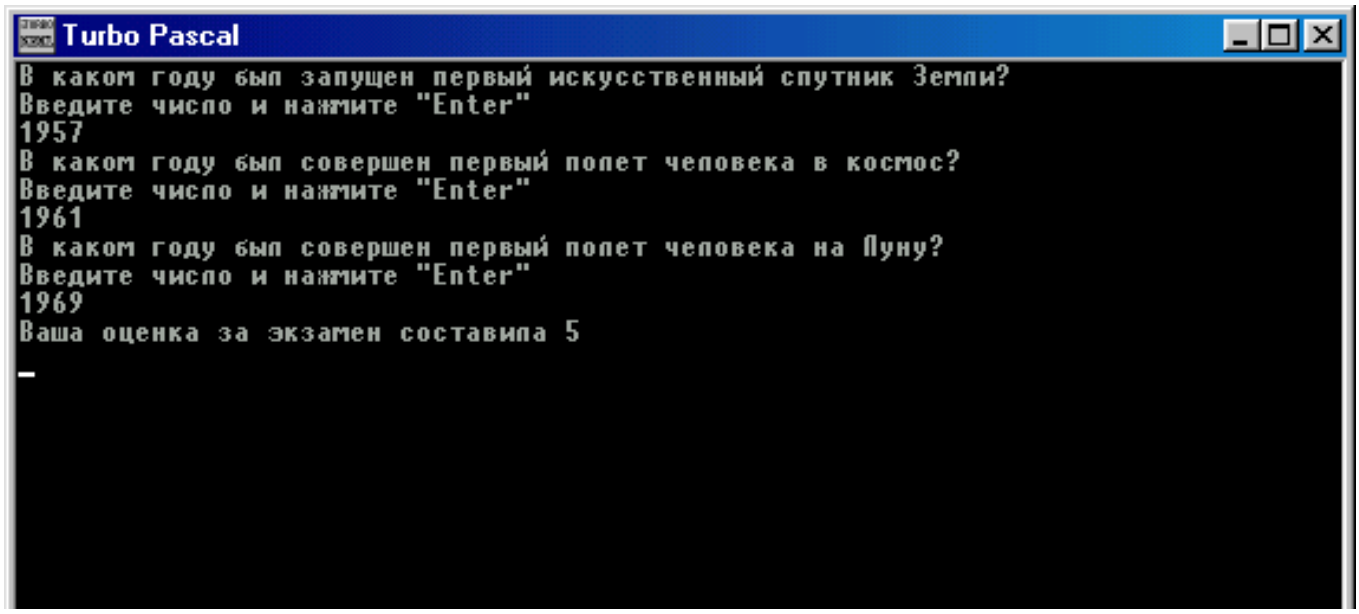


Рис. 16. Программа виртуального экзамена и результат ее работы.

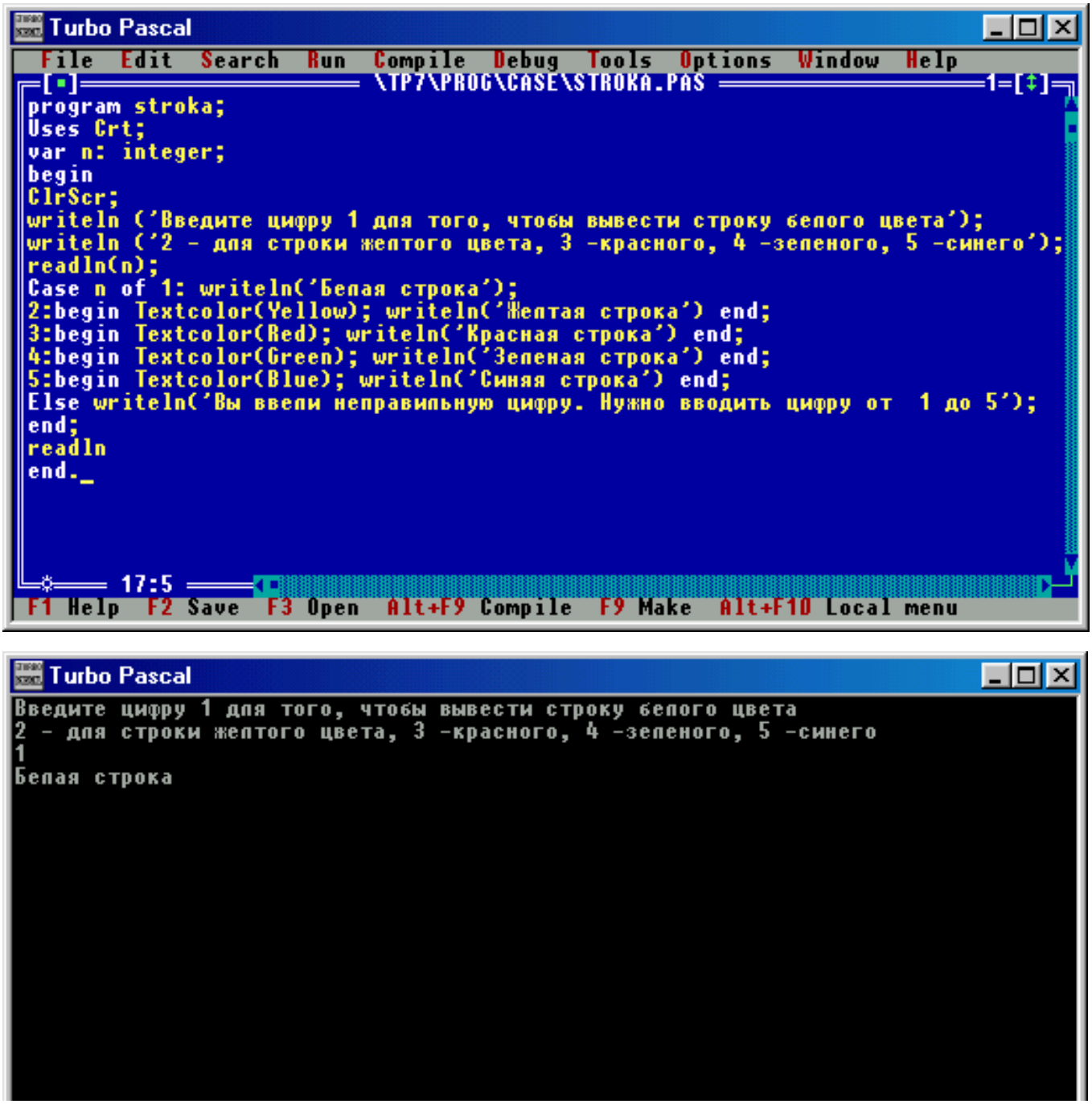


Рис.17 Программа, выводящая строки разного цвета в зависимости от введенной цифры и результаты работы программы.

В качестве селектора в данной программе используется переменная **n**. Обратите внимание, что в операторе **Case** первом варианте цвет не указывается, так как текст выводится белым цветом по умолчанию. Каждый из следующих вариантов оформлен в виде составного оператора, включающего в себя по два оператора. В программе реализована также защита от неправильного ввода данных, которая представлена вариантом, находящимся после служебного слова **Else.** (Существует жаргонное наименование такой защиты – «защита от дурака».) В случае неверного ввода данных вместо цветной строки выдается сообщение об ошибке.

Операторы цикла.

Циклом называется последовательность действий, которая может многократно выполняться в ходе работы программы.

В языке Паскаль используется 3 разновидности циклов:

1. Цикл с заранее заданным числом повторений. Его также называют циклом со счетчиком. (Цикл **for..to**).

2. Цикл с предусловием (Цикл **while**).

3. Цикл с постусловием(Цикл **repeat..until**).

Каждый из этих видов цикла имеет свои особенности и область применения, которые мы и рассмотрим ниже. Общим для всех этих операторов является то, что они включают в себя управляющие конструкции (в операторах **for..to** и **while –** это строка заголовка, а в операторе **repeat..until –** строка заголовка и завершающая строка) и тело цикла (многократно повторяемый оператор или группа операторов, которая заключается между служебными словами **begin** и **end**). Если количество повторений тела цикла заранее известно, то рекомендуется использовать оператор цикла **for..to**. В другом случае нужно использовать оператор **repeat..until** (если для решения поставленной задачи требуется, чтобы тело цикла выполнялось хотя бы один раз), либо оператор **while** (в этом случае перед выполнением тела цикла проверяется, есть ли вообще необходимость в его выполнении).

Оператор For.

Общий вид данного оператора следующий:

for i:=n1 to n2 do
тело цикла;

где **i –** управляющая переменная цикла, называемая также счетчиком цикла, **n1** – начальное значение счетчика цикла, **n2** – конечное значение счетчика цикла, **for** и **to** – служебные слова. При этом конечное значение счетчика цикла должно быть больше, чем начальное. В ходе работы данного оператора значение счетчика при каждом выполнении тела цикла увеличивается на единицу и таким образом принимает все целочисленные значения от n1 до n2 , а тело цикла всего выполняется n2-n1+1 раз.

Загололовок типа **for** может иметь и такой общий вид:

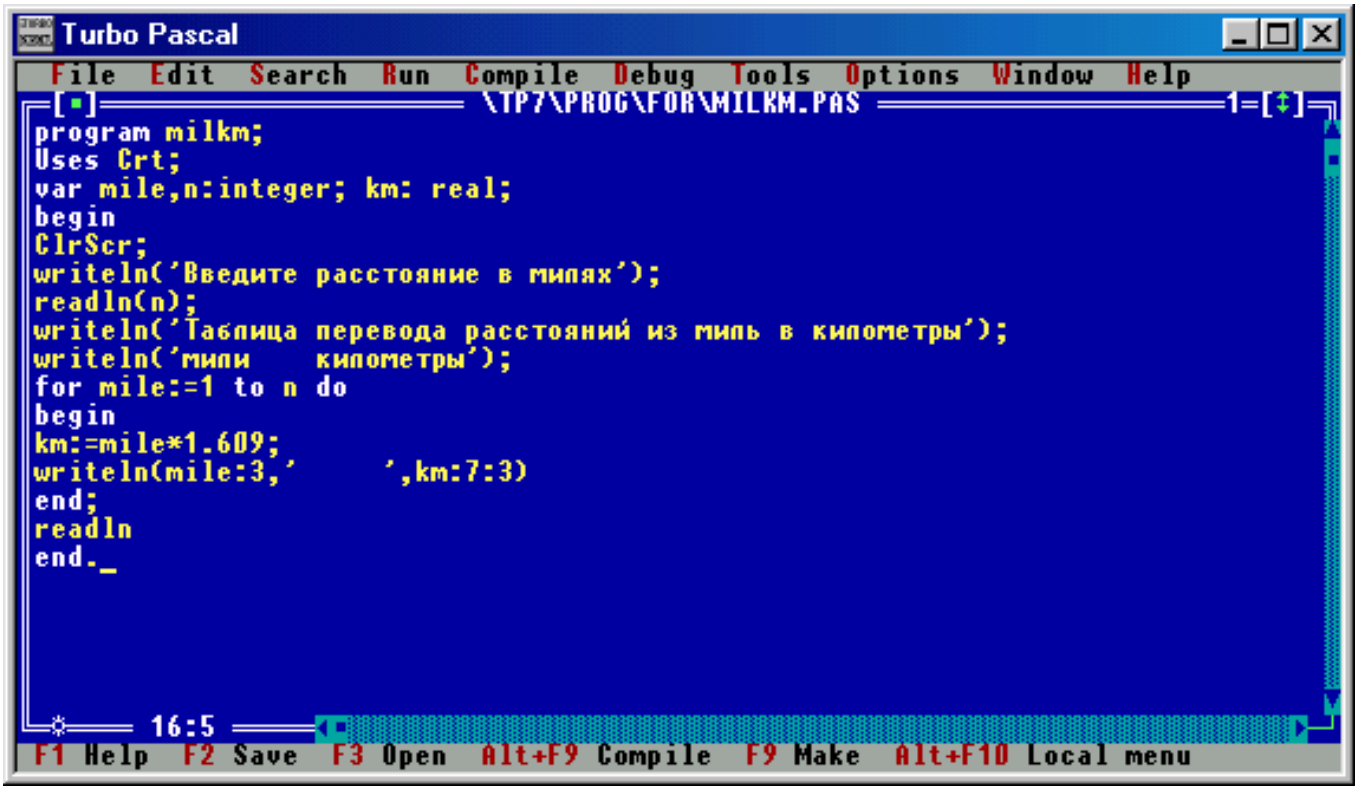
for i:= n1 downto n2 do

в этом случае конечное значение счетчика цикла должно быть меньше начального и при каждом повторении тела цикла значение счетчика **i** уменьшается на единицу.

В качестве примера работы цикла типа **for** приведем программу, которая составляет таблицу для перевода расстояния, выраженного в милях, в километры. На экран компьютера должна быть выведена таблица для расстояний от 1 до n миль, где n – целое положительное число, вводимое с клавиатуры пользователем. 1 миля составляет 1,609 километра.

В данной программе переменная целого типа **mile** содержит текущее расстояние в милях. Эта же переменная является счетчиком цикла. Начальное значение счетчика равно 1, а конечное – n. Тело цикла представляет собой составной оператор, содержащий 2 оператора и ограниченный служебными словами **begin** и **end**. В операторе **writeln,** входящем в состав тела цикла для обеих выводимых переменных используется форматный вывод. Разница заключается только в том, что в первом случае (для целочисленной переменной) указывается лишь общее количество выводимых на экран символов (в данном случае – 3), а во втором случае указывается и общее количество символов и их количество в дробной части (в данном случае соответственно 7 и 3). Это делается для того, чтобы выровнять столбцы

выводимых чисел. При n=10 получаем следующие результаты работы программы:



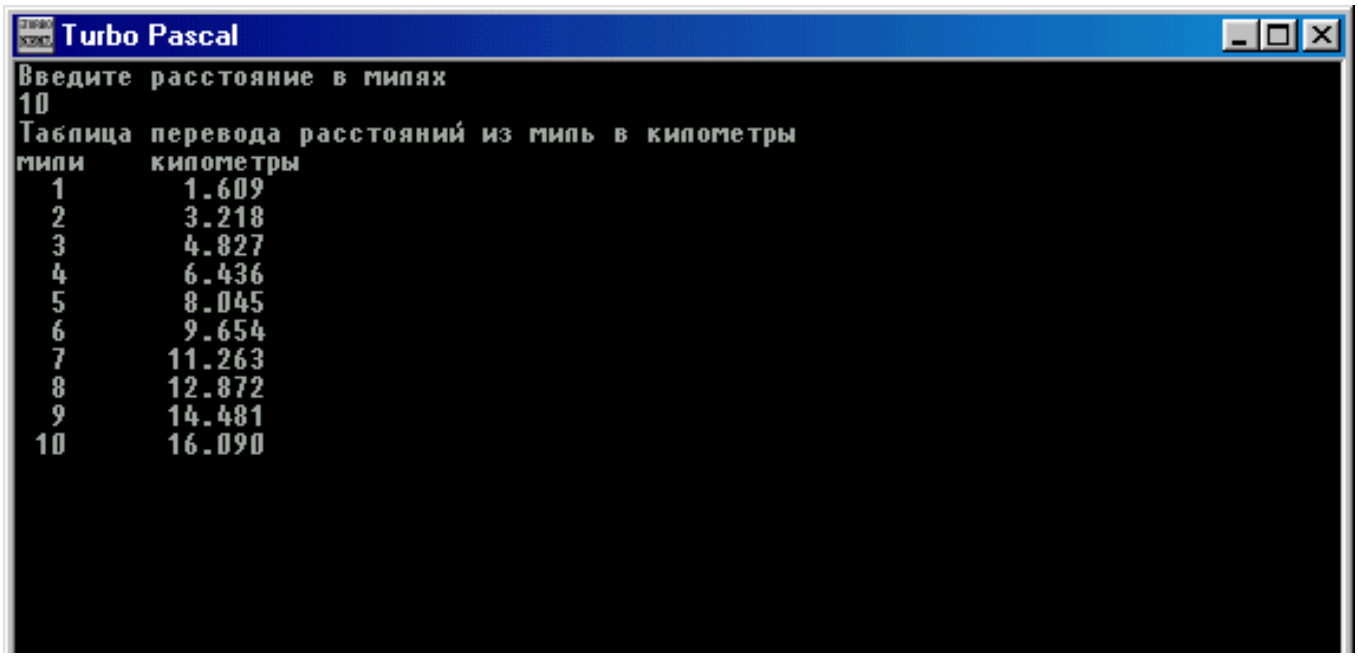


Рис. 18. Программа для пересчета расстояний из миль в километры и результаты ее работы.

Оператор Repeat … until.

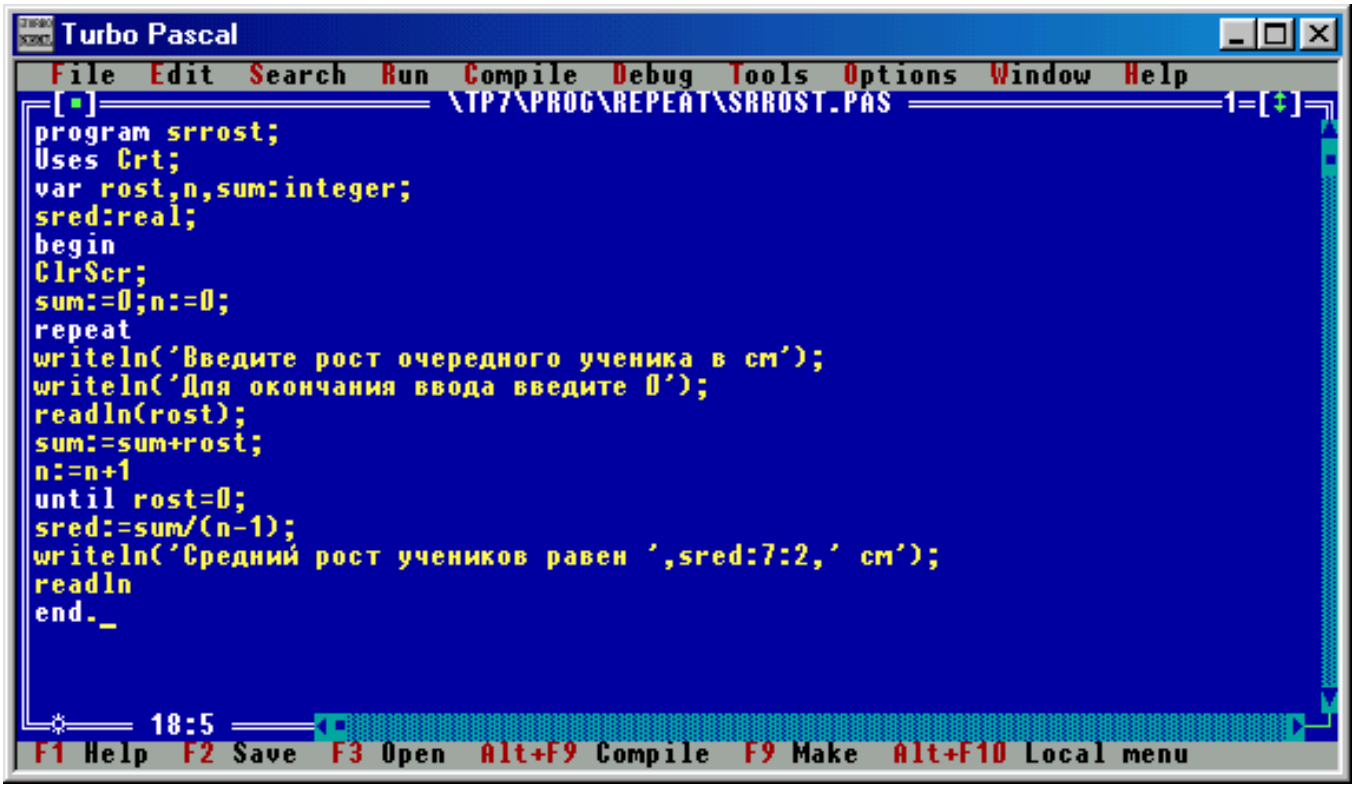
Общий вид оператора следующий:

repeat
тело цикла
until условие;

где **repeat** и **until** – служебные слова. **repeat** означает повторять, **until** – до тех пор. Цикл работает следующим образом: вначале выполняется оператор или группа операторов, входящая в тело цикла. Затем проверяется выполняется ли условие, находящееся после **until**. Если условие выполняется, то работа цикла на этом завершается, если условие не выполняется, то тело цикла выполняется снова, после чего выполняется очередная проверка и т.д.

Цикл такого типа удобно использовать для решения следующей задачи: требуется определить средний рост учеников в классе. Данные о росте каждого ученика вводятся с клавиатуры. Сколько учащихся в классе заранее неизвестно, но известно, что признаком окончания ввода исходных данных является ввод нуля. (См. рис.19)

Решение данной задачи сводится к нахождению среднего арифметического последовательности чисел (в данном случае это значения роста отдельных учеников, выраженные в сантиметрах). Для этого сначала нужно найти сумму членов последовательности, что мы и сделаем, введя переменную sum. Исходное значение этой переменной равно нулю, а при каждом выполнении тела цикла к ней прибавляется рост очередного ученика (текущее значение переменной rost). Необходимо также знать общее число членов данной последовательности. Для этого используем переменную-счетчик n. Начальное значение этой переменной также равно нулю, а после ввода очередного значения в переменную rost значение n будет увеличиваться на единицу. Конечное значение переменной счетчика получится на 1 больше, чем фактическое число учеников, т. к. счетчик сработает и при вводе нуля. Поэтому получившуюся сумму разделим на n-1. Поскольку при делении необязательно получится целое число, то переменную sred, содержащую результат, опишем как переменную вещественного типа.



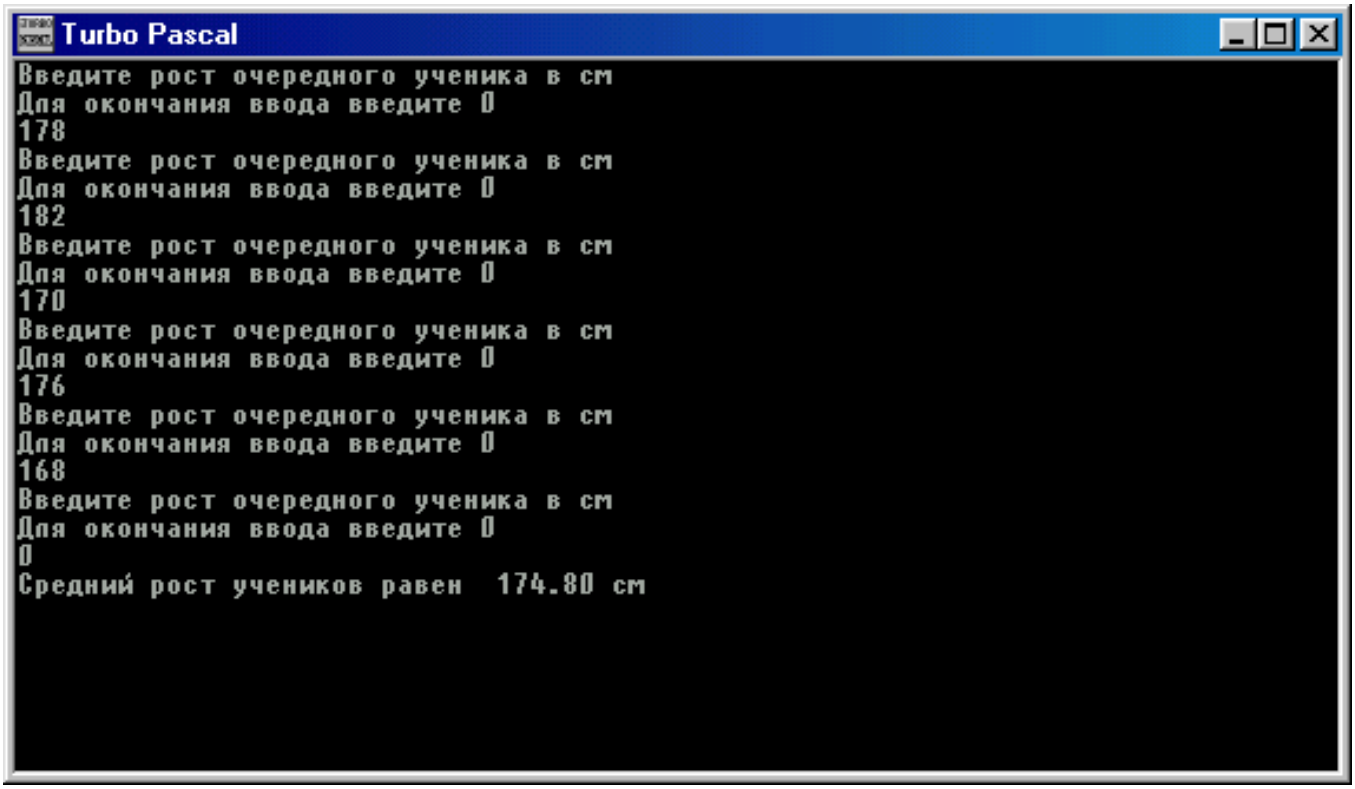


Рис. 19. Программа определения среднего роста и результаты ее работы

Оператор While.

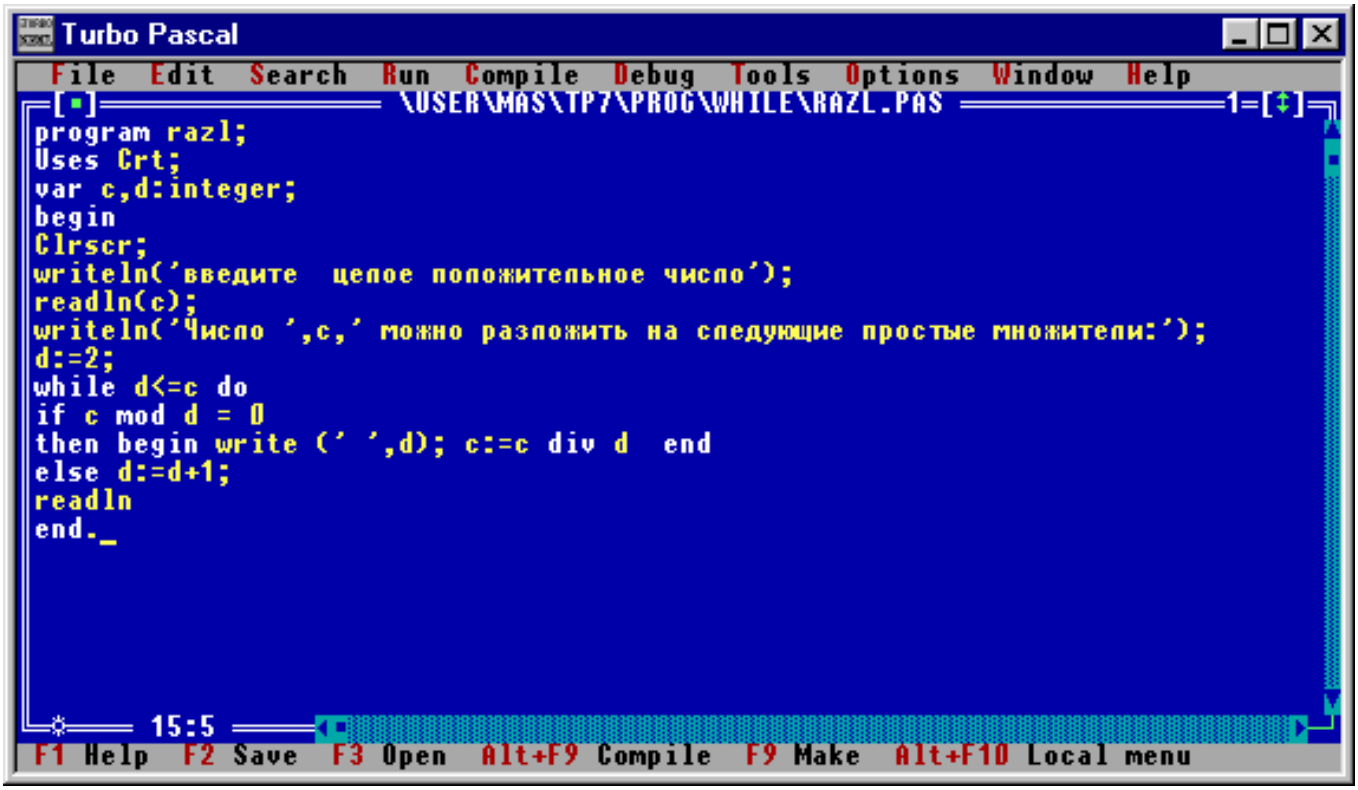
Общий вид оператора while следующий:

while условие **do**
тело цикла;

где **while** и **do** – служебные слова, **while** означает пока, **do –** делать, выполнять. Цикл работает следующим образом: первоначально проверяется условие, находящееся после слова **while**, если данное условие является ложным, то работа цикла на этом и завершается. Если условие является истинным, то выполняется оператор или группа операторов, входящая в тело цикла. Затем снова производится проверка условия, содержащегося в строке заголовка и так далее. При этом для того, чтобы программа не зацикливалась, необходимо, чтобы значения переменных, используемые в условии, изменялись в процессе работы цикла.

В качестве примера использования цикла типа **while** приведем программу разложения целого положительного числа на простые множители (См. рис. 20). Для этого введенное с клавиатуры число **c** делится последовательно на все целые числа, начиная с 2. (Поэтому переменной **d,** содержащей значения делителя присваивается начальное значение 2). Целочисленное деление производится посредством уже известной нам операции **mod.** Если число разделилось без остатка **(c mod d = 0),** то текущее значение переменной **d** является одним из простых множителей и оно выводится на экран компьютера. Вывод осуществляется оператором **write** для того, чтобы все множители были выведены в одной строчке. Переменной **c** при этом присваивается новое значение, равное частному от деления предыдущего значения на текущее значение переменной **d .** Частное находится с помощью операции **div.** Далее программа работает уже с этим новым значением **c**.

В случае же, если число не разделилось нацело, то значение делителя увеличивается на единицу и программа работает с этим новым его значением . Данная последовательность действий повторяется до тех пор, пока значение делителя не сравняется со значением переменной **c** (при этом значение переменной тоже может меняться в ходе выполнения программы). Когда значения переменных **c** и **d** станут равными, работа цикла завершается, а затем завершается и работа программы.



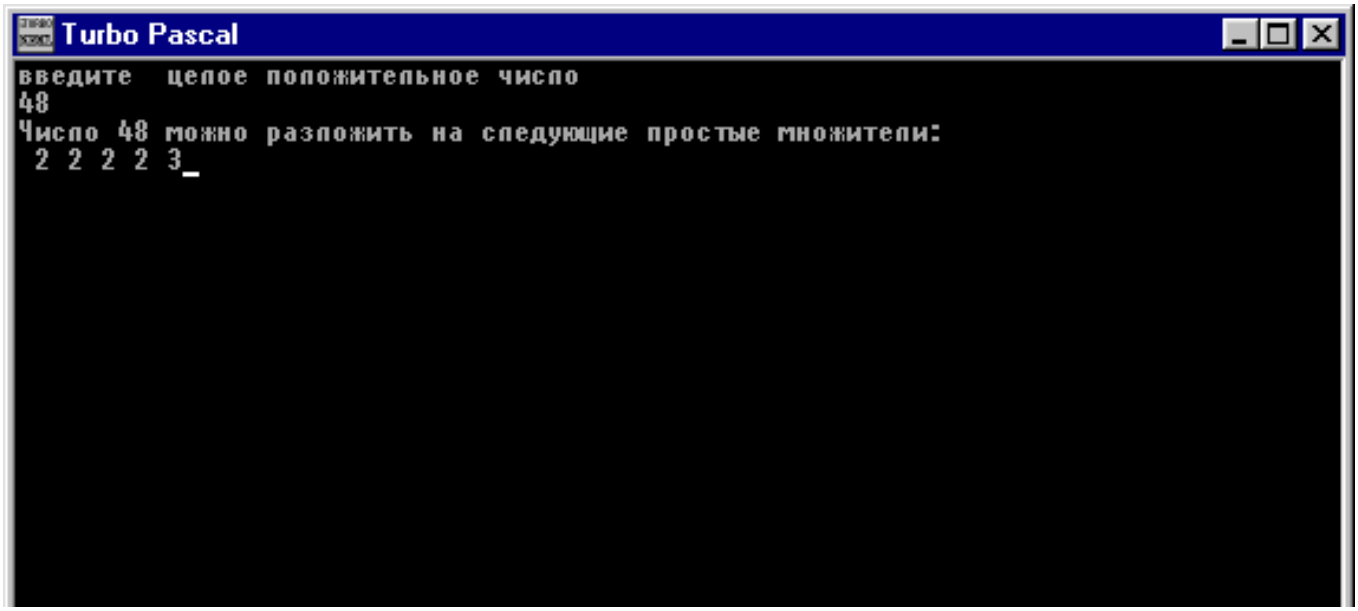


Рис. 20 . Программа разложения числа на простые множители и результаты работы программы.

Работа с символами и строками.

До сих пор мы рассматривали программы, в которых производилась обработка только числовых данных (программы из раздела «Первая программа на Паскале» не являются исключением, так как в них мы имели дело с текстовыми данными, но при этом производился их вывод, а не обработка). Но современный компьютер, как известно, представляет собой устройство, способное не работать не только с числовой информацией, но и с другими ее видами(текстовой, графической, аудио и видеоинформацией). В языке Паскаль также предусмотрена возможность обработки текстовых данных. Для работы с ними используются типы **char** и **string**.

Название типа **char** представляет собой сокращение от английского слова character, означающего в переводе «символ». Значением переменной типа char может быть один из символов, содержащихся в расширенной таблице ASCII. ASCII – это аббревиатура от American Standart Code for Information Interchange, что в переводе означает «американский стандартный код для обмена информацией». Это – используемая во всем мире таблица для кодирования вводимой в компьютер текстовой информации. Данная таблица содержит буквы латинского алфавита (прописные и строчные), цифры, символы арифметических операций и знаки препинания, а также различные специальные символы. Всего эта таблица содержит 128 символов.

К основной таблице ASCII существует дополнение, также содержащее 128 символов. Эта вторая половина таблицы зарезервирована для символов национальных алфавитов и, соответственно, для разных стран выглядит по разному. В России во второй половине содержатся, естественно прописные и строчные буквы кириллицы (русского алфавита). Следовательно, всего расширенная таблица, состоящая из международной и национальной части, содержит 256 символов. Каждый символу в таблице соответствует свой числовой код. Например, латинская буква J имеет код 74, русская буква Ж – 166, цифра 9 – 57, знак $\sqrt{}$ (радикал) – 251 и так далее.

Символьные данные могут использоваться как в константах, так и в переменных. Если в Вашей программе задействована символьная переменная, не забудьте при описании используемых в програме переменных описать ее соответствующим образом. Например так:

```
var i,n: integer;
```

```
d: real;
sim: char;
```

то есть в данной программе используются две переменных целого типа (i и n), одна – вещественного типа(d), и одна – символьного типа (sim). Если переменной символьного типа присваивается какое-либо значение, то оно должно быть заключено в апострофы. Например:

```
sim:=’ш’
```

означает что переменной sim было присвоено значение «ш».

В языке Паскаль для работы с символьными переменными используются специальные функции chr и ord.

Функция chr по известному числовому коду символа определяет сам этот символ. Например, значение chr(33) будет равно ‘!’, chr(233) – ‘щ’ и т. д.

Составим программу, которая будет выводить соответствующую группу символов для заданного диапазона числовых кодов (см. рис. 21).

Чтобы вывести результат работы программы в одной строчке, используем оператор write вместо writeln. Для числовых кодов от 176 до 215 результаты работы программы будут следующими:

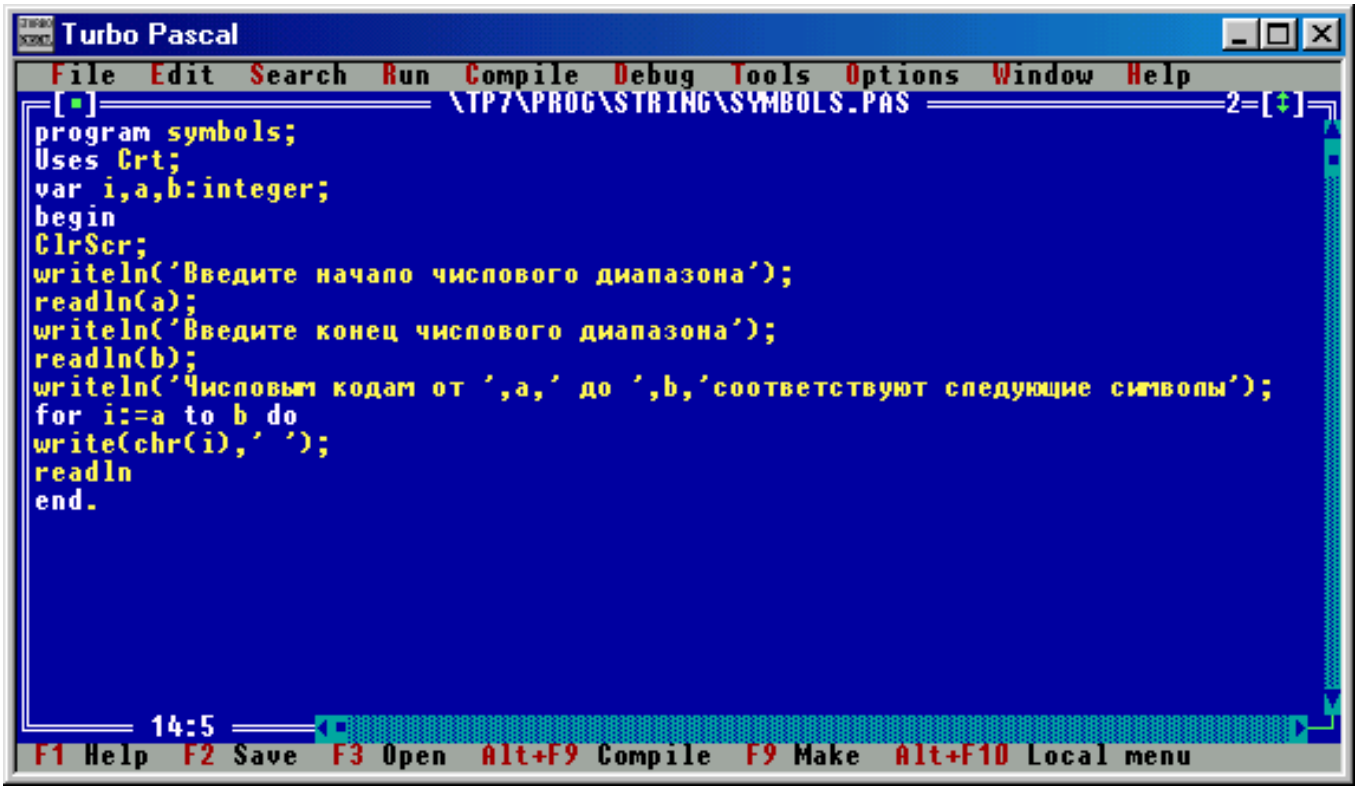
В итоге работы программы мы получили строку символов псевдографики, которые используются в операционной системе MS DOS для создания рамок, таблиц, несложных рисунков и тому подобных объектов.

Действие функции ord противоположно функции chr. Она определяет код символа, являющегося аргументом данной функции. Например, для символа ‘ж’ она выдает код 166, для символа ‘ф’ - 228 и т.д.

Естественно, что для эффективной работы с текстом необходимо уметь обрабатывать не только отдельные символы, но и слова, фразы, строки, то есть группы символов. Для обработки таких групп символов в Паскале используется строковый тип данных. Для описания символьных переменных используется служебное слово **string**. Описание символьной переменной в общем виде выглядит следующим образом:

имя_переменной: **string**[длина]

после слова string указывается длина описываемой переменной, то есть максимальное количество символов, которое может содержать данная



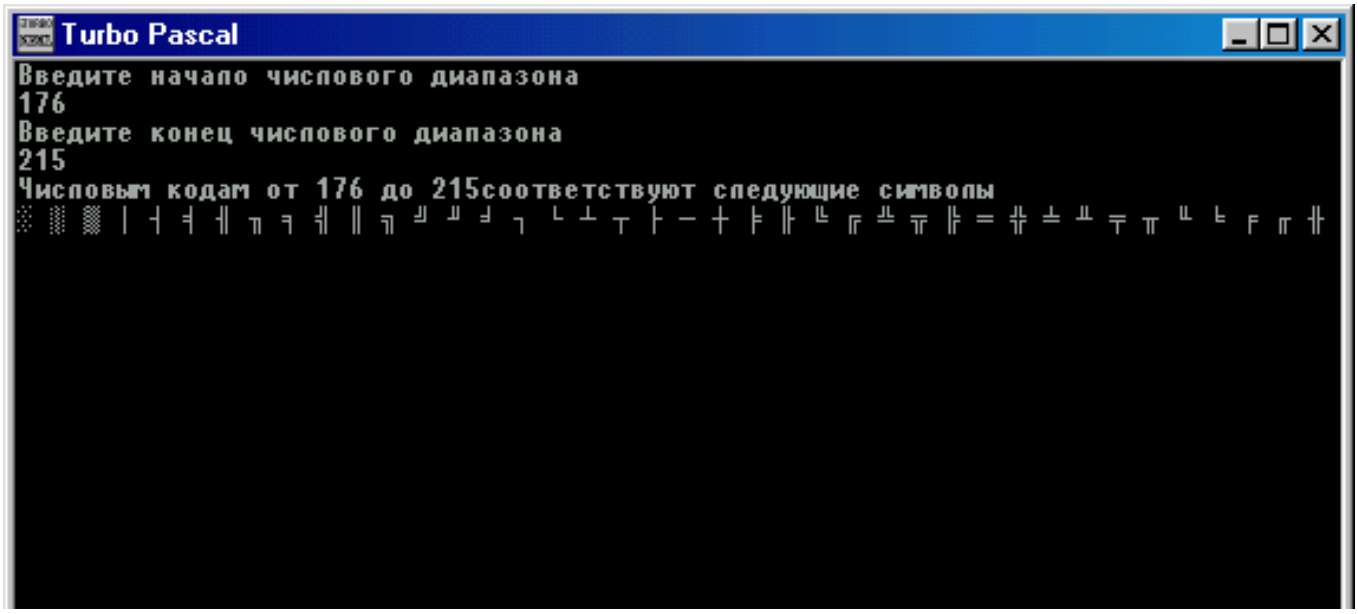


Рис.21. Программа, выводящая символы кодовой таблицы ASCII и результаты ее работы.

переменная. Например, если в разделе описания переменных в программе мы видим:

```
var i:integer;
    x:real
    stroka:string[25];
```

то это означает, что в данной программе наряду с переменными целого типа **i** и вещественного типа **x** используется и переменная строкового типа **stroka,** которая может содержать до 25 символов. Значения символьных переменных так же, как и уже знакомые нам символьные константы (к текстовым константам относятся, в частности, сообщения, выводимые оператором **writeln**) должны обязательно заключаться в апострофы. Таким образом, оператор присваивания строковой переменной **stroka** значения ‘набор_символов’ будет выглядеть следующим образом:

```
stroka:= ‘набор_символов’;
```

Для работы со строковыми переменными в языке Паскаль используются различные функции, среди которых отметим функцию **length.** Аргументом данной функции является имя какой-либо символьной переменной, а значением фактическое количество символов, которое содержится в данной переменной (эта величина может быть меньше максимальной длины, заданной при описании переменной). Так значение функции **length,** аргументом которой является переменная **stroka** после выполнения вышеуказанного оператора присваивания, будет равно 14, а не 25, указанному в разделе описаний.

Если есть необходимость работать не со всей переменной целиком, а с каким-либо отдельным содержащимся в ней символом, то к этому символу можно обратиться в программе непосредственно, указав имя содержащей его строковой переменной и его порядковый номер в этой переменной, заключенный в квадратные скобки. Например, если мы напишем в программе **stroka[5],** то работать мы будем с буквой «р», которая является пятым по счету символом в данной переменной.

При обращении к элементу строковой переменной в скобках после ее имени может указываться не только числовая константа, но и имя переменной целого типа. В этом случае номер «вызываемого» символа будет равен значению данной целочисленной переменной. Например, если мы присвоим целочисленной переменной **i** значение 10, а затем укажем в программе **stroka[i]**, то в результате будет «вызвана» буква ‘в’, которая является в строковой переменной десятой по счету.

В качестве примера использования строковых переменных рассмотрим программу, которая для введенной с клавиатуры строки символов выводит на экран компьютера последовательность соответствующих им числовых кодов таблицы ASCII (см. рис. 22).

В данной программе сразу после ввода строки определяется ее длина, величина которой присваивается переменной **dl**. Затем цикл с заданным числом повторений **dl** раз выводит один за другим числовые коды всех входящих в строку символов.

Если в ответ на приглашение программы введем фразу «Мой компьютер», то получим следующий результат, изображенный на рис. 22.

На экран были выведены числовые коды всех 13 символов, входящих в данную фразу, включая и пробел, который является четвертым по порядку символом и имеет код 32.

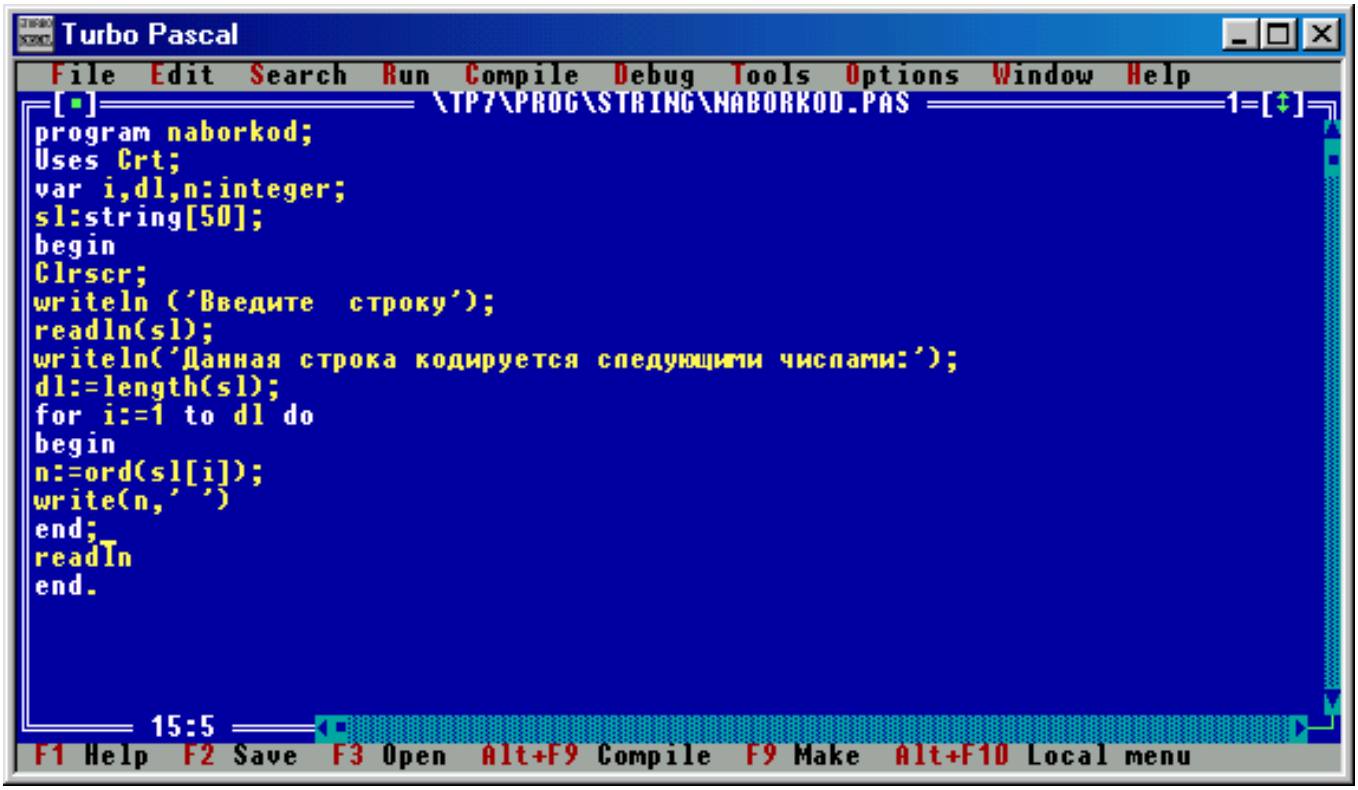
Массивы.

Массив представляет собой упорядоченную последовательность однородных элементов . В массиве каждый элемент имеет свой порядковый номер, который называется индексом. Примером простейшего одномерного массива является список учеников одного школьного класса или студентов одной группы. В этом случае элементами массива будут фамилии учеников или студентов, а индексами – номера учеников или студентов в списке. Важной характеристикой массива является его диапазон, то есть пределы, в которых может изменяться значение индекса массива.

В случае использовании массива в программе, он предварительно должен быть описан в разделе описания переменных. Но описывается массив иначе, чем обычная переменная. В общем виде описание массива выглядит следующим образом:

var имя массива: **array[a..b] of** тип элементов;

где **var, array** и **of** – служебные слова. **Array** означает массив, предлог **of** в данном случае – из, **a** и **b** – соответственно нижняя и верхняя границы диапазона массива.



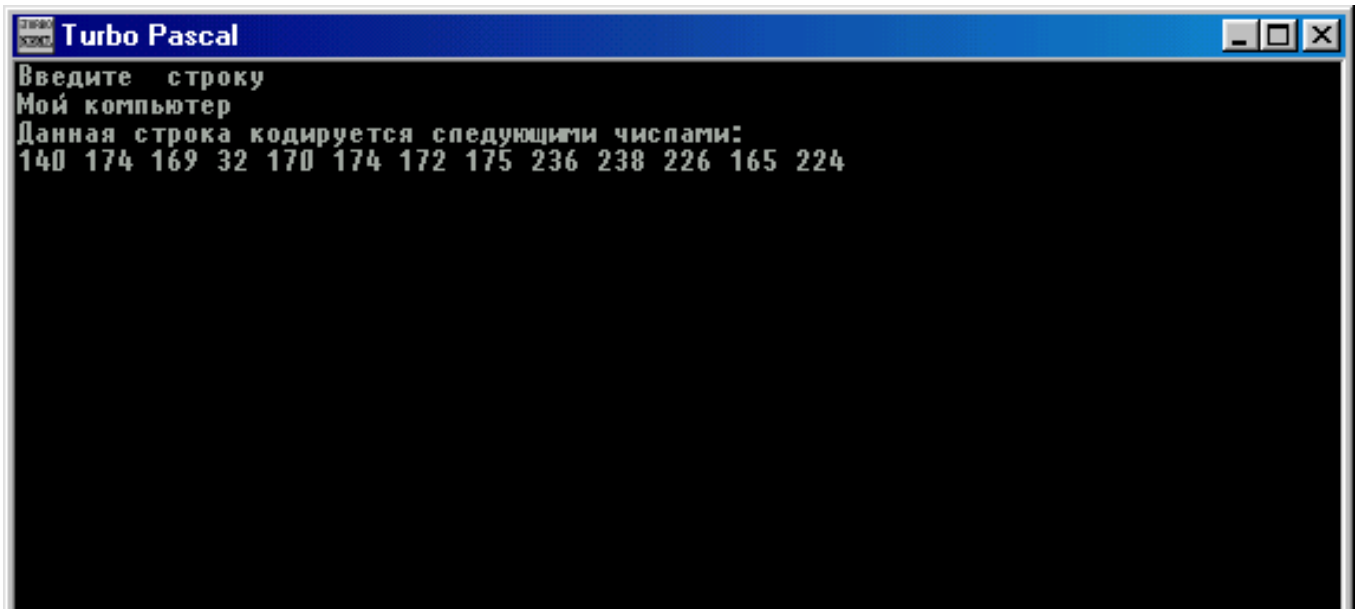


Рис. 22. Программа определения числовых кодов для вводимой строки символов и результаты ее работы.

Приведем примеры описания массивов:

var a: array[1..20] of integer;

это – описание массива с именем a, который может содержать до 20 элементов, причем все эти элементы целого типа.

var st: array[1..10] of string[15];

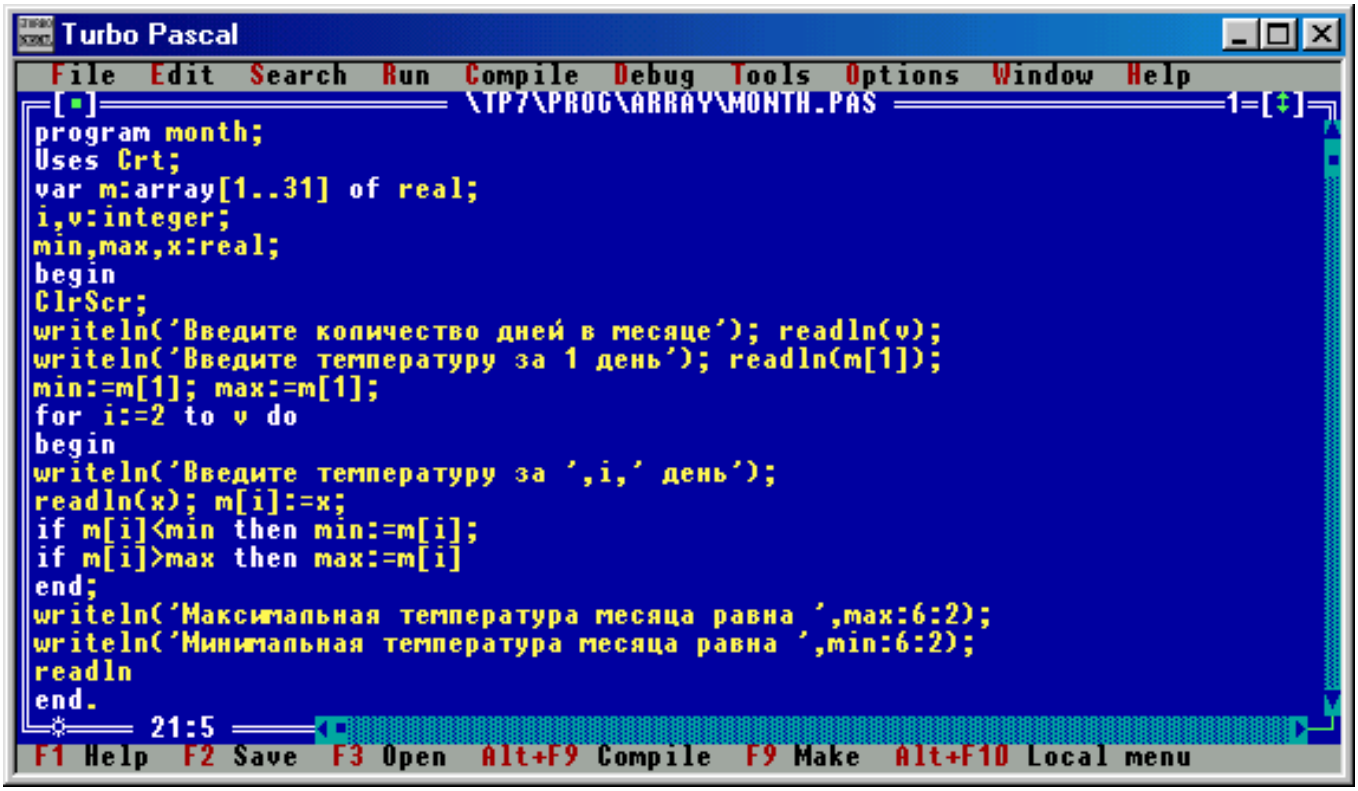
а в данном случае описан массив с именем st из 10 строковых элементов, каждый из которых может содержать до 15 символов.

В программе можно работать не только со всем массивом целиком, но и с отдельными его элементами. Для того, чтобы обратиться в программе к какому-либо элементу массива, нужно указать имя массива и индекс содержащегося в нем элемента. Например, если Вы встретите в программе следующую запись : **a[10]** , то она означает, что мы обращаемся к элементу массива **a** с порядковым номером **10**.

Рассмотрим программу определения максимальной и минимальной температуры за месяц (см. рис. 23). Данные о температуре за каждый день в программе вводятся с клавиатуры и хранятся в массиве. Нижней границей диапазона массива будет 1, а верхней 31 (максимальное число дней, которое может быть в месяце). Число дней, которое фактически содержится в том месяце, данные по которому обрабатываются программой, вводится с клавиатуры и содержится в переменной **v**. Заполнение массива будет производится в цикле с заданным числом повторений. Счетчик цикла **i** будет меняться от 2 до **v,** так как первый элемент массива заполняется не в цикле. (Почему так делается будет объяснено ниже). В ходе работы цикла очередному элементу массива присваивается соответствующее значение, равное температуре за **i –** й день месяца.

Если количество дней в месяце меньше, чем 31, то массив будет заполнен не до конца. Это вполне допустимо и никак не повлияет на работоспособность программы.

Для того, чтобы определить максимальный и минимальный элементы массива (и соответственно температуры месяца) используем следующий прием. Введем две вспомогательные переменные **min** и **max.** До начала работы цикла присвоим каждой из этих переменных значение, равное первому элементу массива (температура за первый день месяца). Затем каждый вновь вводимый (в цикле) элемент массива будем сравнивать с этими двумя переменными. Если значение элемента меньше, чем **min,** то вспомогательной



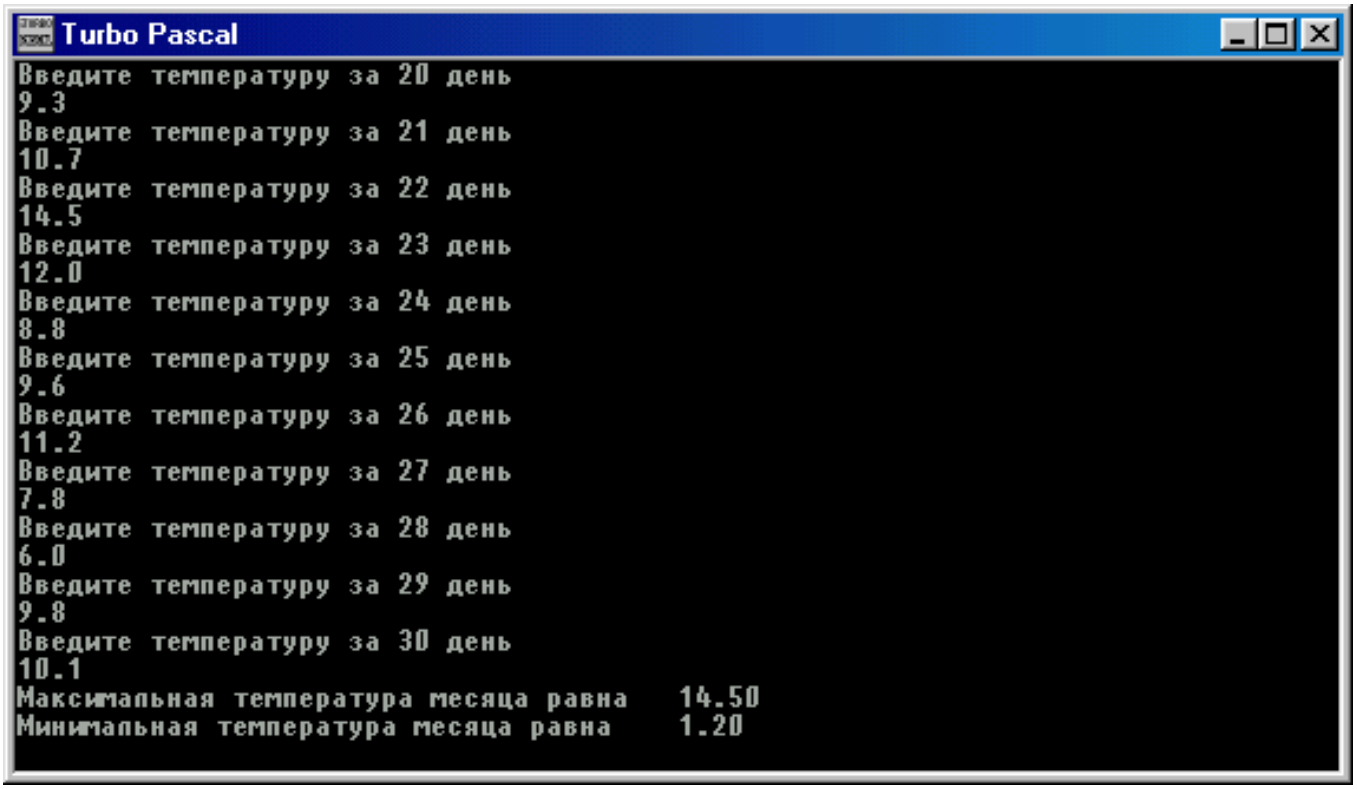


Рис. 23. Программа определения максимальной и минимальной температур за месяц и результаты ее работы.

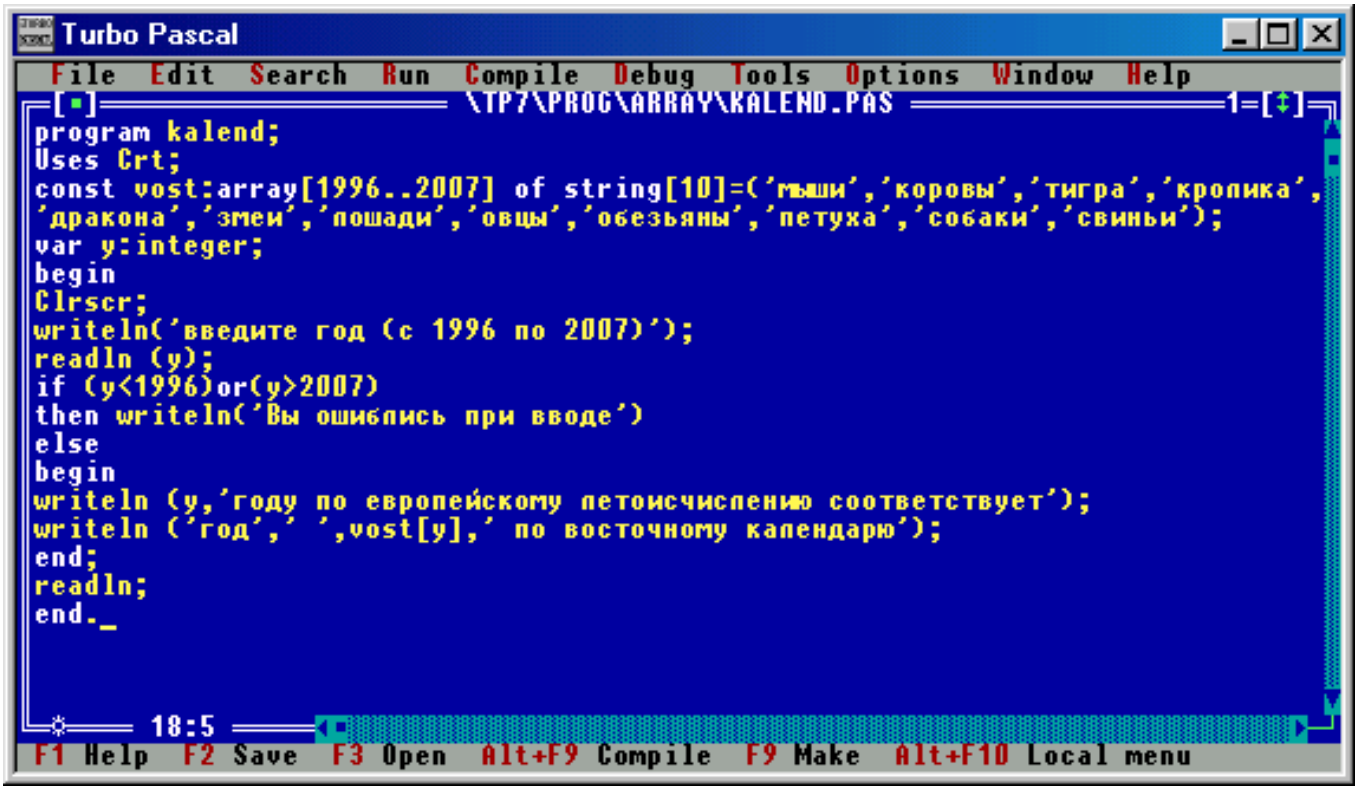
переменной присваивается новое (минимальное на текущем этапе) значение, равное этому элементу. Если же значение элемента больше или равно **min**, то переменная остается без изменения. Для этого используется сокращенный условный оператор. Аналогично производится работа с вспомогательной переменной **max.** В итоге, после завершения работы цикла переменные **min** и **max** действительно будут содержать соответственно минимальное и максимальное значения температур, которые останется только вывести на печать оператором **writeln**.

Отметим еще две особенности массивов. Во-первых, элементами массива могут быть не только переменные, но и константы. Такой массив, естественно, описывается в разделе объявлений как константа, причем константы описываются перед переменными. В этом же разделе такому массиву присваиваются значения. Во-вторых, нижней границей диапазона массива необязательно должна быть единица. Важно только, чтобы значение нижней границы было меньше, чем верхней.

Эти особенности массивов хорошо иллюстрирует программа, которая по номеру года по европейскому летоисчислению определяет его название по традиционному восточному календарю (см. рис.24). Для решения данной задачи создадим в программе массив из символьных элементов. Элементами данного массива являются строковые константы, каждая из которых содержит название одного из годов 12-летнего восточного цикла. Нижней границей диапазона индекса данного массива является 1996 – первый год очередного 12 – летнего цикла. Верхней границей 2007 – соответственно последний год данного цикла. Пользователь должен ввести год, находящийся в границах данного диапазона, и в ответ будет выведен строковый элемент массива с соответствующим индексом, то есть название года.

Так как пользователь может ошибиться при вводе исходных данных, то в программе предусмотрена их проверка, которая производится при помощи условного оператора **if.** При этом проверяются сразу два условия. Вводимое число не должно быть меньше минимального или больше максимального. Для того, чтобы оба условия проверялись параллельно, они объединены служебным словом **or**, что означает «или». Если нарушена или верхняя или нижняя граница диапазона пользователю сообщается о допущенной им при вводе ошибке.

Массивы могут быть не только одномерными как в рассмотренных выше программах, но и двумерными. Простейшим примером такого массива является всем известная таблица умножения в которой результат умножения двух чисел определяется по номеру строки, соответствующей одному из сомножителей, и



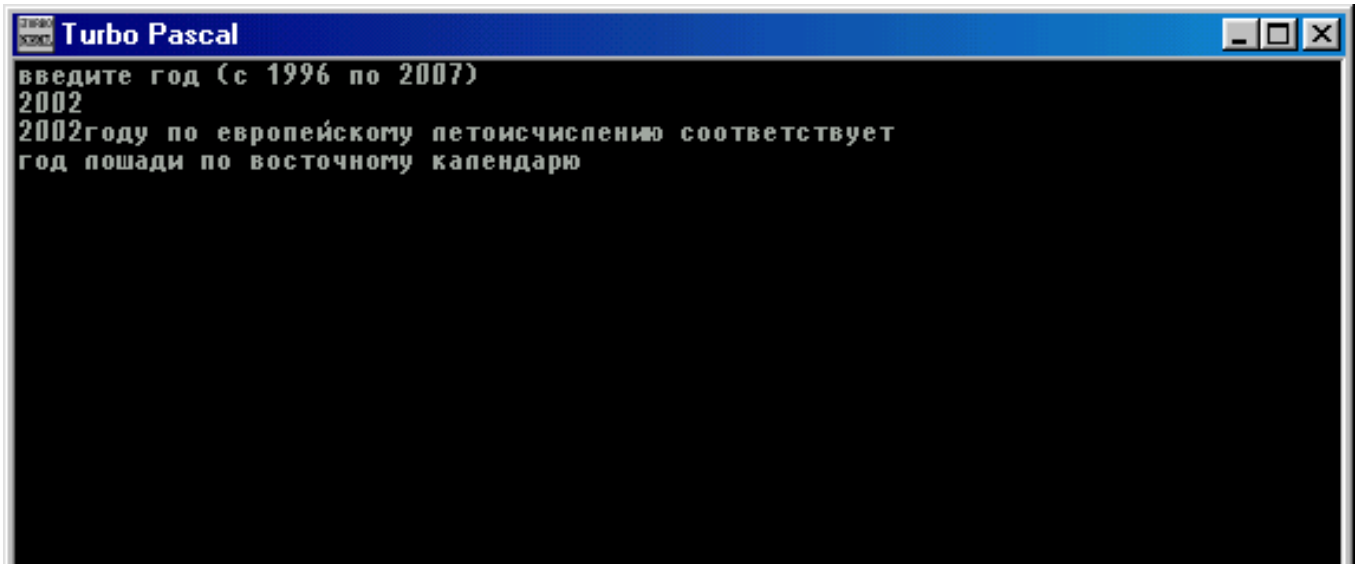


Рис. 24. Программа, определяющая название года по восточному календарю и результаты ее работы

номеру столбца, соответствующему другому. Соответственно и в любом двумерном массиве элемент определяется по двум индексам.

В языке Паскаль двумерные массивы описываются в общем виде следующим образом:

var имя массива: **array[a..b,c..d] of** тип элементов;

где a и b – соответственно верхняя и нижняя граница диапазона значений для первого индекса,

c и d - верхняя и нижняя граница диапазона значений для второго индекса.

Двумерный массив таким образом можно представить в виде таблицы, имеющей b-a+1 строк и c-d+1 столбцов.

Пример описания двумерного массива:

var tabl: array [1..9,1..9] of integer;

таким образом описан целочисленный массив, содержащий 9 строк и 9 столбцов.

Функции.

Часто при разработке программ возникает ситуация, когда в разных частях создаваемой программы приходится выполнять одни и те же действия. Нередко бывает, что аналогичные действия нужно выполнять и в разных программах. К числу таких действий относится возведение числа в квадрат, извлечение квадратного корня, определение модуля (абсолютной величины) числа, вычисление тригонометрических функций, округление дробного числа до ближайшего целого, определение длины строковой переменной и другие.

Для того, чтобы облегчить процесс составления программ в языке Паскаль предусмотрены специальные функции, которые используются для автоматического выполнения этих действий. Такие функции называются стандартными и являются неотъемлемой частью языка Паскаль.

Каждая из этих функций по вводимым в нее исходным данным, определяет некоторый, результат, который далее используется в программе.

Исходные данные называются аргументом функции, а вычисленный результат – ее значением. При обращении к функции аргумент указывается в скобках.

К наиболее широко употребляемым стандартным функциям помимо уже известных нам функций **length**, **ord** и **char**, используемых для работы со строковыми переменными, относятся также следующие:

abs(x) – определяет абсолютное значение аргумента, которым может быть число или выражение целого или вещественного типа.

arctan(x) – вычисляет арктангенс угла, значение которого выражено в радианах.

cos(x) – вычисляет косинус угла, значение которого выражено в радианах.

exp(x) – вычисляет экспоненту аргумента (то есть **e** в степени **x**).

ln(x) – вычисляет натуральный логарифм аргумента (т. е. логарифм по основанию **e**).

sin(x) - вычисляет синус угла, значение которого выражено в радианах.

sqr(x) – вычисляет квадрат аргумента, которым может быть число или выражение целого или вещественного типа.

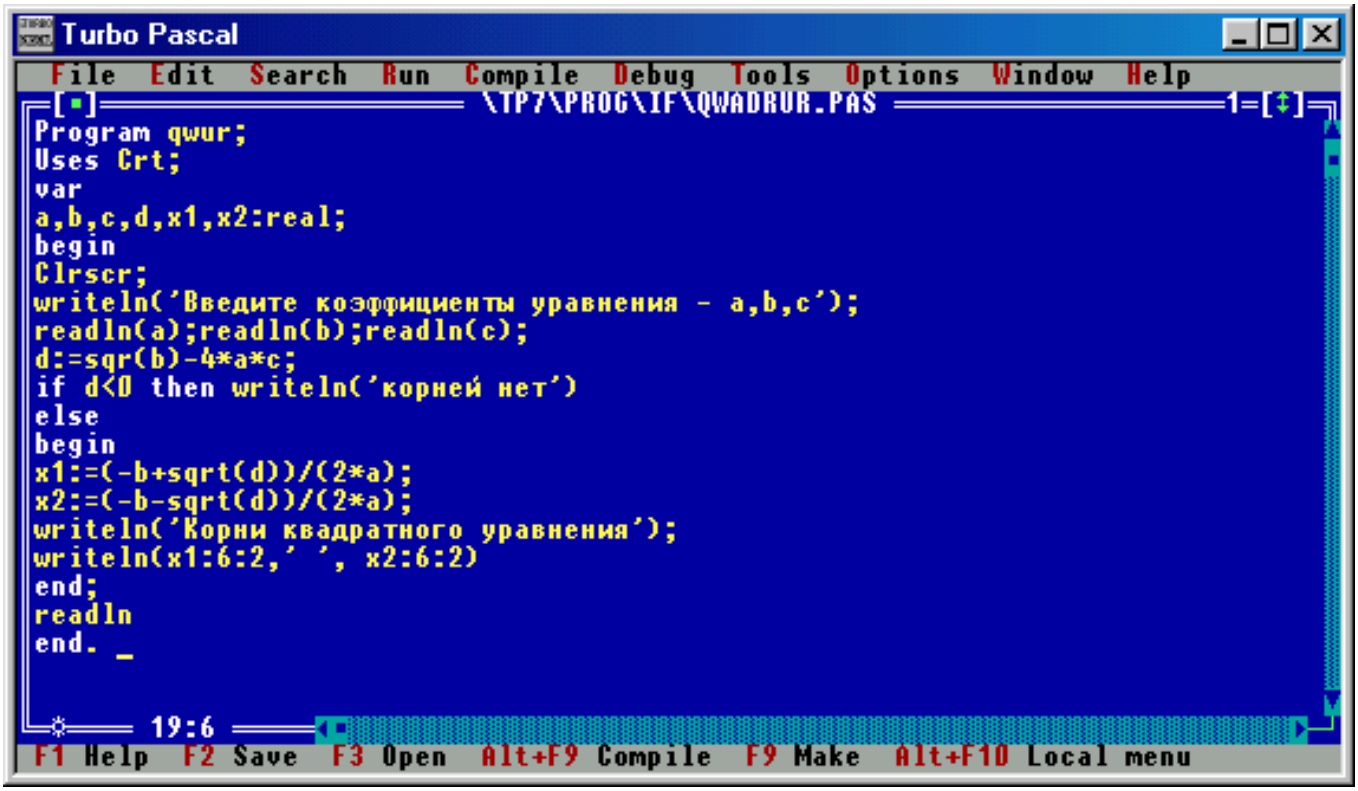
sqrt(x) – вычисляет квадратный корень из аргумента.

round(x) – округляет значение аргумента до ближайшего целого числа.

str(x) – преобразует числовое выражение в строку.

val(x) – преобразует строку, изображающую целое или вещественное число, в число.

В качестве примера программы с использованием стандартных функций рассмотрим программу решения квадратичного уравнения $\mathbf{ax^2+bx+c=0}$ **(**см. рис. 25).



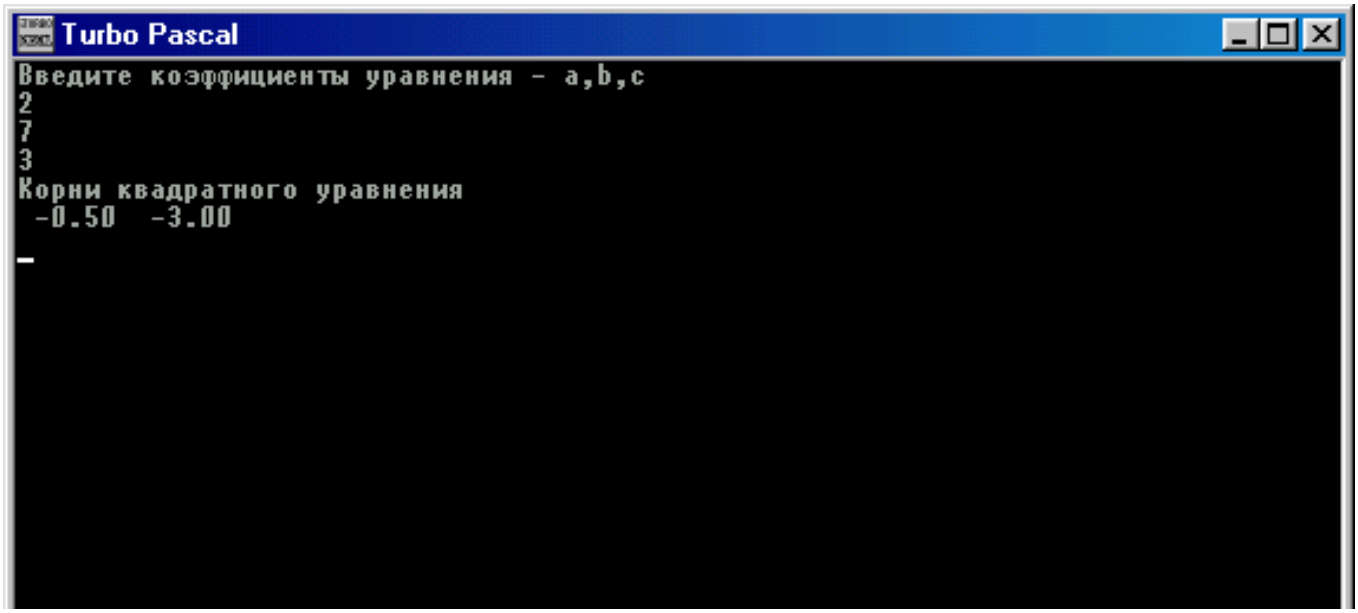


Рис. 25. Программа решения квадратичного уравнения и результаты его решения.

В данной задаче сперва с помощью стандартной функции **sqr** вычисляется дискриминант уравнения. В зависимости от значения дискриминанта выясняется имеет ли данное уравнение решение. Если решения нет, то соответствующее сообщение выводится на экран компьютера. В случае же, если решение имеется, оно определяется с помощью стандартной функции **sqrt**.

При значениях коэффициентов уравнения 2, 7 и 3 получаем следующее решение (см. рис. 25).

Набор стандартных функций языка Паскаль достаточно обширен (он не ограничивается вышеприведенным списком), однако в ряде случаев программисту может потребоваться для решения поставленной задачи создать свою собственную функцию. Такая функция должна быть описана в тексте программы после раздела описания констант и переменных и до начала ее основной части (то есть до слова **begin**). Структура описания создаваемой программистом функции выглядит следующим образом:

заголовок функции;
раздел описания констант и переменных, используемых внутри функции;
begin
операторы функции
end;

Теперь разберем более подробно элементы этой структуры.

Общий вид заголовка функции следующий:

function имя_функции (параметры функции): тип функции;

где **function** – служебное слово, означающее функция; имя функции дается по тем же правилам, что и имена переменных, в скобках указываются аргументы функции, называемые ее параметрами, причем для каждого параметра обязательно должен быть указан его тип (если параметры относятся к одному типу, то они перечисляются через запятую, а после двоеточия указывается их общий тип, если же параметры относятся к разным типам, то они отделяются друг от друга точкой с запятой); после скобок обязательно указывается тип значения самой функции.

Пример заголовка функции:

function beta (x,y:integer; z:real):real;

данная функция имеет имя **beta**, в ней используются 3 параметра: **x** и **y** – целого типа, **z** – вещественного, а значение самой функции является вещественным.

Таким образом заголовок функции в целом напоминает описание переменной, но следом за заголовком в описании функции указываются оператор или группа операторов, по которым вычисляется ее значение. Эти действия записываются в виде составного оператора, который начинается со служебного слова **begin** и заканчивается словом **end**. В составном операторе могут использоваться свои, локальные переменные. Раздел описания этих переменных (для каждой из локальных переменных также должен быть указан ее тип) помещается между заголовком и составным оператором. Не забывайте о том, что в составном операторе обязательно должен быть оператор присваивания, который присваивает получившийся результат функции.

В основной части программы действия, указанные в разделе описания функции, выполняются тогда, когда необходимо найти ее значение. Для этого необходимо осуществить ***обращение*** к функции. Обращение к функции в основной части программы включает в себя имя функции и следующий за ним список параметров, заключенный в скобки. Параметры, указанные при обращении к функции, называются фактическими параметрами, а параметры указанные в описании функции – формальными. Фактические параметры должны быть того же типа, что и формальные.

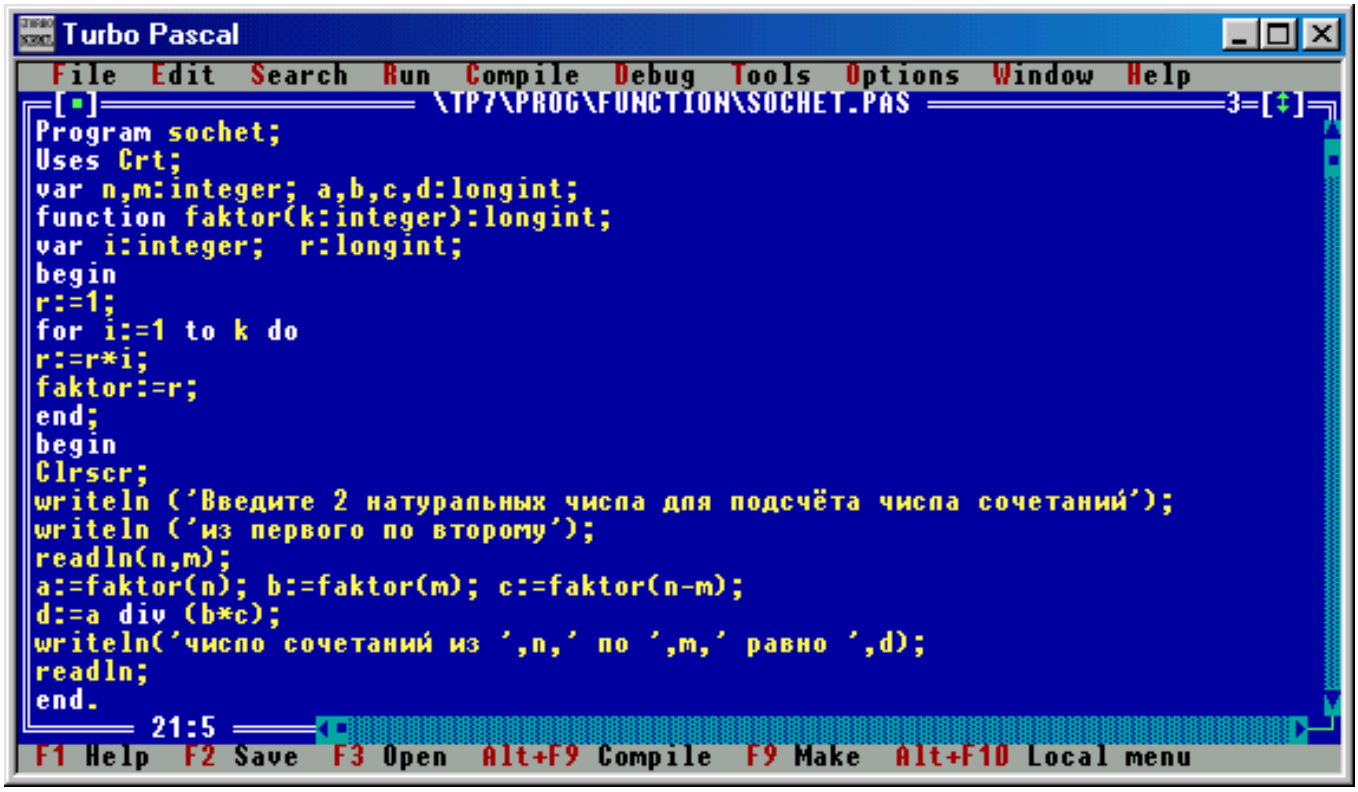
Например, правая часть оператора присваивания

d:=beta(3,4, 7.5)

представляет собой обращение к функции **beta**, а 3, 4 и 7.5 – фактические параметры данной функции в отличие от формальных параметров **x,y** и **z**. В качестве фактических параметров функции могут выступать и константы (как в вышеприведенном примере) и переменные. Обратите внимание, что все фактические параметры перечисляются через запятую, даже если они принадлежат к разным типам.

Рассмотрим программу определения числа сочетаний из n по m. Это число определяется по следующей формуле:

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



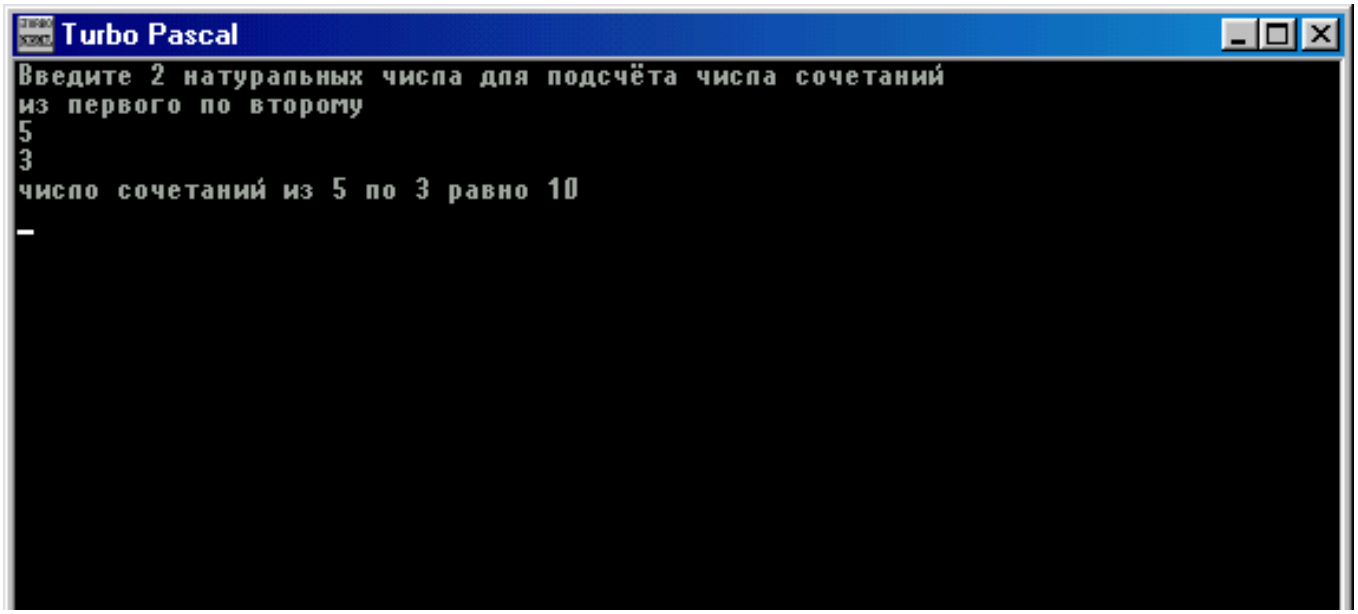


Рис.26. Программа подсчета числа сочетаний из n по m и результаты ее работы.

где **n!,m!** и **(n-m)!** – соответственно факториалы **n,m** и **(n-m).** Факториалом числа **n** называется произведение всех натуральных чисел от 1 до **n**. Так как в программе нам предстоит три раза вычислять факториал различных чисел, то в целях рационализации программы вычисление факториала оформим в виде отдельной функции **faktor**.

Функция вычисления факториала состоит из оператора присваивания, в котором вспомогательной переменной **r** присваивается начальное значение 1, цикла с заданным числом повторений, в котором **r** последовательно умножается на числа от 1 до **k** и оператора присваивания, в котором функция **faktor** получает значение, равное конечному значению **r.** Так как факториалы даже небольших натуральных чисел представляют собой достаточно большие величины (например факториал числа 10 равен 3 628 800), то для описания значения функции используется тип **longint.**

В основной части программы осуществляется ввод исходных данных, затем производятся элементарные вычисления по формуле с использованием значений функции faktor и полученный результат выводится на экран. Ниже приведен результат работы программы при **n**, равном 5 и **m**, равном 3.

Процедуры.

При разработке программ нередко возникают ситуации, когда для того, чтобы решить какую либо сложную задачу, ее предварительно разбивают на несколько более простых. Такие относительно самостоятельные части программы называются в языке Паскаль процедурами (в других языках программирования для обозначения таких частей употребляется также термин подпрограммы).

Процедуры по своему внешнему виду напоминают функции, но если при обращении к функции мы можем получить только один результат – значение данной функции, то процедура может вырабатывать на выходе несколько значений. Кроме того, процедуры используются в программах для выполнения ряда стандартных действий. Такие процедуры, подобно стандартным функциям, являются частью языка Паскаль. К таким стандартным процедурам относятся уже известные нам команды **ClrScr, TextColor** и **TextBackground**. Наряду с этими стандартными процедурами в языке Паскаль широко используются также следующие:

Delay(i)

осуществляет задержку выполнения программы на **i** миллисекунд (тысячных долей секунды)

GotoXY(a,b)

переводит курсор в точку экрана с координатами a,b

Halt

завершает выполнение программы и передает управление операционной системе

В качестве примера использования стандартных процедур Паскаля приведем программу «обратный отсчет» (см. рис. 27). Эта программа создает на экране компьютера «электронное табло», на котором в одном и том же месте последовательно выводятся числа от 10 до 1, то есть производится обратный отсчет времени, как перед стартом космического корабля, а затем выводится слово «Старт».

В данной программе цифры выводятся с помощью цикла с уменьшающимся значением счетчика (см. раздел «Оператор For»). В теле цикла используется ряд стандартных процедур. Так, перед выводом очередного числа, курсор посредством стандартной процедуры **GotoXY** каждый раз перемещается в левый верхний угол экрана для того, чтобы все числа выводились в одной и той же позиции. Далее, для того, чтобы очередное число сразу не исчезало с экрана, а оставалось на некоторое время, используется процедура **Delay**. Затем экран очищается с помощью процедуры **ClrScr**, для того, чтобы освободить место для вывода следующего числа. После завершения работы цикла с помощью тех же стандартных процедур на экран выводится слово «Старт».

Конечно, возможности языка Паскаль не ограничиваются использованием только стандартных процедур. Программист, использующий этот язык может создавать и свои собственные процедуры. Но для того, чтобы такую процедуру можно было использовать в программе, ее подобно вновь создаваемым функциям, следует предварительно описать. Описание процедуры, так же как и описание функции, должно содержаться в программе в разделе описаний после описания констант и переменных. Структура описания функции сходна со структурой основной программы. Описание включает в себя заголовок процедуры, раздел описаний и раздел операторов. К

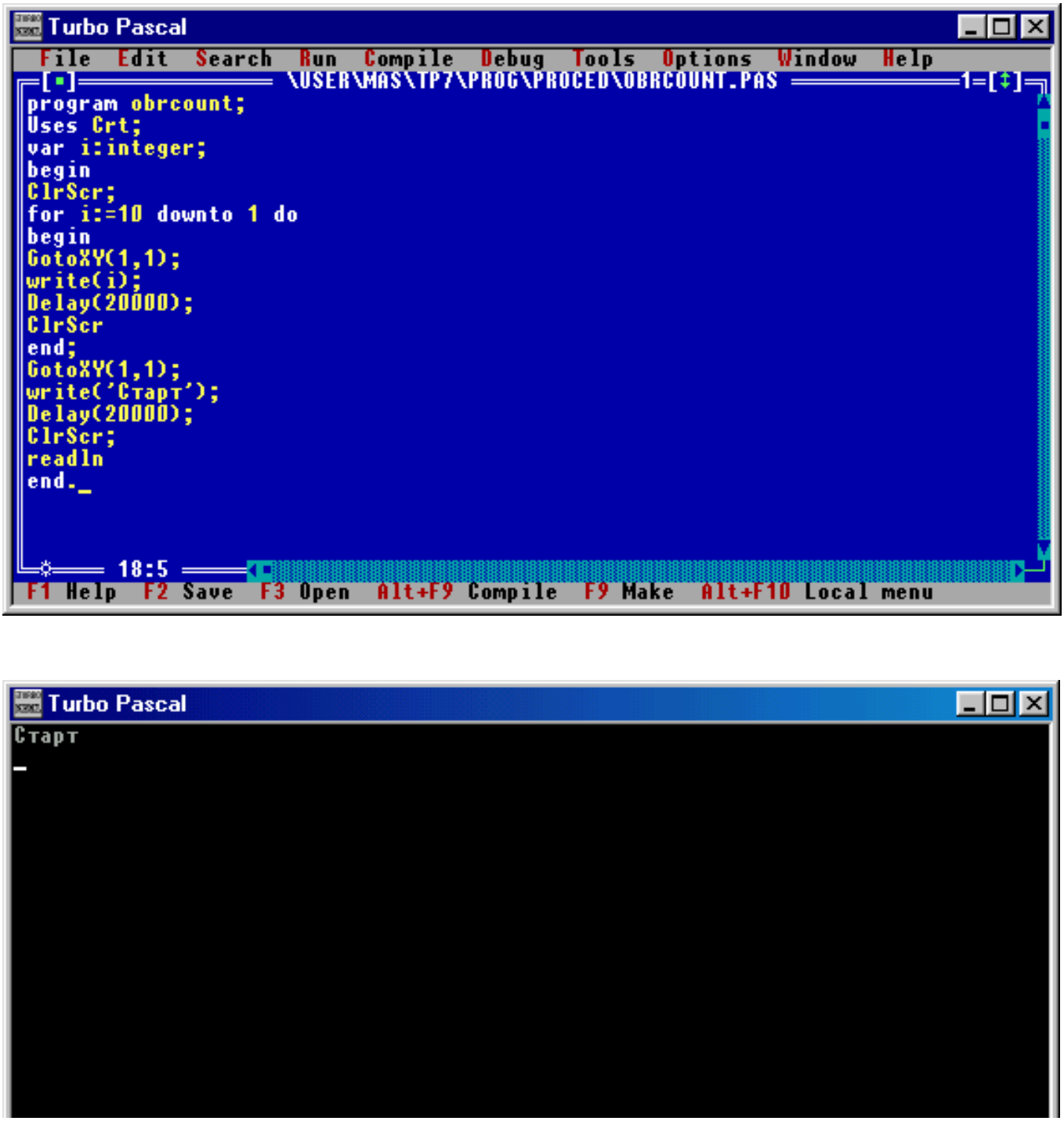


Рис. 27. Программа «обратный отсчет» (исходный текст и программа в действии)

процедуре можно обращаться из основной части программы, из другой процедуры или из функции. Такое обращение называют также ***вызовом*** процедуры.

Общий вид описания процедуры следующий:

заголовок процедуры;

раздел описаний процедуры;

begin

раздел операторов процедуры

end;

Теперь разберем более подробно составные части данного описания.

Общий вид заголовка процедуры следующий:

procedure имя_процедуры (параметры процедуры);

где **procedure** – служебное слово, имя процедуры дается по тем же правилам, что и имена переменных в Паскале, параметры перечисляются в скобках через запятую с указанием их типа. Эти параметры являются формальными. При вызове же процедуры в обращении к ней указываются ее фактические параметры. Тип каждого фактического параметра должен быть таким же, как тип соответствующего ему формального параметра. Количество формальных параметров, имеющихся в описании процедуры, и фактических параметров, используемых при обращении к ней, должно совпадать. При этом первому по счету формальному параметру в описании ставится в соответствие первый по счету фактический параметр в обращении, второму формальному – второй фактический и так далее.

Все формальные параметры делятся на два вида. Если перед именем параметра в заголовке процедуры стоит служебное слово **var**, то это – ***параметр-переменная.*** Если служебное слово **var** перед именем переменной в заголовке отсутствует, то данный параметр является ***параметром-значением.*** При обращении к процедуре формальному параметру-значению присваивается значение соответствующего ему фактического параметра, причем в качестве такого значения может выступать константа, переменная или выражение. При обращении же к процедуре, в которой имеются формальные параметры-переменные, соответствующие им фактические параметры могут быть только

переменными (не константами и не выражениями). Вызываемая процедура получает доступ к ячейкам памяти, в которых хранятся эти фактические параметры, и может изменять значения этих параметров в ходе своей работы.

Пример заголовка процедуры:

procedure vspomog(a,b:integer; var c,d:real);

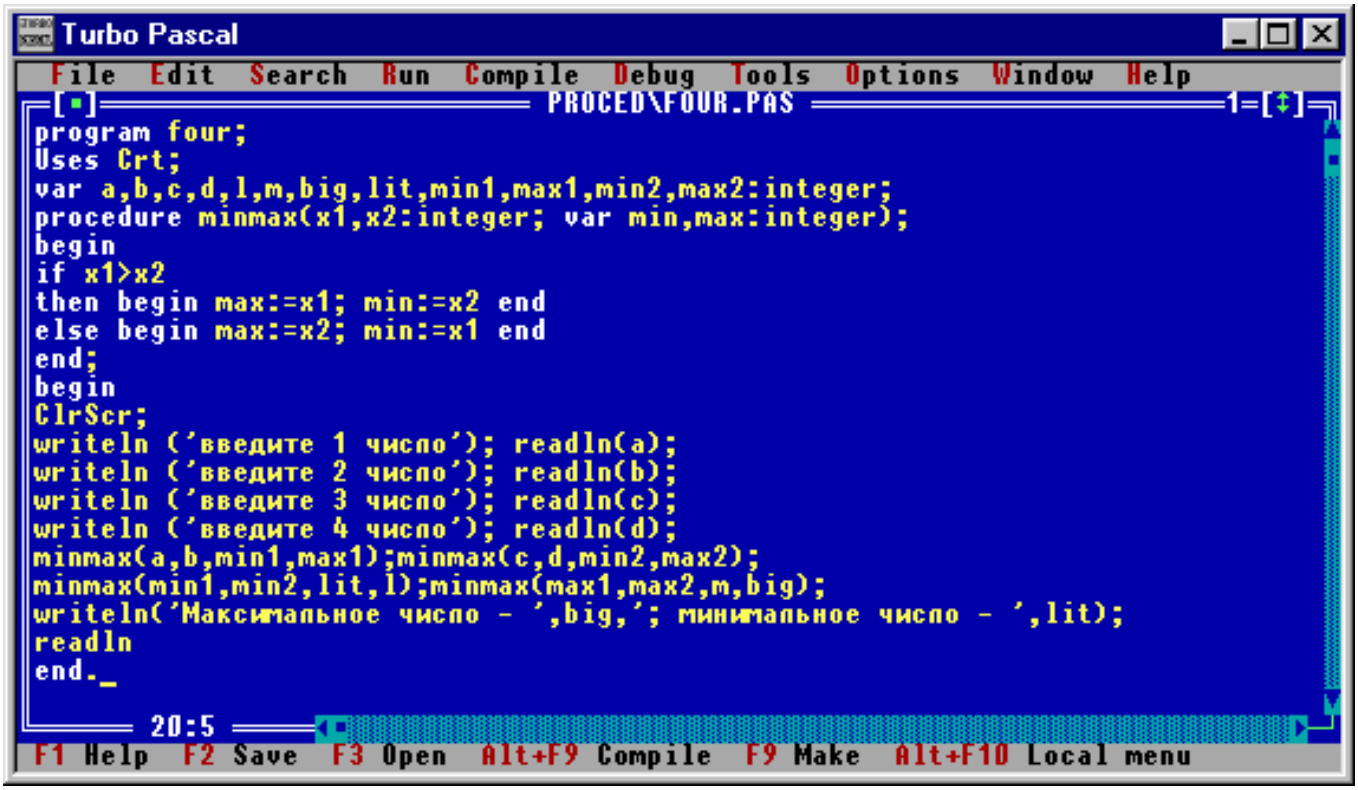
где **vspomog** – имя процедуры, **a,b,c,d** – имена формальных параметров, причем **a** и **b** являются параметрами-значениями, а **c** и **d** – параметрами-переменными.

В разделе описаний процедуры константы и переменные описываются по тем же правилам, что и в основной программе. При этом следует иметь в виду, что эти переменные могут использоваться только внутри данной процедуры. Такие переменные называются ***локальными***. В процедуре могут применяться и переменные, которые описаны в основной части программы. Такие переменные называются ***глобальными***.

Раздел операторов процедуры принципиально не отличается от такого же раздела в основной программе, он может содержать обращения к другим функциям и процедурам, но после служебного слова **end** ставится не точка, а точка с запятой, так как конец описания функции – это не конец программы.

Рассмотрим программу определения максимального и минимального числа в группе из 4 чисел (см. рис. 28). Исходные значения введем с клавиатуры. Для определения максимального и минимального из них воспользуемся следующим алгоритмом. Напишем несложную процедуру **minmax**, которая определяет максимальное и минимальное из двух чисел. Затем разобьем введенные числа на пары и в каждой паре с помощью этой процедуры определим максимум и минимум. Далее с помощью той же процедуры **minmax** определим наибольший из двух максимумов и наименьший из двух минимумов. Это и будет искомый результат.

В заголовке процедуры minmax описаны 2 формальных параметра-значения **x1** и **x2**, которые используются для ввода в процедуру исходных данных и 2 параметра-переменные **min** и **max,** используемые для вывода полученных результатов. Внутри процедуры параметры-значения сравниваются между собой и значение меньшего из них присваивается переменной **min**, а большего - переменной **max.**



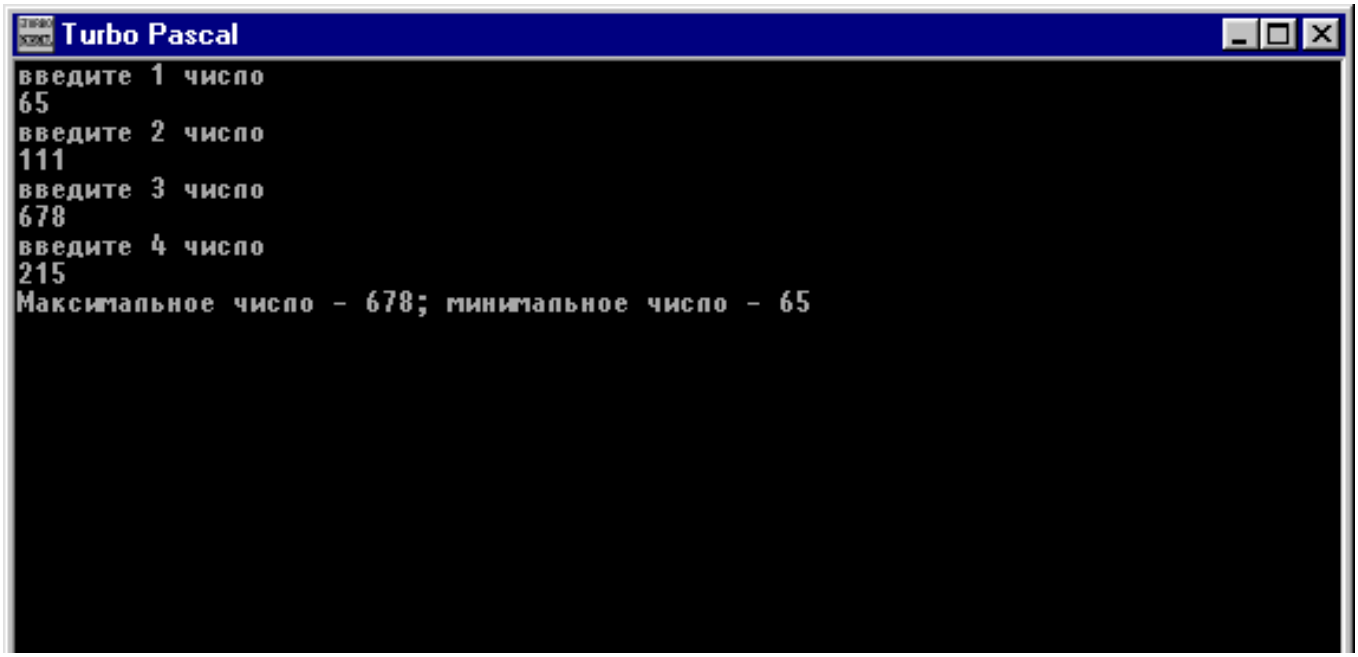


Рис 28. Программа определения минимального и максимального из 4 чисел и результаты ее работы.

В основной части программы обращение к процедуре **minmax** встречается 4 раза. В первых двух случаях в качестве фактических параметров выступают введенные с клавиатуры значения переменных **a** и **b** при первом вызове процедуры и **c** и **d** при втором. Результатами являются соответственно фактические параметры **min1** и **max1** (минимум и максимум в первой паре чисел) и **min2** и **max2** (минимум и максимум во второй паре).

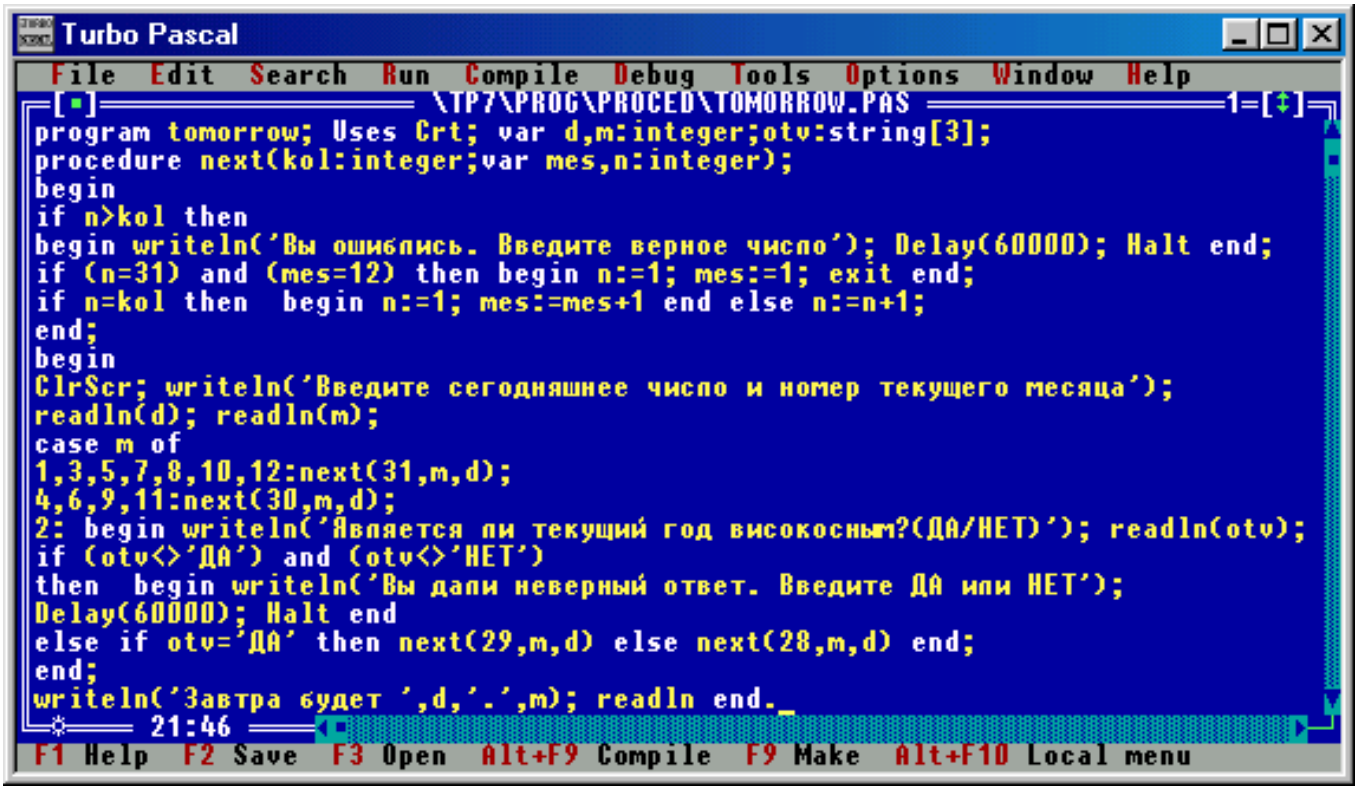
При третьем обращении к процедуре в качестве исходных данных используются 2 найденных минимума, а результатом ее работы является минимальное из этих 2 значений, передаваемое в переменную **lit.** Вспомогательная переменная **l** в принципе для решения данной задачи не нужна, но вводится, так как число фактических параметров в обращении должно соответстововать числу формальных параметров в описании. Аналогично при четвертом обращении к процедуре находится наибольший из двух максимумов **big**, а переменная **m** играет подобно переменной **l** только вспомогательную роль.

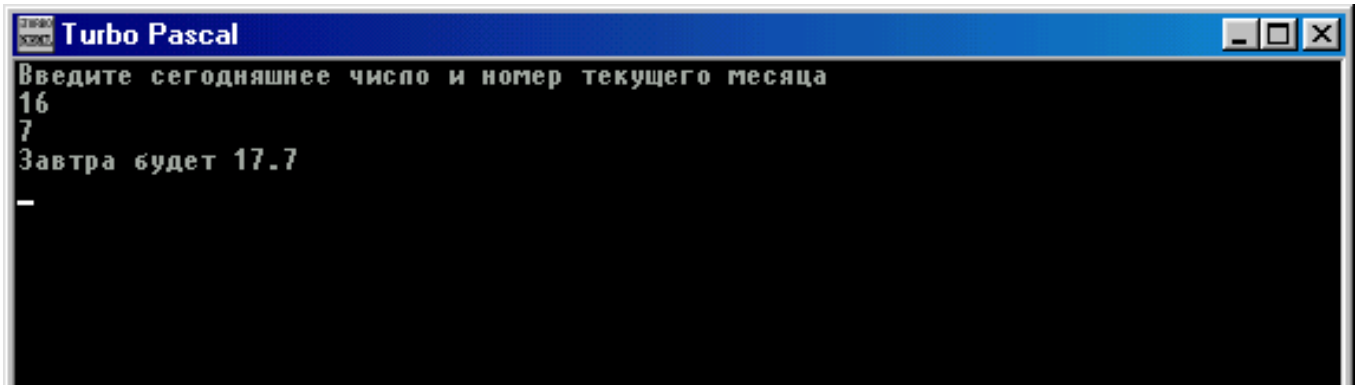
Искомые максимальное и минимальное значения выводятся на экран с помощью оператора writeln.

На рис. 28 приводятся результаты работы данной программы при **a** равном 65, **b** – 111, **c** – 678 и **d** – 215.

Выше уже говорилось о том, что в процедуру могут передаваться не только значения переменных, но и числовые константы. Приведем пример программы где реализована как раз такая возможность. Это – программа, которая по введенному пользователем текущему числу и номеру месяца определяет дату следующего дня (см. рис. 29). Данная задача не представляет сложности (нужно только прибавить единицу к текущему числу), но лишь для всех чисел месяца кроме последнего. В этом случае следующий день будет первым числом следующего месяца, а если последний день месяца – 31 декабря, то следующим днем будет первый день нового года. Необходимо также предусмотреть защиту от неверного ввода данных (чтобы пользователь случайно не ввел число, большее, чем количество дней в текущем месяце). Поэтому для упрощения решения данной задачи решим ее сперва в общем виде. Данное решение будет содержаться в созданной программистом процедуре **next.**

В заголовке процедуры **next** описываются один параметр-значение **kol** и два параметра-переменные – **mes** и **n**. При обращении к процедуре **next** параметру **kol** присваивается значение, равное количеству дней в текущем месяце. Так как это значение зависит от номера текущего месяца m, то оно определяется с помощью условного оператора case, в котором в роли





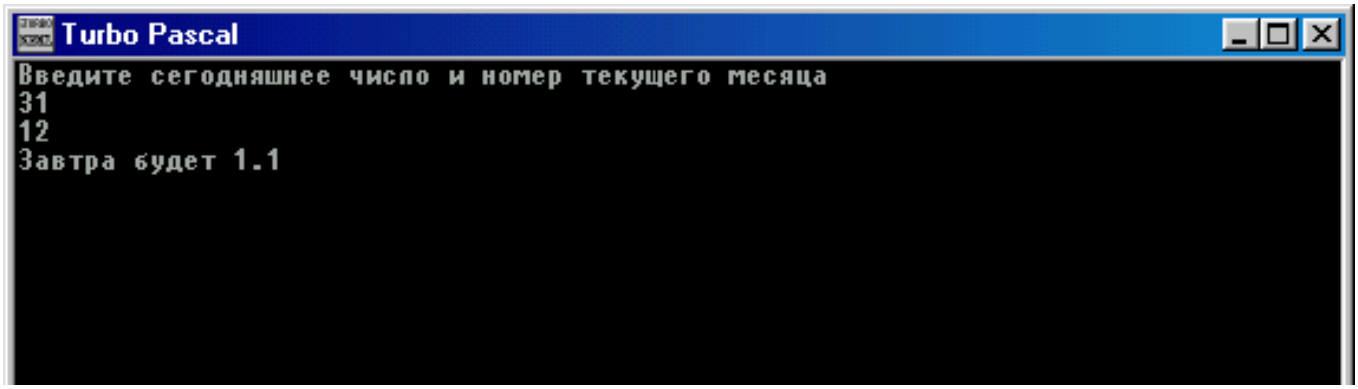


Рис 29. Программа, определяющая дату завтрашнего дня и результаты ее работы (для обычного дня в середине месяца и для 31 декабря)

переменной-селектора как раз и выступает m. Для m, равного 1,3,5,7,8,10,12 то есть для января, марта, мая, июля, августа, октября, декабря данное значение равно 31. Для m равного 4,6,9,11, то есть для апреля, июня, сентября, ноября m будет равно 30.

Сложнее решается вопрос с февралем (m=2), так как в этом случае значение, присваиваемое параметру kol, зависит от того, является ли текущий год високосным или нет. В первом случае kol будет равно 29, во втором – 28. Необходимую информацию программа запрашивает у пользователя, который на вопрос о том, является ли текущий год високосным, должен дать положительный или отрицательный ответ. Для защиты от ошибочного ответа используется сокращенный условный оператор if. Если пользователь вместо «ДА» или «НЕТ» введет что-либо другое, программа выдаст сообщение об ошибке и прекратит работу по команде **Halt**. Для того, чтобы пользователь успел прочитать сообщение об ошибке, после его вывода работа программы приостанавливается на некоторое время процедурой **Delay**.

Параметрам **mes** и **n** ставятся в соответствие фактические параметры – переменные **m** и **d,** в которых содержится порядковый номер текущего месяца и текущее число.

В начале работы данной процедуры с помощью сокращенного условного оператора **if** проверяется правильность введенных данных (то есть не больше ли переданное в процедуру текущее число **n**, чем количество дней в текущем месяце). Если при вводе была допущена ошибка, то процедура выводит сообщение об этом и прекращает работу программы используя стандартную процедуру **Halt**.

Следующий условный оператор проверяет, не является ли введенное число 31 декабря. В этом случае параметрам **mes** и **n** присваиваются значения 1 и 1 (так как следующее число будет 1 января) и затем работа процедуры **next** завершается командой **exit**, после чего происходит возвращение в основную программу.

С помощью последнего в данной процедуре оператора if определяется следующее число для всех остальных дней года. Если число не является последним в данном месяце (n<kol) , то значение параметра n увеличивается на единицу, а mes остается без изменений. В противном случае n присваивается значение 1 (следующее число – начало нового месяца) , а значение параметра mes увеличивается на единицу. На этом работа процедуры завершается, соответствующие формальным параметрам **mes** и **n** фактические параметры **m** и **d** получают новые значения, которые передаются в основную программу.

Затем на экран компьютера выводятся эти новые значения (завтрашнее число и соответствующий ему номер месяца) и на этом программа завершает свою работу.

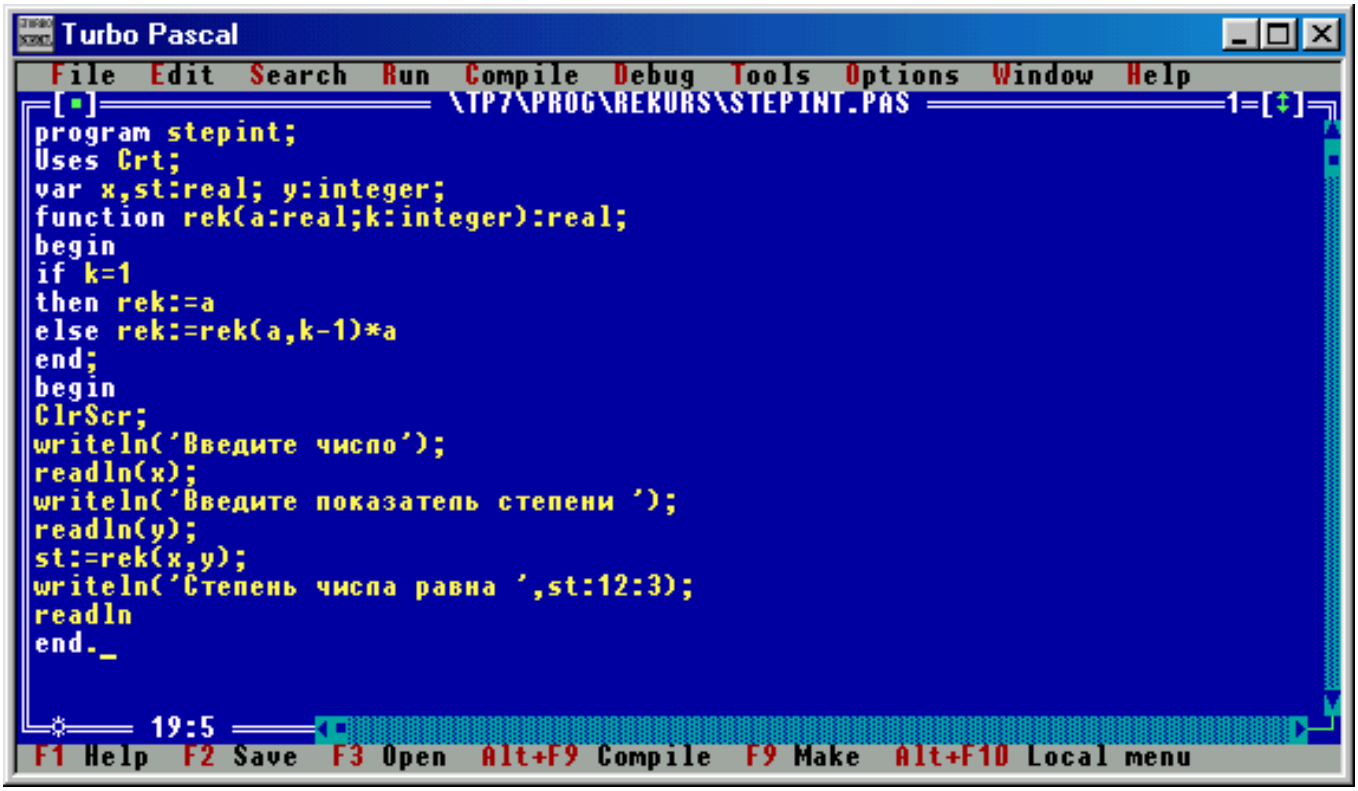
На рис. 29 приводятся результаты работы данной программы для различных исходных данных (для обычного дня в середине месяца и для 31 декабря).

Рекурсия.

В предыдущем разделе данного пособия мы уже рассматривали программу, в которой имеется процедура, содержащая обращения к другим процедурам. Обращения к другим функциям и процедурам могут содержать и функции. При этом возможен и такой вариант, когда какая-либо функция или процедура содержит обращение к самой себе. Такая функция или процедура называется рекуррентной (от английского слова «recurrence», что в переводе означает возвращение или повторение), а процесс вызова функции или процедуры из нее самой – рекурсией.

Процесс рекурсии происходит следующим образом: в начале происходит вызов некоторой функции(процедуры). Затем, еще до завершения своей работы, эта же функция(процедура) вызывает саму себя, и в результате в памяти компьютера появляется второй экземпляр той же функции(процедуры), второй экземпляр в ходе работы создает третий, и так далее. Естественно, что такой процесс может продолжаться до бесконечности, поэтому для решения какой-либо конкретной задачи в рекурсивной функции(процедуре) обязательно должен содержаться условный оператор, одна из ветвей которого обеспечивает прерывание этой бесконечной цепочки и завершение работы всех экземпляров функции (процедуры). Внутри этого оператора и должен находиться вызов функции.

В качестве несложного примера, наглядно иллюстрирующего процесс рекурсии, рассмотрим задачу вычисления целой положительной степени вещественного числа ($\mathbf{x}^{\mathbf{y}}$) (см. рис. 30). В данной программе в основной ее части производится только ввод исходных данных (самого числа **x** и показателя степени **y**), вызов рекурсивной функции **rek** и вывод полученного



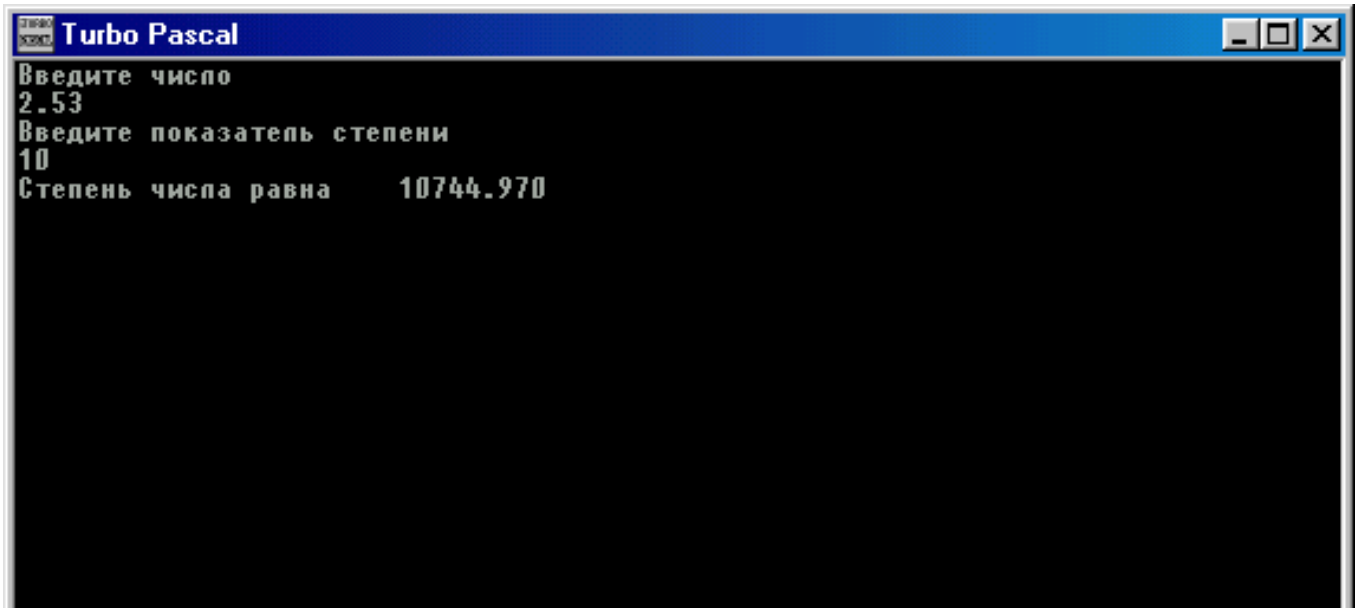


Рис. 30. Программа вычисления целой положительной степени вещественного числа и результат ее работы.

результата, а само вычисление степени производится в функции **rek.** Разберем более подробно механизм действия данной функции.

Для этого нужно ответить на вопрос : чему равно любое число в первой степени. Ответ очевиден – самому этому числу. Это мы и запишем в начале условного оператора: для **k=1** результат **rek** равен исходному числу **a.** Для нахождения же квадрата числа нужно это число умножить само на себя, для нахождения куба – квадрат числа умножить на это число и так далее. Таким образом, существует следующая закономерность **k** степень числа равна этому числу в степени **k-1**, умноженному на само число:

$a^k=a^{(k-1)}*a$

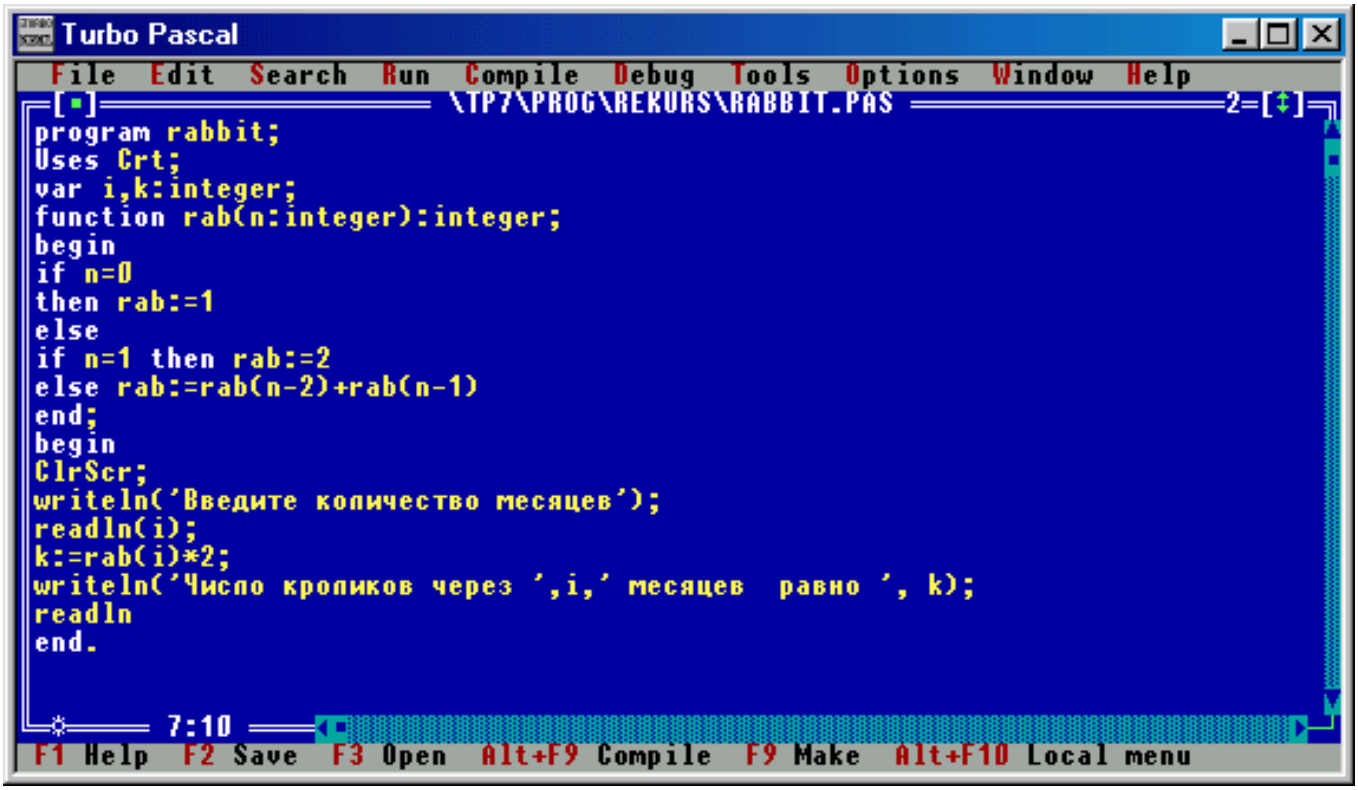
Эту формулу запишем в той ветви условного оператора, которая находится после служебного слова **else.**

Теперь рассмотрим непосредственно работу функции. Данная функция имеет два формальных параметра **a** и **k**, которым соответствуют в основной части программы фактические параметры **x** и **y.** Значением функции является a^k . Для вычисления значения функции **rek** с параметрами **k** и **a** вызывается другой экземпляр той же функции с параметрами **k-1** и **a** (**a** остается неизменным для всех экземпляров функции), затем этот экземпляр вызывает следующий с параметром **k-2** и так далее, пока процесс не доходит до обращения к экземпляру для **k=1**. Значение функции для **k=1** уже известно, оно передается в экземпляр для **k=2** и так далее, пока не определяется значение для **k**. Это последнее значение передается в основную часть программы для вывода на экран компьютера.

На рис. 30 приведен результат работы программы для x=2,53 и y=10.

В заключение данной главы приведем еще один пример использования рекурсии для решения классической математической задачи известной как «числа Фибоначчи». Эта задача была решена в XIII веке выдающимся итальянским математиком Леонардо Фибоначчи. Вот ее условия: пара кроликов каждый месяц дает потомство - двух кроликов, которые через два месяца сами способны давать новое потомство. Сколько кроликов будет через год, если в начале года была одна пара кроликов.

Для решения данной задачи используем рекурсивную функцию **rab(n)**, которая определяет количество пар кроликов через n месяцев после начала года. В начале года существовала только одна пара кроликов, то есть **rab(0)=1**, через месяц их стало две, следовательно **rab(1)=2**. Через 2 месяца количество пар кроликов увеличивается еще на 1, то есть на столько, сколько их было в



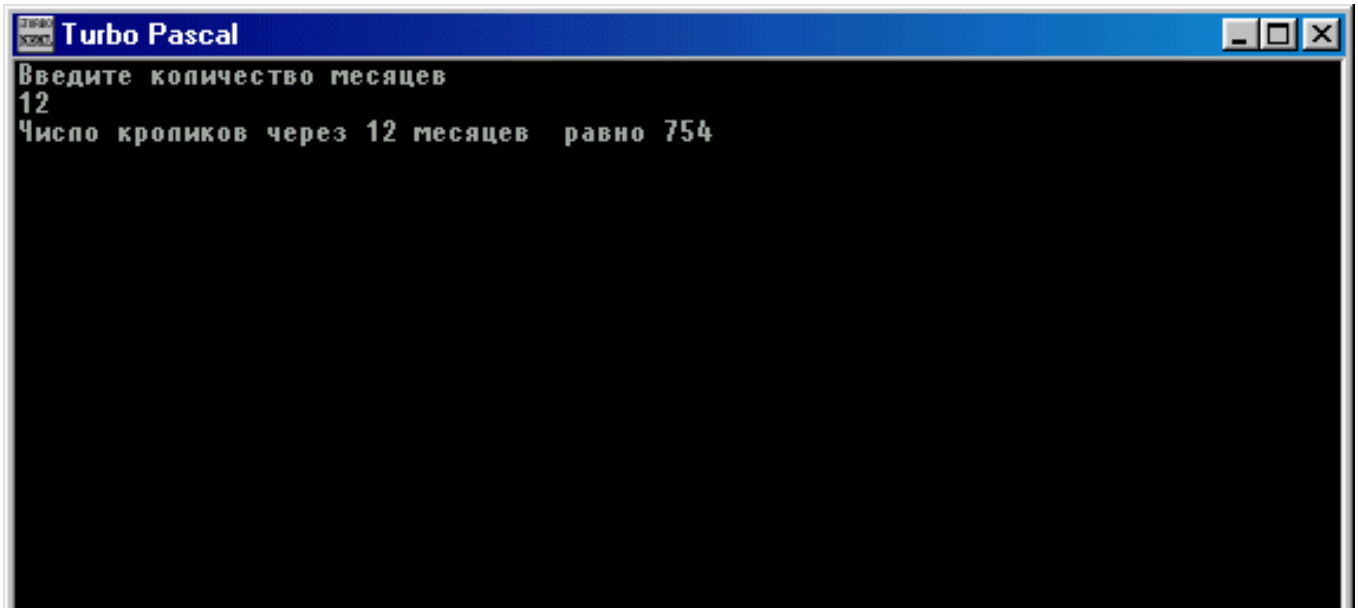


Рис. 31 Программа «числа Фибоначчи» и результаты ее работы.

начале года. Таким образом **rab(2)=3** или **rab(2)=rab(1)+rab(0)**. Продолжая наши рассуждения, получаем следующую зависимость:

rab(n)=rab(n-1)+rab(n-2)

Эту рекурсивную функцию используем в программе rabbit, которая подсчитывает и выводит на экран количество кроликов через n месяцев.

Количество кроликов через n месяцев будет равно rab(n)*2. Ниже приведены результаты работы программы при n=12, то есть искомое количество кроликов через год.