Исмаил Садыгов
Рамин Махмудзаде
Наида Исаева

ИНФОРМАТИКА

9

Учебник для 9-го класса общеобразовательной школы

Утверждено приказом Министерства образования Азербайджанской Республики №712 от 09.06.2008

BAKI NƏŞR

Bakı — 2010

Утверждено
Министерством образования
Азербайджанской Республики

Научный редактор: Расим Алигулиев, член-корреспондент НАНА, д.т.н., профессор
Рецензенты: Алекпер Алиев, д.т.н., профессор
Хаят Ахундова, педагог школы №164 г.Баку
Валид Магеррамов, педагог лицея с физико-математическим и информатическим уклоном г.Баку
Самиха Рустамова, педагог школы №258 г.Баку
Гюльнара Салимова, педагог школы №7 г.Баку
Перевод: Наида Исаева

Информатика — 9. Учебник для 9-го класса общеобразовательной школы. И.Дж.Садыгов, Р.А.Махмудзаде, Н.Р.Исаева. Баку, “Bakınəşr”, 2010, 128 стр.
ISBN-978-9952-430-08-8

1

Язык программирования PASCAL

1.1. КЛАССИФИКАЦИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

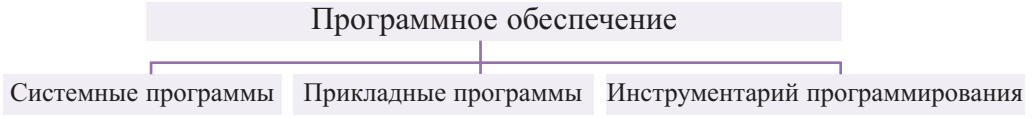
Для выполнения компьютером любой работы, наряду с аппаратным обеспечением, необходимы соответствующие *программы*. Именно программы определяют реакции компьютера на нажатие клавиш на клавиатуре, движения мыши, прием информации с другого компьютера. Посредством программ, имеющихся на компьютере, осуществляется вывод информации на экран, печать документа на принтере, прослушивание музыки.

Программное обеспечение компьютера (“software”) – неотъемлемая часть компьютерной системы, и в то же время является логическим продолжением его технических средств.

Сфера применения того или иного компьютера определяется созданным для него программным обеспечением. Сам по себе компьютер не обладает знаниями ни в одной области применения – все эти знания сконцентрированы в выполняемых на компьютерах программах.

Программное обеспечение современных компьютеров включает миллионы программ – от игровых до научных.

Все компьютерные программы можно условно разделить на три категории: *системные программы, прикладные программы и инструментарий программирования.*



Системные программы. Системные программы предназначены для управления ресурсами компьютера – центральным процессором, памятью, устройствами ввода-вывода – и предусмотрены для всех пользователей без

исключения. Системные программы обеспечивают эффективную работу прикладных программ.

Среди системных программ особое место занимают *операционные системы* – они составляют основу системного программного обеспечения. Это один из важнейших элементов персонального компьютера. Операционная система – это система программ, обеспечивающих запуск компьютера, слаженную работу всех его частей и управление информацией.

В функции операционной системы входят:

- осуществление диалога с пользователем;
- запуск устройств оперативной и постоянной памяти;
- управление компьютером;
- запуск программ на выполнение.

Когда-то в компьютерах типа IBM PC использовалась по большей части операционная система MS-DOS, выпущенная фирмой Microsoft. При работе с этой системой пользователь мог решать только какую-то одну конкретную задачу.

В наши дни в персональных компьютерах используются *многозадачные* операционные системы: микропроцессор распределяет ресурсы компьютера одновременно между несколькими программами и задачами, имеющимися в памяти компьютера. К таким операционным системам относятся OS/2, MacOS, UNIX, Linux, Windows XP, Windows Vista и др.


Другую важную часть системных программ представляют *утилиты* (лат. “utilitas” – польза, выгода). Они дополняют операционную систему и повышают ее возможности. В частности, они самостоятельно решают некоторые важные задачи. Вот некоторые виды утилит:

- интерфейсные программы;
- антивирусные программы;
- программы-архиваторы;
- программы-оболочки;
- программы, тестирующие работоспособность компьютерных устройств;
- программы, управляющие работой устройств (драйверы) и др.

Прикладные программы. Прикладная программа – это программа, способствующая решению какой-либо задачи в пределах данной проблемной области.

В наши дни существуют сотни тысяч прикладных программ для персональных компьютеров. Вот наиболее популярные из них:

- текстовые редакторы (процессоры);
- программы для обработки табличных данных (электронные таблицы);
- издательские системы;
- системы управления базами данных;
- программы для подготовки презентаций;
- графические редакторы;
- программы статистического анализа данных;
- компьютерные игры, обучающие программы и т.д.

Инструментарий программирования. Программы, относящие к этому классу, предназначены для создания системного и прикладного программного обеспечения. Для создания программного обеспечения используются языки программирования Basic, C++, Pascal, Java и другие. На базе этих языков разработаны системы создания программного обеспечения: Visual Basic, Visual C++, Delphi. Для обучения детей основам программирования во многих учебных заведениях мира используется язык LOGO.

1. **Назовите основные категории программного обеспечения.**
2. **Каково предназначение системных программ?**
3. **Что такое операционная система?**
4. **Для чего применяются прикладные программы?**
5. **Что подразумевается под инструментариями программирования?**

1.2. ЯЗЫКИ ПРОГРАММИРОВАНИЯ

Чтобы объяснить что-то человеку, язык которого ты не знаешь, применяют либо язык жестов, либо слова из известного данному человеку языка. У центрального устройства компьютера – процессора тоже есть собственный язык.

Машинный язык. Для непосредственного "общения" с компьютером используется машинный язык ("machine language"), представляющий собою набор цифр. Именно цифрами кодируются и выражаются все команды для процессора. Скажем, 1 обозначает сложение, 2 – умножение, 3 – деление и т.д. (или, соответственно, 01, 02, 03). Для выполнения той или иной операции процессору необходимы не только команды, но и данные. Вот упрощенный общий вид некой абстрактной, примерной команды для процессора:

код команды	первый операнд	второй операнд	номер ячейки, в которую будет помещен результат

Аргументы команд программирования, то есть данные, называются *операндами*.

Каждое данное размещается в ячейках памяти компьютера. Вся память компьютера поделена на ячейки, у каждой из которых имеется свой номер – *адрес*. Таким образом, каждый операнд характеризуется двумя параметрами – значением и местом в памяти.

Запись "00xx" обозначает само число xx, а запись "01xx" – значение ячейки xx в памяти. То есть, если перед данными "xx" стоит "00", то это сами данные, а если "01", то это их местоположение.

Тогда программу для вычисления среднего арифметического чисел, находящихся в ячейках 01 и 02, можно представить следующим образом:

```
01   0101 0102 0103
03   0103 0002 0103
```

В переводе на обычный язык верхняя строчка выглядит так: "Сложить данные, находящиеся в ячейках 01 и 02, и записать результат в ячейку 03". Нижняя же строка будет выглядеть так: "Разделить данные, находящиеся в ячейке 03, на 2 и записать результат в ячейку 03".

> При записи данных в ячейку памяти предыдущая информация в ней стирается.

Запись этой программы в одну строку выглядит так:

```
010101010201030301030002 0103
```

Как видно, даже столь простая программа в записи на машинном языке представляет собой "тайну за семью печатями". В реальных компьютерах же программы в машинных кодах намного сложнее.

Под “машинным кодом” подразумевают *программу, написанную на машинном языке*.

Составить программу с помощью такого кода и проверить ее очень сложно. Для этого нужно либо помнить коды и форматы всех команд, либо каждый раз сверяться со специальными таблицами.

Малейшая небрежность при кодировании, ошибка в записи, путаница в цифрах могут привести к самому неожиданному результату. Найти ошибку при этом нелегко, ибо программист имеет дело не с алгоритмом, написанным на более-менее понятном языке, а с набором цифр.

Программисты на протяжении нескольких лет искали пути облегчения своего труда. Необходимо было разработать такой инструмент, с помощью которого можно было бы писать программы, не расходуя время на мелкие детали, и чтобы программисты могли бы заниматься исключительно творческой работой, то есть создавать алгоритмы, а прочую работу оставить компьютеру.

“Многозначность толкования”

Как было бы хорошо, если бы команды компьютеру можно было бы давать на обычном русском, английском, французском, турецком или любом другом языке! Но, к сожалению, пока компьютеры не понимают нюансы человеческой речи. Ведь люди дополняют свою речь жестами и мимикой, применяют метафоры, иносказания, иронию и прочие приемы обогащения речи – это дает возможность говорить одно, имея в виду совсем другое. Человек часто использует слова и выражения, имеющие несколько значений, уточняя конкретное значение того или иного слова в зависимости от контекста, интонации и прочих обстоятельств. Даже в письменной речи применяются намеки, помогающие правильно воспринять смысл написанного.

Человеческий разум способен разгадывать загадки естественного языка, тогда как компьютеру доступна только математически выверенная, строгая система общения, в которой каждый символ или группа символов имеют только одно значение, каждое предложение досконально понятно, а любого рода намеки и двусмысленности недопустимы.

Assembler. Программирование посредством машинных кодов – занятие весьма кропотливое. Простейшие вычислительные операции (например, загрузка числа из памяти в процессор, сложение его с другим числом, запись полученного результата обратно в память) приобретают при этом такую сложную форму, что данный процесс можно сравнить с возведением гигантского здания из маленьких кирпичиков.

Название языка Assembler происходит от англ. *“assemble”* - “складывать, собирать”.

Первым шагом к решению этой проблемы стала замена команд символами, то есть команды стали представлять обычными сокращенными

словами. Воплощение в жизнь этой простой идеи улучшило восприятие программистами создаваемых программ, уменьшило число ошибок, облегчило труд программистов.

Запись команд при помощи символов называется *мнемоническим письмом*. Рассмотрим пример программы, записанной при помощи данной системы.

```
СУМ    X1, X2  > X3
ДЕЛ    X3, 2   > X3
```

Первая строка этого фрагмента означает то, что необходимо проСУМмировать содержимое ячеек 1 и 2, и полученный результат поместить в ячейку 3. Вторая строка указывает на то, что необходимо разДЕЛить содержимое ячейки 3 памяти компьютера на 2 и полученный результат поместить в ту же ячейку 3. Как видно, такая запись более понятна, чем приведенная выше запись тех же действий при помощи машинного кода.

Это стало первым шагом к созданию языка программирования, понятного как человеку, так и компьютеру.

Вторым шагом стало создание библиотеки процедур, то есть возможность использования кодов многократно – до того каждому программисту приходилось каждый раз заново “открывать для себя Америку”. Библиотека процедур позволила программистам составлять новые сложные программы качественно и быстро.

Эти два направления стали залогом создания и развития новых языков программирования.

“mnemonikon” с греческого означает способность “запоминать”.

Грейс Муррей Хоппер
(1906 – 1992)
В 1952 году в США Грейс Хоппер (Grace Murray Hopper) изобрела первый в мире мнемонический язык программирования – язык Assembler (assembly language). Он включал в себя систему мнемонических команд, библиотеку процедур и специальную программу для преобразования текстов команд в машинный код. Процедура перевода программы на машинный язык называется *компиляцией*, а выполняющая ее программа – *компилятором* (этот термин также принадлежит Грейс М.Хоппер).

Assembler избавляет программиста от самой трудоемкой части его работы – перевода мнемонических команд в машинный код вручную. Тем не менее, у языка Assembler есть два крупных недостатка. Об одном из них вы, наверное, уже догадались: он состоит в том, что работа на этом языке требует от программиста огромной внимательности и терпения, ведь для непосредственного управления процессором предусмотрено великое множество мелких операций.

Второй недостаток заключается в том, что программы, написанные на языке Assembler, не являются “переносимыми” (portable). Например, программу на языке Assembler, написанную для процессора Intel 8080, невозможно использовать на компьютере с процессором Motorolla 6800 – для этого процессора ее придется переделать. Правда, это не такая уж сложная задача, но все же необходимо проделать определенную работу.

1. Что такое машинный язык?
2. Что такое Assembler и в чем состоит его преимущество перед машинным языком?
3. Каковы недостатки языка Assembler?
4. Что подразумевается под “переносимыми” программами?

1.3. ЯЗЫКИ ВЫСОКОГО УРОВНЯ

Язык Assembler, хотя и был мнемоническим, но все же не удовлетворял нуждам ученых, которые были основными пользователями компьютеров на заре компьютерной эры. Причина состояла в том, что, как уже указывалось выше, этот язык был труден для изучения, так как по сути своей он близок к машинному коду. Кроме того, у каждого процессора был “свой Assembler”, в результате чего пользователю, работающему на нескольких машинах, приходилось порой знать “несколько разных языков Assembler”. В довершение всего, при программировании на этом языке приходилось представлять себе весь процесс, вплоть до мельчайших деталей, указывая сложные формулы в виде последовательности различных операций. Однако программист оперирует в уме общими категориями – “вычислить формулу”, “вывести число на экран”, “повторить операцию 10 раз”, и т.д.

Все прочие языки программирования, кроме языка Assembler, именуют *языками высокого уровня*. Но это не говорит о том, что они все находятся на одном и том же уровне – уровень одного из них может быть выше или ниже, чем у другого. Понятие “язык высокого уровня” подразумевает лишь степень близости данного языка к человеческому языку.

Представим такую картину. Вы набираете на клавиатуре фразу “Подсчитать прибыли и убытки за текущий год, подготовить годовой отчет, снять с него несколько копий и разослать их по нужным адресам” – и компьютер тут же исполняет все эти ваши указания! Увы, современные языки программирования пока далеки от подобного идеала…

Для решения этой проблемы были необходимы новые языки программирования. При создании этих языков, в отличие от языка Assembler, основной целью было добиться не того, чтобы они были понятны машине, а того, чтобы они были удобны для человека, работающего с машиной.

Начали создаваться новые языки, более понятные человеку (программисту) и облегчающие процесс программирования. Каждый создатель языка программирования воплощал в своем творении собственные представления о диалоге между человеком и машиной, и потому в течение достаточно короткого периода появились сотни новых языков (так, за период с 1950 по 1993 год их возникло более тысячи). Естественно, лишь небольшая часть этих языков *высокого уровня* (high-level) получила дальнейшее развитие и распространение. В отличие от них всех, Assembler считается языком *низкого уровня* (low-level) , так как он более близок к машинному языку и работает с компьютерными устройствами.

У языков высокого уровня имеются свои достоинства и недостатки. Первейшее преимущество языков высокого уровня перед языком Assembler – то, что их очень легко изучать и применять. Программы, написанные на этих языках, более компактны и легки для понимания, чем программы на языке Assembler, они по большей части “переносимые”, то есть одинаково работают на компьютерах с разными процессорами. А это значит, что для их написания не нужно знать нюансы архитектуры компьютера, на котором они будут использоваться. Естественно, что в таком случае у каждого процессора должен быть свой компилятор. Исполняемые файлы, создаваемые этим компилятором, пригодны только для данного процессора.

Какой язык программирования лучше?

У каждого языка программирования есть свои приверженцы и противники.

В наши дни существуют несколько тысяч языков программирования. Как среди этого количества языков выбрать наиболее удобный?

Инженеры, банкиры, военные решают множество разнообразных задач, для чего выбирают те или иные языки программирования. Инженеры отдают предпочтение языку FORTRAN, банкиры обычно используют COBOL, военные же пишут программы планирования и управления войсками на языке ADA. Ученым, работающие в сфере искусственного интеллекта, больше всего подходят языки PROLOG и LISP. Программисты, пишущие программы для Интернета, обычно предпочитают язык JAVA.

Все перечисленные языки программирования – специальные. В каждом из них имеются такие операторы, при помощи которых наиболее удобно решать те или иные задачи. Наряду со специальными, существуют также универсальные языки. С их помощью можно решать практически любые задачи. Из таких языков наиболее популярны три:

- Basic
- Pascal
- C++

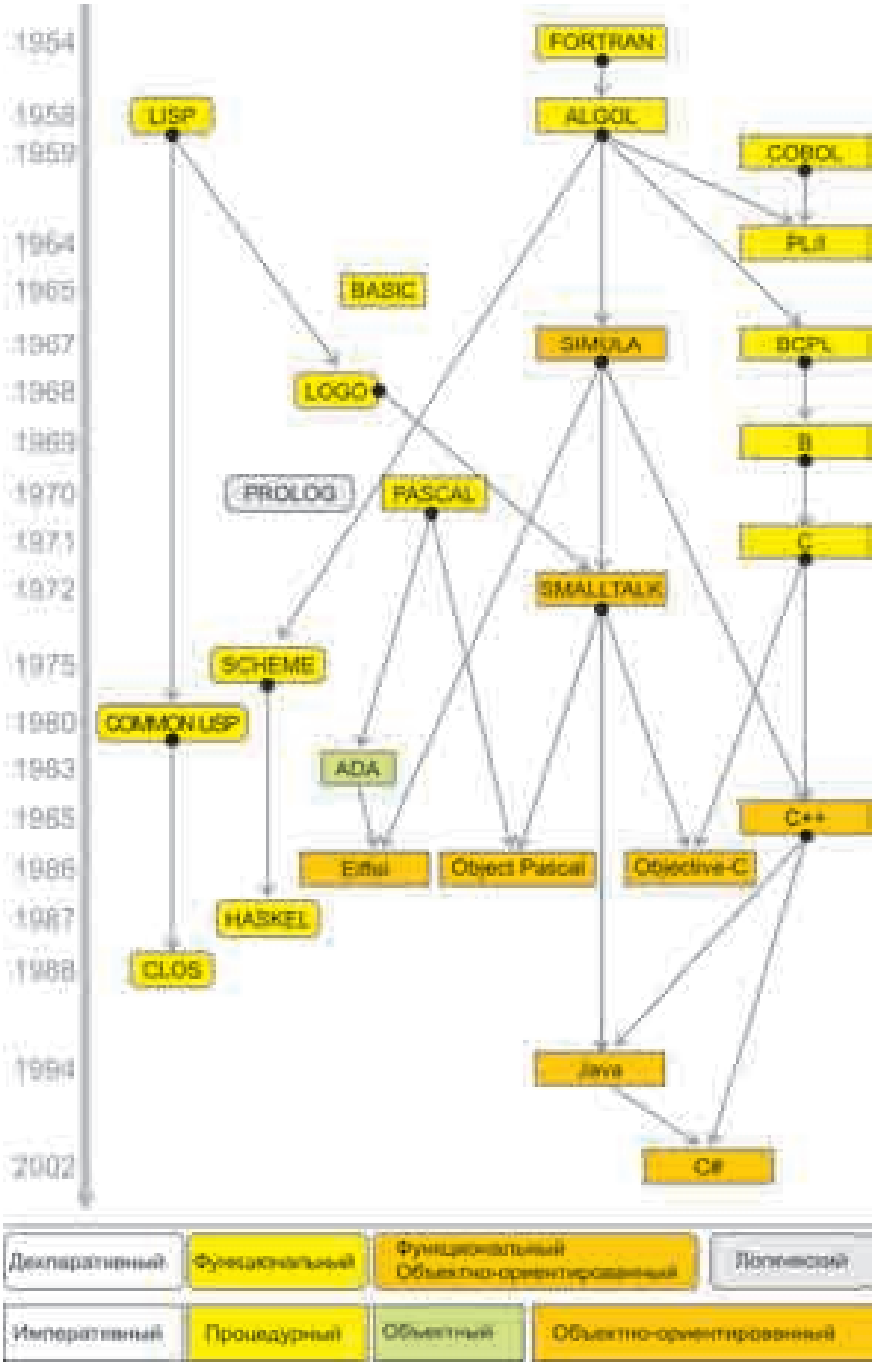


Схема эволюции языков высокого уровня

С другой стороны, программы, написанные на языке Assembler, на практике оказываются более продуктивными, чем написанные на языках высокого уровня. Исполняемые файлы, создаваемые компиляторами языков высокого уровня, занимают больше места и медленнее работают, чем аналогичные по свойствам программы, написанные на языке Assembler. Правда, в последнее время в результате развития микропроцессоров компиляторы генерируют более оптимальные коды.

При помощи языков высокого уровня можно программировать в любой области. Однако существуют и такие языки, которые предназначены для использования в конкретных областях:

- ALGOL – для математических задач
- CHILL – для систем телекоммуникаций
- COBOL – для экономических задач
- FOPTRAN – для математических расчетов
- Java – для работы с объектами
- Linda – для параллельной обработки данных
- PostScript – для изображения макетов
- PROLOG – для решения задач в области искусственного интеллекта

1. Что означают понятия “языки программирования высокого уровня” и “языки программирования низкого уровня”?
2. В чем преимущество программирования на языках высокого уровня перед программированием на машинном языке?
3. Разъясните преимущества и недостатки языков высокого уровня.
4. Как вы себе представляете язык программирования самого высокого уровня?

1.4. РАЗРАБОТКА ПРОГРАММ

Теория программирования развивалась параллельно с практическим программированием. На первом этапе развития была разработана математическая теория обработки информации и в тот же период разработаны средства проверки правильности программ и принципы создания эффективных трансляторов.

В то время на программистов смотрели как на высокоспециализированных работников. Это была одна из редких профессий, и создание программ тогда еще не носило массовый характер.

Развитие и распространение компьютеров привело к переносу центра тяжести в прикладную сферу. Число программистов и созданных ими программ начало исчисляться миллионами. Наряду с этим упал уровень знаний, определяющий специализированных программистов. Постепенно в создании сложных программ все больше стали принимать участие программисты среднего уровня.

Разработка небольших и средних по размеру программ. Небольшую и среднюю по размеру программу (состоящую из нескольких тысяч строк) могут разрабатывать независимо друг от друга несколько программистов. Обычно этот процесс состоит из двух этапов:

Первый этап – стадия *анализа*. На этой стадии определяется назначение программы, разрабатывается алгоритм решения данной задачи, определяются структура данных, объекты программы и выясняются связи между ними.

Второй этап – этап *кодирования*. На этой стадии алгоритм пишется на конкретном языке программирования. Для небольших программ – это основной этап, требующий максимального труда. Тестирование и отладка программ не требуют большого труда. На выполнение этих этапов одним программистом может уйти около полугода.

Разработка больших программ. Большие программы могут содержать несколько миллионов строк. Над их созданием трудятся совместно десятки, а порой сотни программистов в течение нескольких лет.

Этапы создания программ:

1. Постановка и анализ задачи.
2. Разработка технического задания.
3. Проектирование и кодирование.
4. Тестирование и отладка.
5. Внедрение.
6. Сопровождение программы.

На первом этапе проводится *анализ требований*, *предъявляемых к проекту*. Эта одна из важных стадий для успешного выполнения проекта. Безуспешность некоторых больших проектов связана именно с ошибками, допущенными именно на этой стадии. В процессе анализа требований уточняется назначение программы, определяются входные и выходные данные. Проводится оценка необходимых ресурсов и стоимости проекта.

На следующем этапе составляются *спецификации программы*. На этой стадии формируются технические задания для программистов, подготавливаются рабочие документы и строится календарный план работ.

Затем начинаются работы по *проектированию* и *кодированию* программы. В больших проектах это не основной этап разработки программы.

Интерпретаторы. Компиляторы

Для того, чтобы программу, написанную на языке высокого уровня, мог понять и выполнить центральный процессор компьютера, она должна быть переведена на машинный язык. Это преобразование может быть произведено различными способами.

Первый способ заключается в запуске программы, которая *переводит* (транслирует, от англ. *translate*) каждую строку программы на машинный язык. Эта программа переводит одну строку программы на машинный язык, передает ее центральному процессору и только после этого начинает переводить следующую строку. Такая программа называется *интерпретатор* (англ. *interpreter*).

Эту программу можно сравнить с переводчиком, который помогает общаться людям, говорящих на разных языках. Один человек что-то говорит, переводчик переводит сказанное. Второй собеседник отвечает, и его слова переводчик переводит первому собеседнику. Этот процесс продолжается в течение всего разговора.

Преимущество такого подхода – простота восприятия пользователем. Сразу после написания и запуска программы можно видеть, что делает компьютер на каждом этапе. Если нужно что-то поменять в программе, то производятся изменения, и программа заново транслируется. Но существует недостаток такого подхода: даже после того как программа уже готова, прежде чем ее выполнить, транслятор переводит каждую строку программы в машинный код и в результате общее время выполнения программы увеличивается.

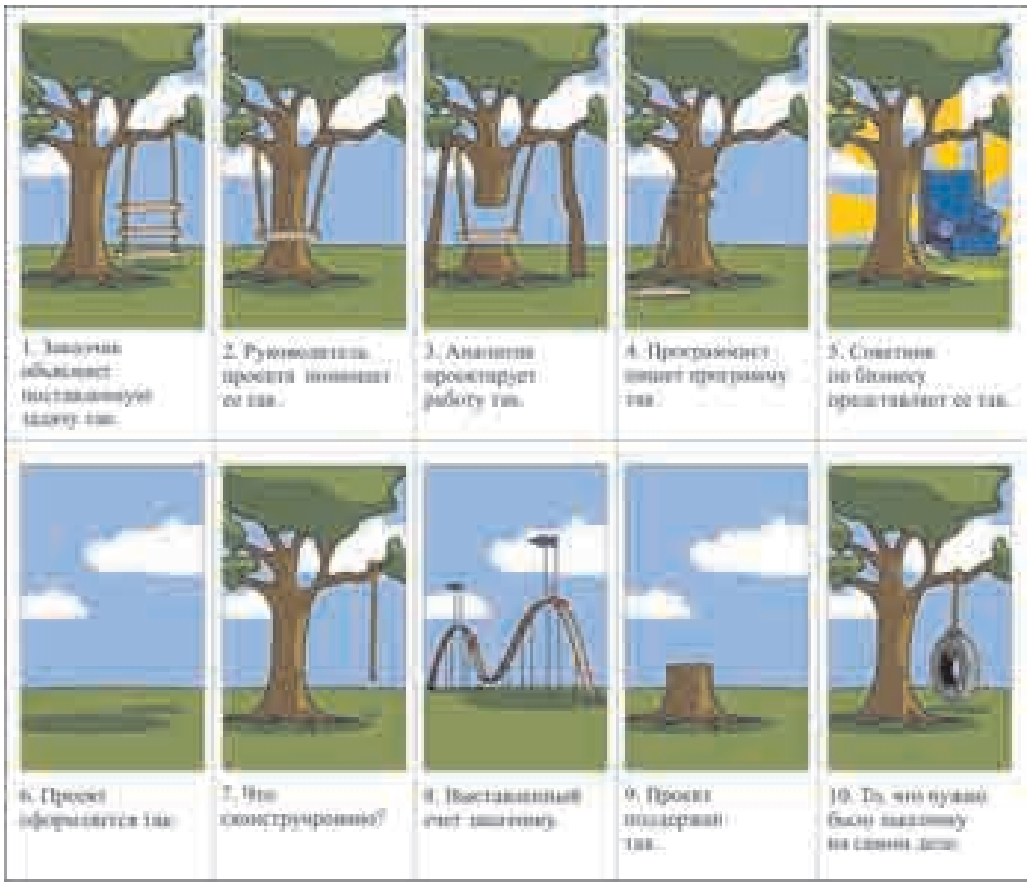
Проведем другое сравнение. Для того чтобы перевести этот учебник на английский язык, издательство заключает соглашение с переводчиком, и тот переводит книгу на английский язык целиком. Другой вид транслятора – *компилятор* (англ. *compile* – компоновка, составление) работает именно таким образом: он читает программу, написанную на языке высокого уровня, переводит ее полностью в машинный код и сохраняет в отдельном файле. Независимо от того, на каком языке был написан исходный код, сохраненный файл можно выполнять многократно. Понятно, что нет необходимости каждый раз транслировать программу.

По окончании кодирования (в некоторых случаях даже заранее) проводятся *тестирование* и *отладка* программы. В процессе тестирования проверяется правильность программы, ее производительность, надежность работы в критических режимах, работа при некорректных входных данных и

устойчивость при технических неполадках. Выявленные в процессе тестирования ошибки устраняются программистами.

Если программа написана с учетом требований конкретного заказчика, то этап *внедрения* обязателен. На этом этапе настраивается оборудование, данные, использованные в других программах, переносятся в программу, персонал, который будет работать с данной программой, проходит обучение.

Последним этапом является этап *сопровождения* программы. На этом этапе даются советы пользователям, исправляются обнаруженные в процессе эксплуатации ошибки, а также дается информация о новых, усовершенствованных версиях программы.



1. Каковы этапы разработки больших программ?
2. В чем состоит предназначение транслятора?
3. Чем отличается работа компилятора от работы интерпретатора?

1.5. РЕДАКТОР TURBO PASCAL

Язык Pascal (читается как “Паскаль”) известен и используется сейчас под именем **Turbo Pascal**. В этом разделе мы будем изучать язык Pascal посредством TURBO PASCAL 7.0, созданного компанией Borland International. У Turbo Pascal есть версии как для операционной системы MS-DOS, так и для операционной системы Windows.

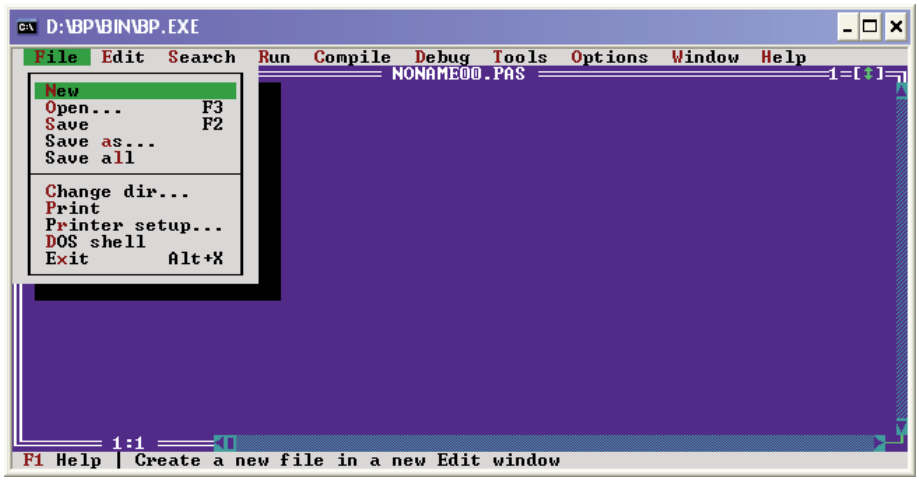
Никлаус Эмиль Вирт (1934)

Язык программирования Pascal был разработан в 1971 году швейцарским ученым Никлаусом Виртом. Pascal был назван в честь французского философа и математика XVII века Блеза Паскаля. Это один из наиболее распространенных языков программирования. Среди других языков он выделяется простотой и логичностью в написании программ и поэтому пользуется популярностью как среди начинающих, так и среди опытных программистов.

Никлаус Вирт является также создателем таких языков, как MODULA и OBERON.

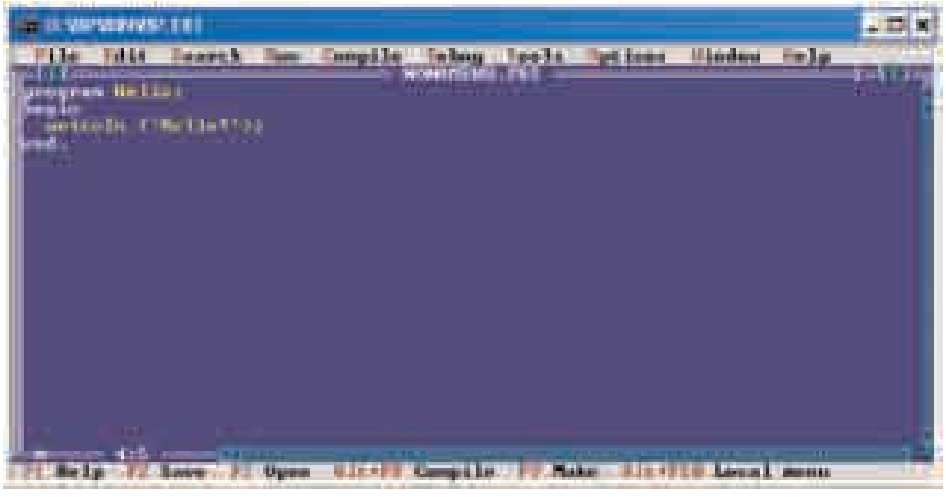
Запуск Turbo Pascal. Программный продукт Turbo Pascal представляет собой *интегрированную среду*. Это значит, что находясь в Turbo Pascal, можно создать, отредактировать, откомпилировать, скомпоновать и загрузить программу.

Для загрузки программы Turbo Pascal необходимо в папке BP (точнее, из папки BIN, находящейся в BP) запустить программу turbo.exe или bp.exe. В результате на экране появится окно, показанное на рисунке.



В верхней части окна расположено главное меню программы (File, Edit, Search и др.). Для выбора одного из пунктов главного меню надо навести указатель мыши на нужный пункт и щелкнуть левой кнопкой. Тот же результат можно получить, если нажав и удерживая клавишу <Alt> на клавиатуре, нажать клавишу с буквой, первой в названии нужного пункта (например, <Alt+F> для File) и затем отпустить обе клавиши.

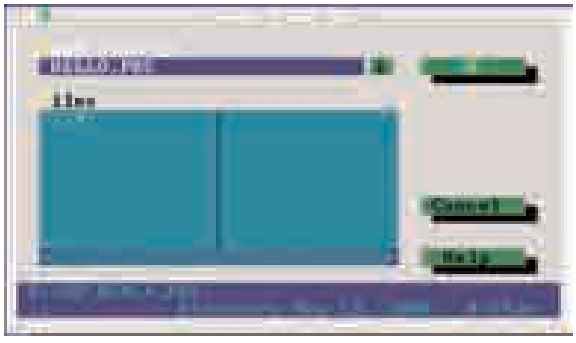
Создание новой программы. Для того, чтобы приступить к созданию новой Pascal-программы, необходимо открыть в рабочей области пустое окно редактора. Для того, чтобы открыть такое окно, достаточно в меню File выбрать пункт New. В результате появится новое окно редактора с именем NONAME00.PAS. Впоследствии при создании новых файлов им будут автоматически присвоены имена NONAME01.PAS, NONAME02.PAS и т.д. Не надо забывать, что созданный таким образом файл хранится в оперативной памяти компьютера. В дальнейшем этот файл необходимо сохранить во внешней памяти компьютера.



Редактор Turbo Pascal обладает достаточными возможностями для набора и редактирования текста программы. Текст программы, как и в любом текстовом редакторе, вводится с помощью клавиатуры. Основной текст программы высвечивается желтым цветом, *ключевые слова*, как элементы языка Pascal, – белым цветом (эти цвета можно изменить, выбрав в основном меню пункт Options⇨Environment⇨Colors). Выделение цветом ключевых слов помогает избежать некоторых ошибок при наборе текста программы.

После завершения ввода программы ее следует сохранить на диске. Это желательно сделать не только в конце работы, но и в процессе набора программы. Для сохранения программы выберите пункт Save в меню File либо

нажмите клавишу <F2>. До этого ваша программа фигурировала в системе Turbo Pascal под именем NONAME00.PAS. Прежде чем сохранить программу под этим именем, редактор дает возможность указать более содержательное имя (например, HELLO.PAS).

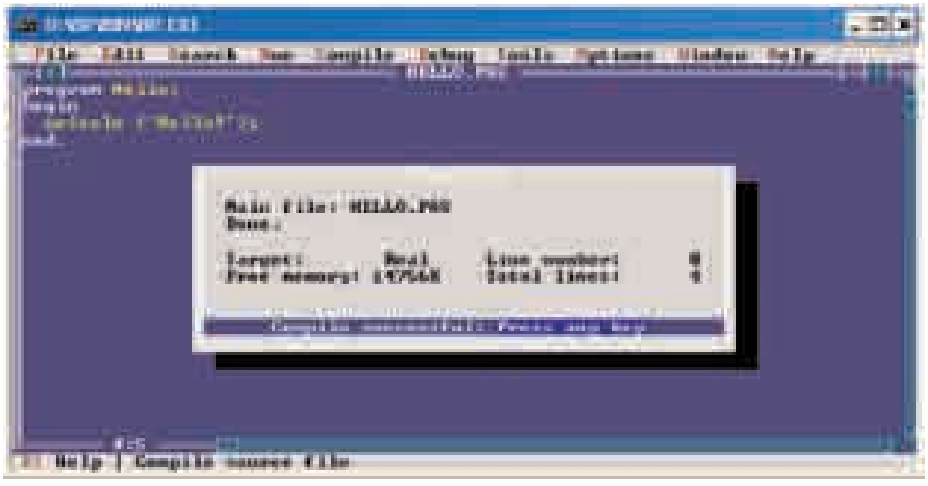


Компиляция и запуск программы. После того, как текст программы набран, его надо откомпилировать. Для этого служит пункт Compile в одноименном меню. После этого Turbo Pascal приступает к компиляции программы, содержащейся в окне редактора. Если в программе будет найдена ошибка, редактор выведет сообщение об ошибке. При этом курсор будет находиться в той позиции программы, где произошла остановка компиляции. Исправив ошибку, следует заново выбрать пункт Compile в одноименном меню. Если в программе уже нет ошибок, то появится окно с сообщением:

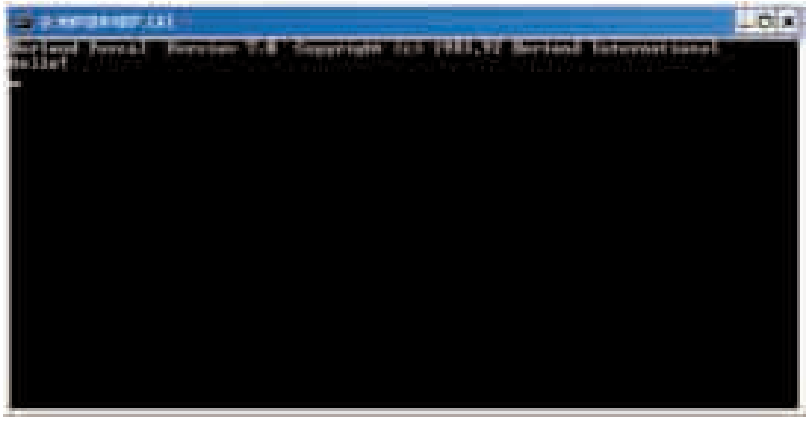
Compile successful: Press any key

(Компиляция прошла успешно; нажмите любую клавишу)

Чтобы вернутся в окно редактора, достаточно нажать какую-либо клавишу на клавиатуре.



Теперь, для того чтобы запустить (на выполнение) только что откомпилированную программу, следует выбрать пункт Run в одноименном меню. В результате на короткое время появится экран пользователя.



Для того, чтобы не спеша рассмотреть этот экран, можно воспользоваться комбинацией клавиш <Alt+F5>. Снова появится экран пользователя и останется перед вами, пока вы не нажмете какую-либо клавишу на клавиатуре.

Загрузка ранее сохраненной программы. Для того, чтобы загрузить в окно редактора ранее сохраненный на диске файл, следует вызвать на экран диалоговое окно Open a File. Для этого нужно выбрать пункт Open в меню File либо нажать клавишу <F3>. После появления указанного диалогового окна требуется либо ввести имя нужного файла в поле Name, либо выбрать имя этого файла в списке Files.

Выход из Turbo Pascal. Для того, чтобы выйти из Turbo Pascal и вернуться в среду операционной системы, необходимо выбрать пункт Exit в меню File. Если выбран этот пункт, а в окне редактора открыт еще не сохраненный файл, система предоставит возможность сохранить этот файл на диске. Для этого на экране появится специальное диалоговое окно.

Если теперь нажать на клавиатуре клавишу <Y>, перед выходом из Turbo Pascal файл будет сохранен.

1. Что такое интегрированная среда?
2. Как определить ошибки, имеющиеся в программе?
3. Как запускается программа на выполнение в среде Turbo Pascal?
4. Как можно просмотреть результаты программы после ее выполнения?

1.6. ОБЩАЯ СТРУКТУРА ПРОГРАММЫ

Любая программа, написанная на языке Pascal, состоит из двух частей – *раздела описания переменных* и *тела программы*.

В тексте программы раздел описания переменных предшествует телу программы.

> Любой объект, встречающийся в программе, должен быть заранее описан в разделе переменных!

Данные обрабатываются с помощью *операторов*. С операторами более подробно вы познакомитесь на следующих уроках.

Тело программы начинается со слова begin и содержит набор операторов. Поэтому иногда этот раздел называют *разделом операторов*. Этот раздел заканчивается ключевым словом end. (с точкой). Программу на языке Pascal в общем виде можно представить так:

```
program < имя программы >;
 < описание переменных >
begin
 < операторы >
end.
```

Программу можно написать в одной строке или же в нескольких строках – от этого ее смысл не меняется (только нельзя переносить слова с одной строки на другую). Надо стараться, чтобы программы были написаны как можно понятнее.

Комментарии. При написании программы вы ясно представляете цели ее написания. Но по истечению некоторого времени, вернувшись к просмотру программы, станет ясно, что некоторые детали вы подзабыли. Поэтому, чтобы сделать программу понятнее как для вас, так и для других, целесообразно в некоторых ее местах поставить комментарии.

```
BLAISE PASCAL
BEGIN
(* 1623 – 1662 *)
END.
```

Как видно из названия, *комментарии* – это заметки для читающего текст программы. Комментарии можно использовать, чтобы указать назначение

программы, сведения об авторах программы, дату последних изменений в ней, назначение переменных, функций и т.д.

Комментарии в программе Pascal пишутся между символами (* и *), или же между символами { и }. Следует отметить, что компилятор игнорирует комментарии и не транслирует их на машинный язык.

```
program Words
{
 Разработчик: Алпай Джалаллы
 Дата создания программы: 13.05.2009
Эта программа определяет количество слов
в тексте.
}
```

Комментарии всегда находятся либо между скобками со звездочками, либо между фигурными скобками.:

(* ...*)

{ ...}

```
program Comments;
(* простая программа, показывающая применение
   комментариев  *)
var a : Integer;
begin (* начало *)
  a:= 1;
  WriteLn('Это все будет напечатано, a=', a)
   { нет необходимости ни в одном из этих
   комментариев  }
end.
```

Идентификаторы. Для того, чтобы именовать различные объекты, например переменные, константы, функции и т.д., в языках программирования используют *идентификаторы*. Несмотря на то, что правила записи идентификаторов различны в различных языках программирования, все же существуют основные принципы выбора идентификаторов:

1. Идентификатор должен представлять собой комбинацию только букв и цифр и начинаться только с буквы.
2. В идентификаторе нельзя использовать пробел и знаки препинания. Можно использовать некоторые символы, такие как “_” , или же “$”.

3. Во всех языках программирования для записи операторов языка существуют ключевые слова. Ключевые слова не могут использоваться в качестве идентификаторов.
4. Идентификатор может быть записан как прописными, так и строчными буквами. Идентификаторы, записанные прописными и строчными буквами, по-разному распознаются в языках программирования. Например, идентификаторы Sum и sum считаются одинаковыми в языках BASIC и Pascal, но различными в языке C.

Примеры возможных идентификаторов приведены ниже:

```
i
a
t0123456789
NoClass
```

Нельзя использовать нижеприведенные идентификаторы:

`1stPlace` – начинается с цифры;

`one and one` – имеется символ пробела;

`yes(no)` – наличие скобок.

Идентификаторы можно разделить на две категории:

1) стандартные идентификаторы;

2) идентификаторы, определенные пользователем.

Все указанные до этого идентификаторы относятся ко второй группе. К стандартным идентификаторам можно отнести процедуры языка – ReadLn, WriteLn, Real и др.

Переменные. Стандартные типы. Назначение раздела описаний Pascal – программы состоит в том, чтобы сообщить компилятору имена всех идентификаторов пользователя, описанных в программе, а также указать, как может быть использован тот или иной идентификатор. Кроме того, данный раздел сообщает компилятору, какие данные будут содержаться в каждой ячейке памяти, используемой в программе.

Представление конкретного значения в памяти зависит от типа данных этого значения. В стандартном Pascal существует четыре предопределенных типа данных. Это Real (предназначенный для вещественных чисел), Integer (для целых чисел), Char (для отдельных символов) и Boolean (для величин, принимающих всего два значения: True и False). Turbo Pascal предоставляет еще один тип данных – string, о котором пойдет разговор на другом уроке. Для каждого типа данных существует свой набор допустимых значений и операций.

Тип данных Integer. В математике целые числа могут быть положительными или отрицательными. Для представления целых чисел в Pascal- программе и предназначен тип данных Integer. Turbo Pascal способен манипулировать значениями этого типа от -32768 до 32767. Существует предопределенная константа MaxInt , значение которой соответствует наибольшему значению типа Integer (т.е. MaxInt = 32767) и которую можно использовать в программах. Вот примеры значений типа Integer:

-1050 425 15 -25

Целочисленные значения можно выводить на экран, выполнять над ними арифметические операции (сложение, вычитание, умножение и деление), а также сравнивать их.

Тип данных Real. Всякое вещественное число состоит из целой и дробной частей, которые разделены десятичной точкой. В Pascal для представления вещественных чисел предназначен тип Real, который начинается и заканчивается цифрой. Поэтому, чтобы дробь – .25 и целое число 64. можно было отнести к типу Real, они должны быть представлены как – 0.25 и 64.0 соответственно.

Вещественные числа можно считывать и отображать на экране, выполнять над ними арифметические операции (сложение, вычитание, умножение и деление), а также сравнивать их.

Тип данных Char. Значения типа Char представляют собой отдельные символы – буквы, цифры или специальные знаки, которые заключены в апострофы:

'A', 'z', '1', ':', '"', ' '

Здесь предпоследний знак – это символ " (кавычки), а последний – пробел.

Арифметические операции над значениями типа Char невозможны. Другими словами, операция '3' + '5' в Pascal недопустима. Можно сравнивать символы, а также считывать и выводить их на экран.

Тип данных Boolean. В отличие от других типов данных, тип Boolean допускает только два значения: True (истина) и False (ложь). Данные этого типа могут быть использованы для условных значений, позволяющих программе принять некоторое решение. Значения типа Boolean можно выводить на экран, однако такое значение нельзя ввести с клавиатуры. Над этими типами данных можно выполнять операции **not** (не), **and** (и) и **or** (или).

Термин “переменная” пришел в программирование из математики. В математике широко используются понятия”переменная величина”, “зависимая величина”. Зависимую величину называют еще функцией. Помимо этого, в математике используется понятие “*аргумент*”, что означает “*независимая, самостоятельная переменная*”. Формула

$$S = v \cdot t$$

как в физике, так и в математике указывает на то, что величина S (путь) зависит от независимых переменных v (скорость) и t (время).
Вследствие того, что первыми создателями языков программирования были математики, многие термины перешли в программирование именно из математики.

Операции отношения. Над целыми числами можно производить операции отношения, результатом которых является логическое значение. Операции отношения бывают следующие:

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

Эти операции применимы и к вещественным числам. К символьным величинам применимы только операции = и <>.

Раздел описания переменных в программе. Этот раздел представлен в виде:

```
var описание1; описание2; описание3; ...
```

Здесь “**var**” — ключевое слово (сокращенное от англ. “variable” — переменная). Каждое “описание” состоит из одного или несколько идентификаторов, разделенных между собой запятой, символом “:” (двоеточие) и типа этих идентификаторов (идентификатора). Оператор описания указывает компьютеру, какие переменные и какого типа используются в программе. Все переменные, имеющиеся в программе должны быть *описаны,* или *объявлены* (причем один раз!).

```
var a, b, c : Real; c : Char;
    n, q1, MaxI : Integer;
        flag : Boolean;
```

Выражения. Приоритет операций. В большинстве программ невозможно обойтись без арифметических выражений. Все арифметические операторы, способы их записи и вычисления представлены в таблице.

Арифметический оператор	Действие оператора	Примеры
+	Сложение	5 + 2 равно 7 5.0 + 2.0 равно 7.0
–	Вычитание	5 - 2 равно 3 5.0 - 2.0 равно 3.0
*	Умножение	5 * 2 равно 10 5.0 * 2.0 равно 10.0
/	Вещественное деление	5 / 2 равно 2.5 5.0 / 2.0 равно 2.5
div	Целочисленное деление	5 **div** 2 равно 2
mod	Вычисление остатка	5 **mod** 2 равно 1

Каждый оператор манипулирует двумя операндами, которые могут представлять собой константы, переменные или арифметические выражения. Операторы +, – , * и / могут использоваться со значениями типов Real и Integer. Как показано в последнем столбце, при использовании операторов +, – и * тип получаемого результата совпадает с типом операндов. Результат применения оператора деления / всегда представляет собой вещественное число. Последние два оператора (**div** и **mod**) могут быть использованы только с целыми числами.

Используя перечисленные выше операторы, можно составить выражение из констант (постоянная величина) и переменных. Например:

```
(a + b) / c
(MaxI * n + q1) div (n + q1)
(flag or not(a = b)) and (n <> q1)
```

При определении порядка вычислений выражений придерживаются стандартных правил старшинства операций: первыми выполняются операции в скобках; затем операции умножения, деления и вычисления остатка; в завершение — операции сложения и вычитания.

1. Что такое идентификатор?

2. Что из нижеперечисленного не является идентификатором? Ответ обоснуй.

```
end  ReadLn program 123XYZ XYZ123
Y=Z  'Max' Ay01 Ay_01 1 Ay
```

3. Из каких основных частей состоит Pascal-программа?

4. Каков порядок наименования переменных?

5. Исправьте синтаксические ошибки в комментариях.

```
{ Это комментарий  *)
{ Это тоже {похоже} на комментарий }
```

1.7. ОПЕРАТОРЫ

Программы, написанные на языке Pascal, состоят из описания переменных и различных операций, производимых над ними – *операторов*.

Данные обрабатываются при помощи операторов. Операторы бывают двух видов: *неисполняемые* (для описания структуры данных и программы) и *исполняемые* (для выполнения различных операций). Вы уже знакомы с оператором описания переменных. Ниже вы познакомитесь с исполняемыми операторами.

Оператор присваивания. Для задания значений переменным или для их изменения во всех языках программирования существует *оператор присваивания*. Общий вид этого оператора таков:

<идентификатор> <символ присваивания> <выражение>

Слева от символа присваивания стоит идентификатор переменной, которой присваивается новое значение. *Символ присваивания* различен для многих языков программирования. Например, для языков BASIC и C символ присваивания – это *знак равенства* (=), а в языке Pascal – это двоеточие и знак равенства (:=).

Примеры операторов присваивания:

x := 5; переменной *x* присвоено число *5*;

y := x; переменной *y* присвоено значение переменной *x*;

y := x + 10; переменной *y* присвоено значение, которое на *10* больше значения *x*;

x := x - 2; переменной *x* присвоено значение, меньшее на *2* предыдущего значения *x*;

y := y + 1; переменной *y* присвоено значение *y*, увеличенное на *1*.

```
program Happiness;
var
   I, You, We: Integer;
begin
   I := 1;
   You := 1;
   We := I + You;
end.
```

Операторы ввода и вывода. Во время работы программы все данные, используемые в программе, хранятся в оперативной памяти.

Программа рассматривает данные, размещенные на других носителях, как *внешние данные*. Получение данных от внешних источников называется *операцией ввода*, передача же данных на внешние носители называется *операцией вывода*.

> Ввод – прием данных от внешних источников.
> Вывод – передача данных на внешние приемники.

Процедура WriteLn. Во многих программах, которые находятся во взаимной интерактивной связи с пользователем, используется оператор вывода информации на экран. Для вывода данных на экран на языке Pascal применяется стандартная процедура WriteLn. Переменные и выражения, которые необходимо вывести на экран, становятся параметрами этой процедуры.

```
WriteLn('Всего ', a);
```

Этот оператор отображает на экране два элемента – строку 'Всего ' и значение переменной a. Если до выполнения этого оператора значение переменной будет, например 2.345, то на экране отобразится

```
Всего 2.3450000000E+00
```

Для отображения вещественного числа, если это не указано иначе, используется экспоненциальный формат Pascal.

Предположим, в программе имеется последовательность операторов:

```
WriteLn('Всего ', a);
WriteLn;
WriteLn('Конец ');
```

Эти операторы при выполнении программы отобразят на экране строки:

```
Всего 2.3450000000E+00
Конец
```

У второго оператора здесь список вывода отсутствует, и поэтому он выводит только пустую строку.

Итак, процедура WriteLn выводит значение каждой переменной или константы, представленной в списке вывода, а затем переводит курсор в начало следующей строки. Если в списке вывода присутствует строка символов, то она печатается без апострофов. Если список вывода отсутствует, курсор просто переводится в начало следующей строки.

Процедура Write. В Pascal имеется еще одна процедура для вывода данных, а именно Write, которая совпадает с WriteLn во всех отношениях, за исключением того, что после отображения списка вывода, курсор не переводится в начало следующей строки. Так, пара операторов:

```
Write('Всего ');
WriteLn(a);
```

эквивалентна одному оператору

```
WriteLn('Всего ', a);
```

Процедура ReadLn. Для ввода данных с клавиатуры в Pascal используется процедура ReadLn. Общий формат этой процедуры таков:

```
ReadLn(список ввода)
```

Процедура ReadLn считывает в память данные, которые пользователь вводит с клавиатуры при выполнении программы. Для каждой переменной, указанной в списке ввода, пользователь должен ввести один элемент данных, а в конце нажать клавишу <Enter>. Имена переменных в списке ввода разделяются запятыми. Порядок ввода данных должен соответствовать порядку, в котором переменные представлены в списке ввода. Вводимые числовые элементы данных разделяются одним или несколькими пробелами. Запятые между данными или внутри данных присутствовать не должны.

```
var a, b: integer;
. . .
Write('Введите значение переменных: ');
ReadLn(a, b);
```

Процедура Read. Еще одним средством ввода данных с клавиатуры является процедура Read. Основное отличие между процедурами Read и ReadLn состоит в том, что все избыточные символы в строке данных процедурой Read не считываются (эти символы могут быть считаны следующей процедурой Read или ReadLn). А процедура ReadLn, напротив, обрабатывает все символы во вводимой строке, однако игнорирует при этом все избыточные символы в конце строки.

Форматирование. Как было отмечено, если нет других указаний, Pascal отображает все вещественные числа в экспоненциальном виде. Как же можно представлять данные в нужном нам формате? Проще всего указать формат переменной или значения типа Integer, которое должно быть выведено Pascal-программой. Для этого после имени переменной (или после зна-

чения) достаточно добавить двоеточие и число, указывающее ширину поля, отводимого для этого числа. Если эта ширина меньше длины числа, то ширина игнорируется. Операторы

```
Write('a = ', a:1);
WriteLn(' и b = ', b:2);
```

указывают, что для значения a отводится одна позиция, а для значения b – две позиции. Например, если значение переменной a равно 7, а b равно 8, то в этом случае вывод программы будет выглядеть так:

```
a = 7 и b =  8
```

Внимательно вглядевшись, можно заметить дополнительный пробел перед значением переменной b. Это объясняется тем, что спецификация формата :2 обеспечивает место для вывода двух цифр.

Для того, чтобы задать формат вывода для переменной или значения типа Real, необходимо указать как ширину поля, так и необходимое число десятичных разрядов. Общая ширина поля должна быть достаточна велика, чтобы вместить все разряды до и после десятичной точки. При этом одно знакоместо необходимо выделить для десятичной точки и одно – для знака минус, если данная переменная может принимать отрицательные значения.

Если X – это переменная типа Real, которая может принимать значения в пределах от -99.9 до 999.9, оператор

```
WriteLn(X :5 :1);
```

выведет значение X с точностью до одного десятичного знака. В нижеприведенной таблице показаны спецификации формата для целых и вещественных чисел.

Значение	Формат	Вывод
234	:4	234
234	:5	234
234	:1	234
-234	:6	-234
3.14159	:5:2	3.14
3.14159	:5:3	3.142
0.1234	:4:2	0.12
-0.006	:8:3	-0.006
-0.006	:8:5	-0.00600
-0.006	:7:5	-0.00600

Константы. Помимо переменных, для описания данных в программе используются постоянные выражения (константы). На языке Pascal имеется возможность описывать константы и задавать им имена, которые впоследствии используются в тексте программы вместо самих констант.

Константы описываются в *разделе описания констант*, начинающемся с ключевого слова const:

```
const im1 = значение1;
      im2 = значение2;
      .........
      imN = значениеN;
```

Здесь im1, im2,... – произвольные идентификаторы, значение1, значение2, ... – записанные по правилам числа, символы, заключенные в апострофы, константы true, false. Например:

```
const g = 981E-2;
      atmosfer = 0.76;
      pi = 3.1415926;
```

Можно указать как минимум две причины, по которым желательно иметь в программе константы.

Во-первых, традиция использования вместо постоянных величин букв заимствована из физики и математики. Сохраняя ее, программы становятся более понятными. Задание константам осмысленных имен − один из способов комментарий в программе. Запись “длина_строки” более информативна, чем запись “60”.

Во-вторых, описание констант облегчает внесение изменений в программу. Например, пусть значение длины строки, описанной в программе, надо взять не 60, а 40. Для этого достаточно в разделе описания констант изменить “длина_строки = 60” на “длина_строки = 40”. В противном случае придется найти в программе все числа 60, удостовериться, что это значение длины строки, и поменять их на 40.

Определение нового типа данных. Программист помимо стандартных типов данных имеет возможность задавать новые типы данных и давать им имена. После этого их можно использовать как стандартные типы данных. Раздел определения типов имеет такой вид:

```
type im1 = описание1;
     im2 = описание2;
     .........
     imN = описаниеN;
```

Здесь im1, im2,... – произвольные идентификаторы, а описание1, описание2, ... – описание этих типов данных.

Инициализация переменных. Под *инициализацией переменных* понимают присвоение начальных значений переменным до выполнения или присвоение значений во время работы программы. На языке Pascal переменную можно также инициализировать во время ее объявления. Например:

```
const i: Integer = 3;
```

Этот оператор не является ни оператором ввода, ни оператором присваивания, а i является фактически переменной, а не константой. Переменная i называется *типизированной константой*.

1. В чем заключается преимущество задания имен константам в программе?

2. Дан фрагмент программы:

```
const
 MyPi = 3.14159;
 MaxI = 1000;
var
 X, Y : Real;
 A, B, I : Integer;

 A := 3;
 B := 4;
 Y := -1.0;
```

Найдите допустимые операторы и укажите их значения.

```
а) I := A mod B
б) I := (990 - MaxI) div A
в) I := B div 0
г) X := A / Y
д) X := MyPi div Y
е) I := (MaxI - 990) mod A
```

3. Чем отличается процедура `WriteLn` от процедуры `Write`?

4. Запишите нижеследующие словесные алгоритмы соотвествующими операторами языка Pascal.

Выберите произвольное число между 1 и 100.
Умножьте это число на само себя.
К полученному числу прибавьте выбранное число, умноженное на 4.
К результату прибавьте 3.
Разделите полученный результат на сумму выбранного числа и 3.
От частного вычтите выбранное число.
Выведите ответ на экран.

1.8. ОПЕРАТОРЫ ВЫБОРА IF И CASE

При выполнении алгоритма команды обрабатываются последовательно, одна за другой. Но в жизни редко встречаются такие задачи, где команды выполняются последовательно. Для решения более сложных задач требуются гибкие алгоритмы, в которых последовательность выполнения шагов меняется.

В алгоритмах выбор одного из возможных действий осуществляется с помощью *ветвления*. Ветвление является одной из основных алгоритмических структур. Ветвление основывается на проверке одного или нескольких условий, и в зависимости от истинности этих условий выполняется определенное действие.

Во всех языках программирования предусмотрены специальные операторы, обеспечивающие ветвление. Такие опреаторы называют *условными операторами*. Простой условный оператор состоит из двух частей:

1) условия;
2) выполняемого оператора.

Выполняемый оператор выполняется в случае истинности условия.

```
if x < 5 then x := x + 1;
```

Здесь **if** (если) – ключевое слово, которое показывает начало условного оператора. После него следует условие. Затем следует ключевое слово **then** (то). В конце стоит *выполняемый* оператор. Такой условный оператор называют также *одноальтернативным*.

Если при выполнении условия необходимо выполнить несколько операторов, то используют "*операторные скобки*" (ключевые слова begin и end). Например:

```
if X < 5 then
begin
   X := X + 1;
   Y := Y + 1;
end;
```

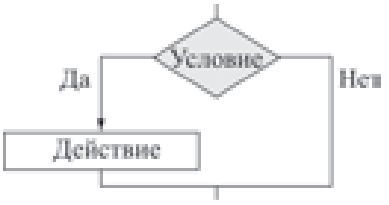
> Ключевые слова **begin** и **end** называют *операторными скобками*.

Полная форма условного оператора. Очень часто при истинности условия требуется выполнение одного, а при ложности условия – другого действия. В таком случае используется *полная форма* условного оператора.

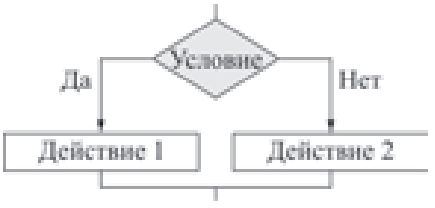
```
if X < 5 then
   X := X + 1
else
   X := X - 1;
```

При истинности уловия выполняется оператор (операторы), следующий за словом **then**. Если же условие ложно – выполняется оператор (группа операторов), следующий за словом **else** (иначе). Такой оператор называют иногда *двухальтернативным* условным оператором.

> На языке Pascal перед ключевым словом **else** не ставится (;).



Блок-схема одноальтернативного условного оператора



Блок-схема двухальтернативного условного оператора

Цепочка условных операторов. Во многих задачах требуется проверка нескольких условий. В этом случае в алгоритме образуется цепочка условных операторов.

```
if X < 5 then
   X := X + 1
```

```
else if X < 10 then
   X := X - 1
else if X = 13 or X = 15 then
   X := X * 2
else
   X := 10;
```

Цепочку условных операторов можно рассматривать как группу вложенных друг в друга операторов. В ее записи следующее ключевое слово **if** должно располагаться на одной строчке с **else**, так как в этом случае структура программы выглядит более простой и понятной.

В цепочке выполняется только один оператор, обособленный командами **else if**.

Оператор выбора. Если в цепочке условных операторов в качестве условия используется одно и то же выражение и получает оно только целочисленное значение, в таком случае можно упростить написание данной группы операторов. Для этого служит *оператор выбора*. На языке Pascal оператор выбора начинается со строки **case of**. Между этими двумя словами записывается выражение, которое проверяется. Используя оператор выбора, можно записать приведенную выше цепочку условных операторов таким образом:

```
case X of
    1..4:   X := X + 1;
    5..9:   X := X - 1;
    13,15: X := X * 2;
else
    X := 10;
end;
```

Проверочные значения задаются отдельно в виде чисел и интервалов. Чтобы задать интервал, надо указать минимальное и максимальное значение и поставить между ними две точки (. .). Если один и тот же оператор нужно выполнить для различных значений проверочного выражения, эти значения (и интервалы) записываются через запятую. После списка значений ставится двоеточие, а затем записывается оператор, который выполняется при совпадении проверочных значений с хотя бы одним значением из списка. В противном случае значение выражения сравнивается со списком, заданным на следующей строке.

Если значение выражения не совпадает ни с одним из значений из списков, выполняется оператор, следующий за ключевым словом **else** (раздела **else** может и не быть). Оператор выбора всегда заканчивается ключевым словом **end**.

Если одно и то же значение указано сразу в нескольких списках, то выполняется оператор, соответствующий первому списку с таким значением. Затем управление передается команде, следующей за оператором выбора.

1. Что такое ветвление и каким оператором оно реализуется в программировании?

2. Во фрагменте программы поменяй местами строки таким образом, чтобы условный оператор был записан верно.

```
b := a + 2
b := a - 2;
if a > 2 then
else
```

3. Какое условие должно быть записано в условном операторе для того, чтобы переменная `c` получила минимальное из значений `a` и `b`?

```
if ... then
   c := a
else
   c := b;
```

4. Какое сообщение отобразится на экране в результате работы данного оператора?

```
if 12 < 12 then
   WriteLn ('Меньше')
else
   WriteLn ('Не меньше');
```

1.9. ЦИКЛЫ. ОПЕРАТОРЫ WHILE, FOR И REPEAT

При разработке алгоритма задач иногда приходится выполнять некоторые команды несколько раз подряд. Конечно, можно написать такую последовательность команд нужное число раз. Но это не совсем приемлемый путь.

Если число команд и повторений очень велико, то может получиться длинная запись алгоритма. Помимо этого, во многих алгоритмах заранее неизвестно число повторений, и только во время выполнения программы оно может стать известным. Чтобы избавиться от этой проблемы, в программах используют специальную алгоритмическую структуру – *цикл*.

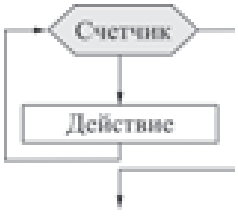
Во всех языках программирования есть конструкции для создания цикла. Цикл состоит из трех основных частей:

1. *Инициализация* – подготовка к выполнению цикла;
2. *Тело цикла* – группа повторяющихся операторов;
3. *Условие окончания (либо продолжения) цикла* – проверяется перед выполнением тела цикла и используется для того, чтобы проверить, когда закончится цикл.

Цикл со счетчиком. Если заранее известно количество повторений тела цикла, то с легкостью можно написать оператор цикла. Для этого используют *цикл со счетчиком*.

```
for <переменная цикла> := <нижняя граница цикла>
    to <верхняя граница цикла> do
    <тело цикла>
```

Счетчик, или *переменная цикла,* является служебной переменной и он меняется автоматически во время выполнения цикла. Первый оператор цикла (его и называют *оператором цикла*) показывает *границы цикла*.



Блок-схема цикла со счетчиком

Цикл выполняется в следующем порядке:

1. Если границы цикла заданы в виде выражений, то вначале вычисляются их значения.
2. Переменной цикла присваивается нижнее значение цикла.
3. Переменная цикла сравнивается с верхней границей цикла.
4. Если значение переменной цикла больше верхней границы цикла, то прекращается его выполнение.
5. В противном случае выполняется тело цикла.
6. Текущее значение переменной цикла увеличивается на 1.
7. Выполнение цикла продолжается с шага 3.

Допустим, требуется найти сумму первых десяти натуральных чисел. Фрагмент программы для этой задачи можно записать так:

```
S := 0;
for I := 1 to 10 do
  S := S + I;
```

Первый оператор, находящийся за пределами цикла, задает начальное значение переменной S. Очень часто до выполнения цикла требуется провести подготовительные работы.

В качестве счетчика здесь используется переменная I. Границы цикла заданы в виде констант 1 и 10. Если нижняя граница цикла больше по значению ее верхней границы, то вместо ключевого слова to используется ключевое слово downto. В таком случае при выполнении цикла значение счетчика уменьшается.

Особенности цикла со счетчиком

Цикл со счетчиком имеет некоторые особенности. Если границы цикла заданы в виде выражений, то значения этих выражений вычисляются в период *инициализации цикла*. Даже если значения переменных, входящих в эти выражения, будут меняться в процессе выполнения цикла, это никак не повлияет на число выполнений тела цикла.

Проверка условия конца цикла происходит до первого выполнения тела цикла. Границы цикла могут иметь такие значения, что тело цикла не будет выполнено ни разу.

Цикл с условием. Цикл с условием является более общей формой составления цикла. Обычно таким циклом пользуются в случае, когда число повторений цикла заранее не известно.

Различают два вида цикла с условием:

1) цикл с предусловием
2) цикл с постусловием.

Цикл с предусловием работает так:

1. Проверяется условие повторения для цикла.
2. Если условие не выполняется, то работа цикла прекращается.
3. При выполнении условия выполняется тело цикла.
4. Выполнение цикла заново начинается с 1-го шага.

Общая форма записи цикла с предусловием такова:

```
while <условие> do
  <тело цикла>;
```

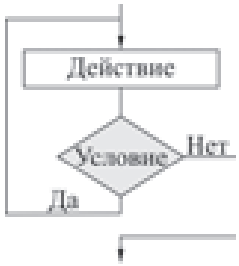
> В цикле с предусловием тело цикла может не выполниться ни разу.

Используя цикл с предусловием, фрагмент алгоритма вычисления суммы квадратов первых ста натуральных чисел можно записать так:

```
S := 0;
I := 1;
while I <= 100 do
  begin
    S := S + I * I;
    I := I + 1;
  end;
```

Цикл с постусловием работает так:

1. Выполняется тело цикла.
2. Проверяется условие повторения цикла.
3. Если условие не выполняется, цикл прекращает работу.
4. Если условие выполняется, работа программы продолжается с 1-го шага.



Блок-схема цикла с постусловием

Во многих языках программирования может потребоваться проверка условия для окончания цикла.

Цикл с постусловием записывается таким образом:

```
repeat
 <тело цикла>
until <условие>;
```

Тело цикла с постусловием выполняется как минимум один раз. На языке Pascal выполнение цикла с постусловием прекращается при выполнении заданного условия. Программист должен составить программу так, чтобы значения переменных, входящих в выражение для условия, изменялись в теле цикла. Если условие цикла останется неизменным, то цикл может выполнятся бесконечно.

Фрагмент программы, вычисляющей сумму квадратов первых ста натуральных чисел, написанной с использованием оператора repeat, может выглядеть так:

```
S := 0;
I := 1;
repeat
  S := S + I * I;
  I:= I + 1;
until I > 100;
```

Выход из цикла. Во многих программах часто бывает необходимо использовать сложные циклы, в которых может быть несколько условий для их окончания. Для упрощения создания таких циклов в современных языках программирования предназначен оператор *выхода из цикла*. Этот оператор чаще всего используется внутри условного оператора, находящегося в цикле. При выполнении этого оператора управление немедленно передается первому оператору, следующему сразу после цикла. Внутри одного цикла может быть сразу несколько операторов выхода из цикла. Оператор выхода из цикла записывается так:

```
break;
```

Нижеследующий фрагмент программы вычисляет сумму чисел, вводимых с клавиатуры. При вводе отрицательного числа цикл прерывается.

```
S := 0;
while True do
  begin
```

```
        Read(I);
        if I < 0 then break;
        S := S + I;
    end;
```

В этом примере использован “бесконечный” цикл, так как условие цикла всегда верно. Но вследствии того, что в цикле использован оператор выхода из цикла, процесс не зацикливается.

1. Из каких частей состоит цикл?

2. С помощью каких операторов описываются цикл со счетчиком, цикл с предусловием и постусловием?

3. Объясните принцип работы цикла со счетчиком.

4. Определите значение переменной k после выполнения последовательности операторов:

```
var i, k: Integer;
 ...
k := 0;
for i := 1 to 100 do
if i mod 2 = 0 then
k := k + 1;
```

5. Составьте программу, вычисляющую сумму квадратов первых ста натуральных чисел, используя оператор for.

6. Номера билетов автобусов – шестизначные: начиная с 000000 до 999999. Билет считается “счастливым”, если сумма первой, третьей и пятой цифры равна сумме второй, четвертой и шестой цифры. Составьте программу, находящую и печатающую все “счастливые” билеты.

7. Какой результат получится при выполнении этого фрагмента программы?

```
for Ch := 'A' to 'Z' do
     Write (Ch);
```

1.10. МАССИВЫ

В программах часто используются величины одного типа. Пронумерованная последовательность однотипных величин называется *массивом*. У массива должно быть имя, и оно относится к каждому элементу массива. Конкретный элемент массива можно определить по его номеру, который называют *индексом* элемента.

Как и для простых переменных, во всех языках программирования массив заранее объявляют (описывают). В описании массива должны быть указаны число элементов массива, возможный диапазон индексов и тип каждого элемента.

```
var a: array [1..10] of Integer;
```

После ключевого слова array в квадратных скобках указывается диапазон индексов элементов, разделенный двумя точками (..). Затем следует ключевое слово of и указывается тип элементов. В данном примере описываемый массив состоит из 10 целых чисел.

Массив, количество элементов которого задается в начале программы, называется *статическим*. В некоторых языках программирования используются *динамические* массивы. У таких массивов количество элементов задается в ходе выполнения программы. Как только массив объявлен, в памяти компьютера для всех его элементов отводится место. Поэтому динамические массивы используют память компьютера более рационально.

Обращение к элементу массива. Для обращения в программе к элементу массива надо указать имя и индекс этого элемента. Например, нижеследующий оператор присваивает 7-му элементу массива a значение его 6-го элемента.

```
a[7] := a[6];
```

В программах очень часто массив обрабатывается как единое целое. Чтобы одно и то же действие применить ко всем элементам массива, можно использовать цикл со счетчиком. В этом случае для обращения к элементу массива счетчик цикла выступает в роли индекса элемента. Допустим, имеется массив b, состоящий из 10 элементов. В нижеследующим операторе цикла всем элементам массива присваиваются нулевые значения.

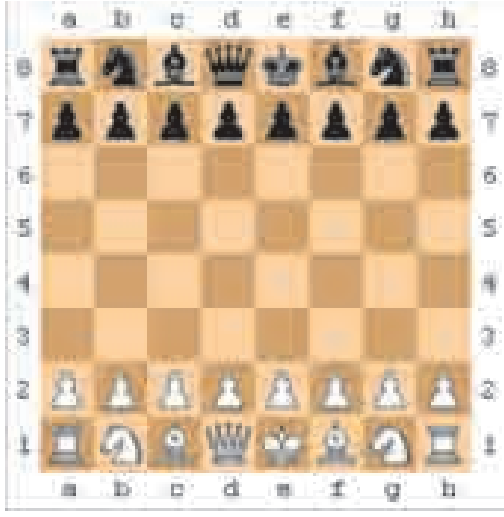
```
for i := 1 to 10 do
    b[i] := 0;
```

Рассмотрим такую задачу. Задан целочисленный массив из n элементов. Требуется найти элемент массива с минимальным значением. Если в программе обозначить минимальный элемент как min, а его индекс – imin, то фрагмент программы для решения данной задачи можно написать так:

```
imin := 1;
min  := a[imin];
for i := 2 to n do
    if a[i] < min then begin
       min  := a[i];
       imin := i;
    end;
WriteLn (min, ' минимальный элемент, ',
         imin, ' индекс минимального
         элемента в массиве.');
```

Двумерные массивы. Массивы с одним индексом называют *одномерными*. Для многих задач такие массивы непригодны. Во многих языках программирования есть возможность создания многоиндексных – *многомерных* массивов. Например, *двумерный* массив можно рассматривать как таблицу, в которой один индекс показывает номер столбца, а второй – номер строки.

Для объявления двумерного массива нужно для каждого индекса указать диапазоны изменения. Эти диапазоны разделяются запятой. Например, целочисленный массив, описывающий шахматную доску, можно объявить так:



```
var t: array [1..8, 1..8] of integer;
```

На языке Pascal двумерный массив можно рассмотреть как “массив массивов”. Тогда предыдущее описание массива эквивалентно следующему:

```
var t: array [1..8] of array [1..8] of integer;
```

При обращении к элементу двумерного массива надо указать два его индекса, отделенные друг от друга запятыми. На языке Pascal можно также указать каждый из индексов отдельно.

Например, к ячейке e4 шахматной доски, являющейся соответствующим элементом массива, можно обратиться так:

```
t[5,4],
```

или

```
t[5][4].
```

Вложенные циклы. При обработке двумерного массива для вычисления значения индексов можно обойтись одним циклом с единственным счетчиком. Но в этом случае будет трудно разобраться в программе.

В алгоритмах работы с двумерным массивом обычно используют два цикла. Счетчик каждого цикла принимает всевозможные значения соответствующего индекса. В этом случае, чтобы охватить все элементы массива, один цикл должен находиться внутри другого. Такие циклы называют *вложенными*. Например, для того, чтобы присвоить всем элементам шахматной доски нулевые значения, надо организовать следующие циклы:

```
for i := 1 to 8 do
    for j := 1 to 8 do
        t[i,j] := 0;
```

При выполнении внутреннего цикла счетчик внешнего цикла остается неизменным. Затем он увеличивается на единицу, и заново идет выполнение внутреннего цикла. Применение вложенных циклов в программировании удобно не только для работы с массивами, но и для решения других задач.

Допустим, задан двумерный массив a, содержащий n строк и m столбцов. Требуется определить, имеется ли в этом массиве хотя бы один отрицательный элемент. Фрагмент программы с использованием цикла с постусловием для решения данной задачи можно написать так:

```
i := 0;
repeat
    j := 0;
    i := i + 1
    repeat
        j := j + 1;
```

```
    until (j = m) or (a[i,j] < 0);
until (i = n) or (a[i,j] < 0);
if a[i,j] < 0 then
  WriteLn('Да, есть отрицательный элемент!')
else
  WriteLn('Нет отрицательного элемента!');
```

1. Что такое массив и как можно описать его на языке Pascal?

2. Составьте программу, находящую максимальный элемент массива `a` среди элементов с индексами от `k` до `m`. Найденное значение присвойте переменной `j`.

3. Дан целочисленный массив, состоящий из 10 элементов. Два элемента этого массива имеют одинаковые значения. Составьте программу, определяющую индексы этих элементов. Значения этих индексов присвойте переменным `i` и `j`.

4. Что будет значением переменной `L` после выполнения данного фрагмента программы? Выберите правильный ответ.

```
var a: array [1..4, 1..4] of integer;
    L, f, g, v: integer;
...
L := 0;
for g := 1 to 4 do
begin
  v := 0;
  for f := 1 to 4 do
      if a[f,g] < 0 then v:= v + 1;
  L := L + v;
end;
```

а) число положительных элементов массива `a`;

б) число не положительных элементов массива `a`;

в) число отрицательных элементов массива `a`;

г) число отрицательных элементов в первой строке массива `a`;

д) максимальное значение отрицательных элементов массива `a`.

1.11. РАБОТА СО СТРОКАМИ

Во всех языках программирования предусмотрена работа с последовательностью символов, другими словами, *работа со строками*.

Строка может быть задана или константой, или значением переменной. На языке Pascal *строковая константа* задается последовательностью символов, взятых в кавычки:

```
'Pascal'
'1234'
'река Араз'
```

'' – тоже специальная строковая константа, имеющая нулевую длину. Такую строку называют *пустой строкой*.

На языке Pascal строковые переменные имеют тип string. Например, в программе запись

```
var a: string;
```

указывает на то, что a – строковая переменная.

Максимальная длина строки зависит от конкретного языка программирования или же от транслятора заданного языка. В языке Pascal переменная типа string может содержать не более 255 символов. В современных языках программирования строки могут быть практически любой длины.

Операции со строками отличаются от операций с числами. Операции сложения, вычитания, умножения и деления не имеют смысла в работе со строками. Основная операция над строками – это их *сцепление,* или *конкатенация*. В результате этой операции вторая строка дописывается в конец первой. Операция конкатенации обозначается символом **+** (плюс).

```
var a, man: string;
...
man := 'десять';
a := man + ' манат';
```

Несмотря на одинаковую запись (**+**), операция сложения отличается от операции конкатенации.

Для объединения строк используют также функцию Concat. Например,

```
a := Concat(man, ' манат');
```

Количество операций над строками превосходит количество операций над числами. Но использование любого из знаков этих операций не очень удобно и не просто. Поэтому остальные операции над строками осущест-

вляются с помощью *стандартных процедур* и *функций*. Во многих языках программирования возможны следующие действия над строками:

- определение длины строки;
- выделение подстроки;
- удаление или добавление символов;
- поиск символа в строке;
- изменение регистра символов (букв) строки.

Ознакомимся с функциями, предусмотренными в языке Turbo Pascal для выполнения этих и других действий.

- **Length** (St). Эта функция вычисляет длину строки St. Под "длиной" строки подразумевается число символов в ней. Длина пустой строки равна 0. Вычисление длины строки S и присвоение этого значения переменной LenS, можно представить так:

```
LenS := Length(S);
```

Внизу приводится фрагмент программы с использованием функции Length.

```
S := 'Hello';
L := Length(S);
WriteLn(L);    { Будет выведено число 5 }

S := 'Hello Students';
WriteLn(Length(S)); { Будет выведено число 14 }

S := '  ';           { Пустая строка }
WriteLn(Length(S)); { Будет выведено число 0 }
```

- **Copy** (St,Index,Count). Иногда требуется выделить и вырезать часть строки. Например, если три компонента (день, месяц, год) в строке '17 мая 2009', присвоенные переменной Date, требуется обработать отдельно друг от друга, то используется функция Copy.
Допустим в программе строка даты записана в формате DD MMM GGGG. Здесь DD – день месяца (1-й и 2-й символ), MMM – краткая запись названия месяца (4 – 6 символы), GGGG – год (8 – 11 символы).

Если Date, Month, Day и Year – строковые переменные, то функция Copy присвоит переменной Day первые два символа переменной Date (‘17’).

```
Day := Copy (Date, 1, 2);
```

Таким же образом, подстрока 'май' присваивается переменной Month, а подстрока '2009' – переменной Year.

```
Month := Copy (Date, 4, 3);
Year  := Copy (Date, 8, 4);
```

- **Insert** (Subst,St,Index). Эта процедура вставляет строку Subst в строку St, начиная с позиции Index. Например, после выполнения функции

```
Insert ('п', 'Алай', 3)
```

строка

```
'Алай'
```

превратится в строку

```
'Алпай'.
```

- **Delete** (St,Index,Count). Эта процедура, начиная с позиции Index, стирает в строке St Count символов. Например, вследствии применения функции Delete (‘Алгоритм’,1,4) строка ‘Алгоритм’ превратится в строку ‘ритм’.
- **Pos** (Subst,St). Эта функция, разыскивающая вхождение подстроки Subst в строку St. В качестве результата своей работы она выдает либо 0 – если вхождения обнаружено не было, либо целое положительное число, являющееся номером позиции, с которой выявлено вхождение. При этом оценивается только первое вхождение. Если вхождений несколько, то обнаружено будет только первое из них и только его позиция будет выдана в качестве результата функции.

```
S := 'средняя школа номер 1';
S1 := 'школа';
J := Pos(S1, S);
WriteLn(J); { будет выведено число 9 }

S1 := 'лицей';
J := Pos(S1, S);
WriteLn(J); { будет выведено число 0 }
```

- **Val** (St,X,Code). Если переменная или константа типа String содержит символьное представление вещественного или целого числа, например, '17.5' или '1234', то с помощью процедуры Val можно преобразовать

это число в значение переменной типа Real. Здесь St – это сама исходная строка, X – это переменная, которой будет передано получившееся вещественное число. Целочисленный параметр Code содержит 0, если преобразование прошло успешно, в противном случае – номер места в строке, в котором произошла ошибка. В данном фрагменте программы переменной NumStr присваивается целое значение.

```
repeat
    Write ('Введите целое число: ');
    ReadLn (NumStr);
    Val (NumStr, IntNum, Error);
until Error = 0;
```

Если введенное число является строкой, то процедура Val присваивает целой переменной IntNum эту строку. Если в веденной строке помимо цифр есть и другие символы, значение переменной Error будет отлично от 0, и произойдет повторение цикла.

- **Str** (X,St). Эта процедура, обратная предыдущей, преобразует число из переменной X в строку, которая присваивается переменной St. Например, процедура Str (123 :5, NumSt) присвоит переменной NumSt строку ' 123'. Здесь 5 – число символов в строке.

В качестве примера применения некоторых процедур рассмотрим программу, которая подсчитывает, сколько раз встречается указанное пользователем слово в строке.

```
program P;
var
    s, s1: string;
    k, i: Integer;

begin
    Write ('Ведите исходную строку: ');
    ReadLn (s);
    Write ('Введите слово, которое следует
    искать: ');
    ReadLn (s1);

    k := 0;
```

```
    while Pos(s1, s) > 0 do
    begin
        k := k + 1;
        Delete(s, Pos(s1, s), Length(s1));
    end;
    WriteLn (k);
end.
```

1. Что такое строка и какие операции производятся над строками?
2. Допустим, что , Temp1 := 'Абра' və Temp2 := 'кадабра'.
   Определите результаты выполнения приведенных ниже функций и процедур.
   а) `Magic := Concat(Temp1, Temp2)`
   б) `Length(Magic)`
   в) `HisMagic := Copy(Magic, 1, 8)`
   г) `Delete(HisMagic, 4, 3)`
   д) `Insert(Temp1, HisMagic, 3)`
   е) `Pos(Temp2, Magic)`
   ж) `Pos(Temp1, Magic)`
   з) `Val('1.234', RealNum, Error)`
   и) `Str(1.234 :3:1, RealStr)`
3. *Палиндром строки* состоит из самой строки и строки, записанной в обратном порядке. Запишите программу, создающую палиндром строки. Например, на входе задается строка 'abc', а на выходе должна получиться строка 'abccba'.
4. Составьте программу, вычисляющую число гласных букв в строке.

1.12. ПОДПРОГРАММЫ. ФУНКЦИИ И ПРОЦЕДУРЫ

В программе, содержащей пару сотен строк, еще можно как-то разобраться. Но с увеличением количества строк теряется общая логика работы. Даже зная, какие элементарные действия производят конкретные операторы, понять общее назначение программы бывает довольно сложно. Становится неясной структура программы и последовательность ее выполнения. В результате отладка программы приобретает хаотичный характер: становится сложно отслеживать исправления и вносить их в программу.

Для решения этой проблемы алгоритм разбивается на отдельные алгоритмы, выполняющие простые действия. Такие алгоритмы называют *вспомогательными алгоритмами*. В программировании вместо термина "вспомогательный алгоритм" употребляется термин "*подпрограмма*". Для обращения к вспомогательному алгоритму (подпрограмме) ее надо вызвать.

Обычно при разработке программ среднего объема возникает необходимость разделения их на несколько подпрограмм, каждая из которых выполняет несложные вычисления. Итоговый алгоритм состоит не из отдельных операторов, а из блоков, имеющих собственные имена. В этом случае подпрограммы можно рассматривать как новые операторы, разработанные программистом.

Стандартные подпрограммы. Многие вспомогательные алгоритмы очень часто используют в различных задачах. Например, очень часто требуется вычислить значения математических функций или же произвести стандартные операции над строками. Если бы каждый программист сам составлял такие алгоритмы, то он терял бы много времени. Эту проблему решают, используя стандартные подпрограммы.

Обычно стандартные подпрограммы определены не в языке программирования, а в среде (системе) программирования. Они включены в библиотеку стандартных подпрограмм, дополнительно к транслятору. Большие библиотеки стандартных подпрограмм значительно облегчают работу программиста.

Типы вспомогательных алгоритмов. Подпрограммы обычно делят на две категории: *процедуры* и *функции*.

> Процедура обычно выполняет какое-то самостоятельное отдельное действие. Функция же вычисляет конкретное значение и пересылает (возвращает) это значение вызывающей ее программе (или подпрограмме).

В некоторых языках программирования (например, на языке С) нет разделения подпрограмм на функции и процедуры. Их всех рассматривают как функции. В программах, написанных на этих языках, процедура – это не возвращающая значение функция.

Параметры подпрограммы. Для того, чтобы работа подпрограммы имела какой-то смысл, надо, чтобы она получала данные из вызывающей ее программы. Данные передаются подпрограмме в виде *параметров*. Каждая подпрограмма ждет получения совокупности данных конкретного типа. Существуют подпрограммы, в которых нет необходимости в параметрах.

Параметры **Формальные параметры** **Фактические параметры**

При создании подпрограммы заранее неизвестны значения, которые передаются ее параметрам. При описании подпрограммы в ее заголовке указываются *формальные параметры*. Формальные параметры – это произвольные идентификаторы, определяющие тип переданных данных. Они нужны только в представлении операций в подпрограмме.

Для вызова подпрограммы указываются уже *фактические параметры*. Во время выполнения подпрограммы формальные параметры заменяются соответственно фактическими параметрами.

Вызов подпрограммы. Вид оператора, вызывающего подпрограмму, зависит от типа подпрограммы и от синтаксиса конкретного языка программирования. Чтобы вызвать подпрограмму, нужно указать ее имя, затем в скобках показать список фактических параметров. Тип и количество фактических параметров должны совпадать с типом и количеством формальных параметров в подпрограмме. Фактическими параметрами могут быть не только переменные, но и константы, выражения.

Функцию можно вызвать из любого места программы. В нижеследующем примере значение длины гипотенузы прямоугольного треугольника с катетами x и y присваивается переменной z.

```
z := sqrt(x*x + y*y);
```

Вызов процедуры оформляется обычно как отдельный оператор. Для этого на языке Pascal не требуется особого ключевого слова. Например, процедуру P, принимающую в качестве параметра два целых числа, можно вызвать так:

```
P(1, 2);
```

Программирование вспомогательных алгоритмов. Описание вспомогательных алгоритмов вводится в исходный текст программы. Во многих языках программирования до вызова подпрограммы требуется ее описание.

Описание подпрограммы включает *заголовок*, *тело* и *конец*. В заголовке подпрограммы задается имя подпрограммы и дается описание формальных параметров. Для функции задается также тип возвращенного значения:

```
function Square(x: Integer) : Integer;
begin
  Square := x*x;
end;
```

На языке Pascal нет специального оператора конца. Тело функции размещено между операторами begin и end.

> Значение, возвращенное функцией, должно быть присвоено переменной с тем же названием, что и сама функция.

Эту переменную можно использовать в теле функции в операторе присваивания только слева.

Нижеследующая функция вычисляет факториал числа (*факториалом* натурального числа *n* является произведение натуральных чисел от *1* до *n* включительно, записывается $n!=1\cdot2\cdot3\ \ldots\ \cdot n$).

```
function Fact(n : Integer) : Integer;
var
   i : Integer;
   res : Integer;
begin
   res := 1;
   for i := 1 to n do res := res * i;
   Fact := res;
end;
```

В описании процедуры нет необходимости вычислять возвращенное значение. Ниже описана процедура, выдающая на печать сумму двух целых чисел.

```
procedure PrintSum(x, y: Integer);
begin
      WriteLn(x+y);
end;
```

Как правило, описание процедуры должно быть дано до первого обращения к ней (так удобнее транслятору). Но по некоторым причинам это может быть неудобно прораммисту. В некоторых языках программирования есть возможность размещать описание процедуры после ее первого вызова, но в таком случае требуется дать ее *простое описание* до ее использования.

Для этого на языке Pascal повторяется заголовок процедуры. Отсутствие ключевого слова **begin** показывает, что это не *описание*, а всего лишь *объявление* процедуры.

```
procedure PrintSum(x, y: integer);
```

Заранее объявленная процедура дает транслятору информацию о параметрах процедуры. Это, в свою очередь, дает возможность правильно обработать вызов процедуры.

Стандартные математические функции языка Turbo Pascal

Функция	Назначение	Аргумент	Вывод
`Abs(X)`	Абсолютное значение X	`Real` или `Integer`	`Real`, или `Integer`
`Cos(X)`	Косинус угла X	`Real` или `Integer` (в радианах)	`Real`
`Exp(X)`	e^x, e = 2.71828...	`Real` или `Integer`	`Real`
`Ln(X)`	Натуральный логарифм X, X > 0	`Real` или `Integer`	`Real`
`Round(X)`	Самое близкое целое число к числу X	`Real`	`Integer`
`Sin(X)`	Синус угла X	`Real` или `Integer` (в радианах)	`Real`
`Sqr(X)`	Квадрат X	`Real` или `Integer`	`Real` или `Integer`
`Sqrt(X)`	Квадратный корень из X X > 0.0	`Real` или `Integer`	`Real`
`Trunc(X)`	Целая часть числа X	`Real`	`Integer`

Нижеследующая программа Возрастание меняет между собой значения переменных a, b, c таким образом, чтобы они расположились в возрастающем порядке ($a \le b \le c$).

```
program Vozrastanie;
var
    a, b, c : Integer;
procedure Swap(var x, y : Integer);
var
    t : Integer;
begin
    t := x; x := y; y := t;
end;
begin;
    WriteLn('Введите три числа ');
    ReadLn(a, b, c);
    if a > b then Swap(a, b);
    if b > c then Swap(b, c);
    if a > b then Swap(a, b);
    WriteLn(a:5, b:5, c:5);
    ReadLn;
end.
```

Обратите внимание на процедуру Swap. Эта процедура меняет местами значения переменных. Такой перестановкой очень часто пользуются в программировании.

Остановка выполнения подпрограммы и возврат в вызывающую программу происходят в случае перехода управления к его последнему оператору. Это не всегда удобно. Для того, чтобы преждевременно закончить выполнение подпрограммы и вернуться в вызывающую программу, используют оператор exit.

1. Каково преимущество использования подпрограмм?
2. Какие существуют виды подпрограмм?
3. Чем отличается функция от процедуры?
4. С клавиатуры введено натуральное число *n*. Напишите функцию, определяющую, делится или не делится нацело это число на 3.

1.13. РАБОТА С ФАЙЛАМИ

Все программы, с которыми мы имели дело до сих пор, представляли собой интерактивные программы. *Интерактивная программа* считывает все входные данные с клавиатуры, а весь вывод осуществляется на экран. Интерактивные ввод и вывод хороши для программ, манипулирующих небольшими количествами данных, однако, подобный подход малоэффективен для программ, предназначенных для обработки обширных объемов информации. В последнем случае, чтобы выйти из положения, можно для ввода и вывода использовать файлы. Pascal работает с двумя видами информационных файлов: *текстовыми* и *двоичными* файлами.

Текстовой файл представляет собой набор отдельных символов, которые хранятся на диске под одним именем. Все данные, обрабатываемые программой, прежде чем запускать эту программу, можно сохранить в текстовом файле. После этого следует преобразовать программу таким образом, чтобы она считывала нужные ей данные не с клавиатуры, а из текстового файла.

> Текстовой файл представляет собой набор отдельных символов, которые хранятся на диске под одним именем.

Входные и выходные файлы. *Входной файл* – это файл, содержащий входные данные программы. Входной файл может представлять собой текстовой файл или двоичный файл. Одно из преимуществ использования входного файла заключается в том, что с применением текстового редактора в нем можно исправить все ошибки, прежде чем эти данные будут обработаны программой. Второе преимущество состоит в том, что файл с входной информацией может считываться программой много раз. Эта возможность облегчает отладку, поскольку программа при каждом запуске может считывать свои данные каждый раз из того же файла. А в интерактивной программе приходится каждый раз заново вводить с клавиатуры данные.

> Входной файл – это файл, содержащий входные данные программы.

В то же время можно преобразовать программу таким образом, чтобы вычисленные результаты она выводила не на экран, а в текстовой файл. Это позволит иметь на диске электронную версию результатов работы программы. Этот файл затем может быть выведен на печать, либо даже использован в качестве входного файла другой программой. Файл, содержащий результаты работы программы, называется *выходным файлом*.

> Выходной файл – это файл, содержащий результаты работы программы.

Для того, чтобы работать с текстовыми файлами, надо выполнить пять операций.

1. **Описание файла.** Как и все другие объекты программы, текстовой файл перед использованием следует описать в разделе var с использованием ключевого слова text. Например,

```
var f1, f2: text;
```

указывает на то, что f1 и f2 являются текстовыми файлами. Их также называют *текстовыми переменными*.

2. **Сопоставление файловой переменной файлу на диске.** Для того, чтобы иметь доступ к файлу, сохраненному на диске, необходимо указать имя этого файла, а также указать в какой папке (каталоге) он содержится. То есть надо сообщить полное имя файла. Для этого на Pascal используется процедура **Assign**. Например, следующий оператор вызова процедуры связывает файловую переменную f1 с файлом in.txt, содержащимся на диске C в папке ALTAY.

```
Assign(f1, 'c:\ALTAY\in.txt');
```

Если указано только имя файла, то Turbo Pascal принимает во внимание, что файл находится в одном каталоге с программой. Например, процедура

```
Assign(f2, 'out.txt');
```

связывает файл out.txt, находящийся в папке программы с файловой переменной f2. Процедура **Assign** является единственной, которая оперирует фактическим именем файла на диске. Все другие процедуры и функции используют файловую переменную. По этой причине программисты называют f1 внутренним именем файла, а c:\ALTAY\in.txt – его внешним именем.

3. **Открытие файла для чтения или для записи.** Прежде чем программа сможет манипулировать текстовым файлом, этот файл должен быть подготовлен к вводу или выводу – иными словами, открыт. Текстовой файл не может быть открыт для ввода и для вывода одновременно. То есть, если идет процесс считывания данных из текстового файла, вы не сможете осуществить запись результатов в тот же файл. Открытие текстового файла на Pascal производится операторами **Reset** и **Rewrite**.

```
Reset(f1);
```

Данный оператор подготавливает файл, ассоциированный с файловой переменной f1, к вводу в программу, перемещая при этом указатель текущей позиции к началу файла. *Указатель текущей позиции* указывает на символ в файле, который будет обработан следующим. Вызов процедуры **Reset** должен иметь место до того как из файла будет что-либо считано.

```
Rewrite(f2);
```

Эта процедура подготавливает файл, связанный с файловой переменной f2, для приема данных, являющихся результатом работы программы. Если на диске не будет файла с таким именем, то он будет создан. Если файловая переменная ссылается на существующий файл, то все его содержимое будет удалено при открытии для записи, и указатель текущей позиции будет переведен на начало файла.

4. **Запись данных в файл или чтение данных из файла.** Для того, чтобы прочитать данные из текстового файла, используют процедуру **ReadLn**. Эта процедура обрабатывает каждую строку текстового файла также, как и данные, введенные с клавиатуры. Процедура

```
ReadLn(f1, n);
```

читает данные с входного файла f1 и записывает переменной n. Для того, чтобы считать с файла два последовательно идущих элемента и присвоить их переменным a и b, можно использовать процедуру

```
Read(f1, a, b);
```

Как вы уже знаете, для вывода данных на экран используются процедуры **WriteLn** и **Write**. Они применяются также и для записи данных в текстовой файл. Процедура

```
WriteLn(f2, n);
```

записывает значение переменной n в файл f2. То есть данные будут записаны в файл с новой строки. Процедура

```
Write(f2, a, b);
```

записывает в файл f2 числа a и b и новая запись будет производиться в той же строке.

5. **Закрытие файла.** После окончания работы с входным или выходным файлом необходимо его закрыть, то есть прекратить связь файла на диске с программой. Другими словами, открытые файлы должны быть закрыты. Для этого в Pascal имеется процедура **Close**.

```
Close(f1);
Close(f2);
```

Запись **Close** (f1, f2) считается неверной – *каждый файл надо закрыть в отдельности.*

Рассмотрим два примера работы с файлами. В первом – два числа записываются в файл, во втором – они же читаются из файла и выводятся на экран.

Пример 1.

```
program P1;
var
   f: text;       {Объявление файла}
   n: Integer;
begin
   Assign(f,'file1.txt');
  {Файл f  связывается с файлом
                 file1.txt, расположенном на диске}
   Rewrite(f);    {Открывается файл f для записи }
   n := 7;
   WriteLn(f, n); {Значение переменной n
                          записывается в файл}
   WriteLn(f, 5); {Число 5 записывается в файл}
   Close(f);    {Закрытие файла}
end.
```

Пример 2.

```
program P2;
var
   f: text;      {Объявление файла}
   n: Integer;
begin
   Assign(f,' file1.txt'); {Файл f  связывается с
          файлом  file1.txt, расположенном на диске}
Reset(f);        {Открывается файл f для чтения}
```

```
ReadLn(f, n);  {Первое число заносится в переменную n}
WriteLn(n);    {Значение переменной n (число 7)
                выводится на экран}

ReadLn(f, n);{Второе число заносится в переменную n}
WriteLn(n);    {Значение переменной n (число 5)
                выводится на экран}
Close(f);      {Закрытие файла}
end.
```

А теперь ознакомимся с часто используемыми при работе с файлами функциями.

Добавление информации в текстовые файлы. Для текстовых файлов предусмотрена возможность добавления информации в их конец. Для этого файл следует открывать не процедурой **Rewrite**, а процедурой **Append**.

```
Append(<Файловая переменная>);
```

После вызова данной процедуры файл открывается для записи, но информация в нем не стирается, как в случае использования процедуры **Rewrite**. Указатель текущего элемента перемещается в конец файла, соответственно выводимая информация дописывается к открытому файлу.

В качестве примера допишем к файлу file1.txt, созданному программой P1, новую запись:

```
program P3;
var
   f: text;
begin
   Assign(f,'file1.txt');

   Append(f);
   WriteLn(f, 9);

   Close(f);
end.
```

В нижеследующей таблице дана краткая информация еще о некоторых фунциях, используемых при работе с файлами.

Функция	Назначение
Eof (<Файловая переменная>)	Проверяет конец файла End-of-file (конец файла) – возвращает истинное значение, если достигнут конец файла, или ложное, если конец файла не достигнут.
Erase (<Файловая переменная>)	Удаляет файл, соответствующий указанной файловой переменной.
Rename (<Файловая переменная>, <Новое имя файла>)	Переименовывает файл.
MkDir (<Имя каталога>)	Создает новый каталог.
RmDir (<Имя каталога>)	Удаляет каталог. При этом удаление возможно для каталогов, которые не содержат файлы и подкаталоги.

1. Каково преимущество ввода данных из файла?
2. Какие виды файлов имеются в Turbo Pascal?
3. Каким образом можно использовать текстовые файлы в Turbo Pascal и какие операторы для этого нужны?
4. Что такое файловая переменная?
5. Объясните сущность внешнего и внутреннего имени файла.

1.14. ПРАКТИКУМ

ОПЕРАТОРЫ

1. Какое значение получит переменная s после выполнения последовательности команд?

а)
```
s := 7; s := 23;
```
б)
```
s := 1; s := s + 3;
```
в)
```
a := 2; b := 5; b := b - a; s := b + a;
```
г)
```
s := 0; k := 30; d := k - 5; k := 2*d;
s := k - 100;
```

2. Какие значения получат переменные x и y после выполнения следующих операторов? Поменяют ли эти переменные свои значения?

```
x := 2;
y := 9;
x := y;
y := x;
```

3. Какие значения получат переменные a, b, c после выполнения следующих операторов (a = 1, b = 2, c = 3) ?

```
a := b;
b := c;
c := a;
```

4. Что будет выведено на экран после выполнения следующих операторов?

```
a := 4;
Write(a);
Write('a');
```

5. Что будет выведено на экран после выполнения данной программы?

```
program Task5;
var
    a, b, c : Integer;
begin
    Write(1);
    Write(2, 3);
    WriteLn(4);
    Write(5);
    WriteLn(6, 7);
    WriteLn;
    Write(8);
    ReadLn;
end.
```

6. Что будет выведено на экране после выполнения данной программы, если с клавиатуры ввести числа 1, 2, 3?

```
program Task6;
var
    a, b, c : Integer;
begin
    WriteLn('Введите три целых числа');
    ReadLn(a, b, a);
    c = a + b;
    Write('a+b=', c);
    ReadLn;
end.
```

7. Какие числа и в каком порядке надо ввести с клавиатуры, чтобы на экране оказалось выведенным число 123?

```
Read(a, b, c);
Write(c, b, a);
```

8. Найдите и устраните ошибки в программе.

```
proqram Task8;
var
    a; b; c : Integer;

begin
    WriteLn(' Введите число ', a);
    ReadLn(a)
    b = 5;
    c = ab;
    WriteLn(a ' * ' b ' = ', s);
    ReadLn(a);
end.
```

9. Составьте программу, требующую ввести целое число и выводящую на экран само число, его квадрат и куб. Результаты программы должны быть приблизительно такими:

```
Введите число.
4
4**2 = 16
4**3 = 64
```

10. Нижеследующая программа требует от пользователя количество дней в текущем месяце и текущее число и выводит на экран число дней, оставшихся до конца текущего месяца. В программе переменная kd показывает количество дней в текущем месяце, переменная td – текущее

число, а переменная od – число оставшихся дней до конца месяца. Проверьте, как работает программа.

```
program Task10;
var
   kd, td, od : Integer;
begin
   WriteLn('Сколько дней в текущем месяце?');
   ReadLn(kd);

 WriteLn('Какое сегодня число?');
 ReadLn(td);
 od := kd - td;
 WriteLn('До конца этого месяца осталось ', od,
         'дней');
 ReadLn;
end.
```

11. Напишите программу, запрашивающую год вашего рождения, текущий год и выдающую на экран ваш возраст. Результат работы программы должен быть приблизительно такой:

```
Год твоего рождения?
1995
Какой сейчас год?
2009
Тебе 14 лет.
```

12. При каких значенях x будут верны следующие равенства?

а) `x div 5 = 8`
б) `50 div x = 7`
в) `50 mod x = 7`
г) `x div 5 = x mod 5`
д) `20 div x = 20 mod x`

13. Допустим, в переменной S хранится пятизначное число, переменная a показывает число десятков тысяч в этом числе, переменная b – число

Примечание: 10. – задание с решением. Обратите внимание на решение и проанализируйте его.

тысяч, переменная c – число сотен, переменная d – число десятков, а переменная e – число единиц. Укажите в следующей таблице соответствие между столбцами таблицы.

Действие	Переменная
`s div 100 mod 10`	a
`s mod 10`	b
`s div 10 mod 10`	c
`s div 10000`	d
`s mod 100 div 10`	e

14. Составьте программу, находящую цифры произвольного четырехзначного числа. Внизу дан примерный диалог между пользователем и компьютером. Число, введенное пользователем, выделено жирным шрифтом.

```
Введите четырехзначное число.
4523
Число тысяч 4
Число сотен 5
Число десятков 2
Число единиц 3
```

УСЛОВИЕ

15. Какое значение примет переменная p после выполнения следующих операторов?

```
q := -1;
p := 1;
if (p > 0) and (q > 0) then
   p := 2
else
   if (p < 0) and (q < 0) then
      p := 3
   else
      p := 4;
```

16. Какие значения примут переменные p и q после выполнения следующих операторов?

```
q := false;
p := true;
p := p and q;
q := q or false;
q := (not q) or p;
```

17. Какое значение примет переменная c после выполнения следующих операторов?

```
a := 8;
a := a + 2;
b := a - 1;
c := a + b div 2;
```

18. Найдите и устраните ошибки в данном фрагменте программы.

```
if a >= 10 and a <= 99 then
   WriteLn(a ' двузначное число');
   WriteLn('его квадрат =', sqrt(a));
else (a, ' число не двузначное');
```

19. Какие значения примут переменные v, t, u после выполнения программы, если с клавиатуры ввести числа 3, 5, 9?

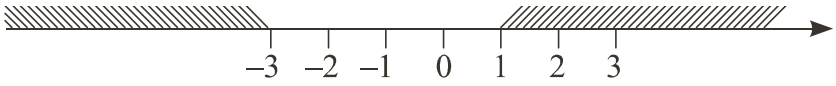
```
program Task19;
var
   a, b, c, v, t, u : Integer;

begin
   WriteLn('Введите три числа');
   ReadLn(a, b, c);
   v := 1; t := 0; u := 0;
   if a mod 3 = 0 then begin
      v := v * a;
      t := t + 1;
      u := u + 1;
   end;
   if b mod 3 = 0 then begin
      v := v * b;
     t := t + 1;
      u := u + b;
   end;
   if c mod 3 = 0 then begin
      v := v * c;
      t := t + 1;
      u := u + c;
```

```
        end;
        WriteLn('v=', v, ' t =', t, ' u=', u);
        ReadLn;
    end.
```

20. Используя логические операции and, or, not, напишите следующие условия на языке программирования:

а) x принадлежит отрезку [-3, 2] ;

б) число a находится в заштрихованной области.



21. Дано уравнение $ax + b = 0$. Составьте программу, находящую корень данного уравнения или выдающую сообщение об отсутствии корней.

22. Напишите программу, относящую пользователя, в зависимости от его возраста, к разной возрастной категории:

- До 13 лет – детский возраст
- От 14 до 24 – молодой возраст
- От 25 до 59 – зрелый возраст
- Свыше 60 – старческий возраст

23. Дано квадратное уравнение $ax^2 + bx + c = 0$. Данная программа находит решение этого уравнения при заданных значениях a, b, c, или же дает сообщение о том, что данное уравнение не имеет решения. Проверьте выполнение программы.

```
program KvadratUravn;
var a, b, c  : Real;
    D : Real;
    x1, x2 : Real;

begin
   Write ('Введите коэффициенты a, b, c: ');
```

```
  ReadLn(a, b, c);
  if (a = 0) and (b = 0) and (c = 0)
     then begin
        Write ('Все коэффициенты равны 0');
        WriteLn ('X - произвольное число')
     end
     else
     if (a = 0) and (b <> 0)
      then WriteLn('Уравнение имеет один корень
                    x=', (-c/b):6:2)
      else
       begin
          D := b*b - 4*a*c;
          if D > 0
             then begin
               x1:=(-b+sqrt(D))/(2*a);
               x2:=(-b-sqrt(D))/(2*a);
               WriteLn('x1=', x1:6:2,'x2=',
                        x2:6:2)
             end
             else
               if D = 0
                  then begin
                    x1: = -b/(2*a);
                     WriteLn('Корни одинаковые');
                     WriteLn('x1,2=',x1:6:2);
                   end
             else WriteLn('Нет действительных
                           корней');
       end;
end.
```

ЦИКЛЫ

24. Выведите на экран квадраты натуральных чисел от 1 до 20.

25. Напишите программу, выводящую на экран таблицу умножения на 4.

26. Выведите на экран натуральные числа, нацело делящиеся на 4 в интервале от 1 до 100.

27. Даны натуральные числа n и m. Напишите программу, вычисляющую произведение этих чисел, не используя операцию умножения.

28. Дано натуральное число n. Проверьте программу, результатом которой будет представление данного числа в следующем виде:

```
6
6 6
6 6 6
6 6 6 6
6 6 6 6 6
6 6 6 6 6 6
```

(показано для n=6).

```
program Task28;
var i, j, n  : Integer;

begin
  ReadLn(n);
  for i := 1 to n do begin
      for j := 1 to i do Write(n, ' ');
      WriteLn;
  end;
end.
```

29. Напишите программу, выводящую целое число в таком виде:

```
6 6 6 6 6 6
6 6 6 6 6
6 6 6 6
6 6 6
6 6
6
```

30. Напишите программу, выводящую целое число в таком виде:

```
0
1 0
2 1 0
3 2 1 0
4 3 2 1 0
5 4 3 2 1 0
```

МАССИВЫ

31. Какие значения получат переменные p и q при выполнении данной последовательности операторов?

```
for i := 1 to 10 do
    for j := 1  to 5 do
      A[i,j]  := i*j;
p := 0;
q := 0;
m := 2;
n := 5;
for k := 1 to 5 do begin
    p: = p + A[m,k];
    q := q + A[n,k];
end;
```

32. Данная программа выводит на экран первые 5 элементов массива X, состоящего из 10 элементов. Проверьте работу программы.

```
program Print;
var i : Integer;
    X : array[1..10] of Integer;

begin
   for i := 1 to 5 do
       Write(X[i], ' ');
   WriteLn;
 end.
```

33. Данная программа вычисляет и выдает на экран число положительных и отрицательных элементов одномерного массива. Проверьте работу программы.

```
program Task33;

 const Nmax = 100;
 type  TArr = array[1..Nmax] of integer;
 var   A : Tarr;

 procedure Solve;
   var i, n, p : Integer;
   begin
     p := 0;
     ReadLn(n);
      for i := 1 to n do Read(A[i]);
     for i := 1 to n do
        if A[i] >= 0 then Inc(p);

     WriteLn('Число положительных элементов ',
             p);
     WriteLn('Число отрицательных элементов ',
             n - p);
   end;
```

```
begin
  Solve;
end.
```

34. Напишите программу, определяющую номер последнего отрицательного элемента одномерного массива.

35. Дан целочисленный массив. Найдите сумму элементов массива, заключенных между числами q и t, введенных с клавиатуры.

36. Заполните массив A размером n x m следующим образом:

```
1  2  3  4  5  6  7
8  9  10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
```

РАБОТА СО СТРОКАМИ

37. Данная программа выводит на экран порядковый номер символов ASCII и сами символы. В программе использована переменная k как счетчик в задании на одной строке 15 символов. Проверьте работу программы.

```
program Task37;

var i, k : Integer;

begin
   WriteLn('Вывод на экран порядковых номеров
           символов - значение переменной
           i и самих символов');
   k := 0;
   for i := 1 to 255 do begin
     Write(i : 4, ' Символ ', Chr(i));
     k :=k + 1;
     if k = 15 then begin
        WriteLn;
```

```
            k := 0;
        end;
    end;
end.
```

38. В соответствии с правилами набора текста, после запятой в тексте всегда ставится пробел. Эта программа находит в тексте такого типа ошибки и исправляет их. Проверьте работу программы.

```
program Task38;

var i : Integer;
    s : string;

begin
   WriteLn('Введите текст');
   ReadLn(s);
   i := 1;
   while i < Length(s) do begin
     if (s[ i] = ',') and not(s[ i+1] = ' ')
          then Insert(' ', s, i+1);
     i :=i + 1;
   end;
   WriteLn(s);
   ReadLn;
end.
```

39. Измените данную выше программу таким образом, чтобы она проверяла наличие пробела после таких знаков как “!” , “?”, “.”, и если есть ошибка, то исправляла ее.

40. Напишите программу, находящую самое короткое и самое длинное слово в строке.

41. Напишите программу, вычисляющую, сколько раз встречается та или иная буква в строке (например буква “а”).

ФАЙЛЫ

42. Задан текстовой файл test.txt на диске:

```
123 17 25
256 80 5
89 56 234
123 123 123
81 11 11 11
```

Что будет выведено на экран после выполнения программы?

```
program Task42;

var f : text;
    s : string;
    n, m : Integer;
    c, z : Char;

begin
   Assign(f, 'test.txt');
   Reset(f);
   ReadLn(f, s);
   ReadLn(f, n);
   Read(f, m);
   ReadLn(f, c);
   Read(f, z);
   Close(f);
   WriteLn('s=', s);
   WriteLn('n=', n);
   WriteLn('m=', m);
   WriteLn('c=', c);
   WriteLn('z=', z);
   ReadLn;
end.
```

43. Допустим, задан текстовой файл f и строка st. Данная программа ищет в строках файла f значение переменной s и найдя, записывает строку, где найдено это значение, в новый файл g. Проверьте работу программы.

```
program Task43;

var f, g : text;
    s, st : string;

begin
WriteLn('Введите строку ');
ReadLn(s);

Assign(f, 'test.txt');
Assign(g, 'test2.txt');
Reset(f);
Rewrite(g);
while not eof(f) do begin
   ReadLn(f, st);
   if pos(s, st) <> 0 then WriteLn(g, st);
end;
close(f);
close(g);
ReadLn;
end.
```

44. Задан текстовой файл. Запишите самую короткую строку этого файла на новый файл.

45. Напишите программу, определяющую самую длинную строку в текстовом файле.

46. Задан текстовой файл, состоящий из списка учащихся. На каждой строке приведены имя и фамилия одного ученика. Напишите программу, считывающую этот файл и выводящую его на экран.

2

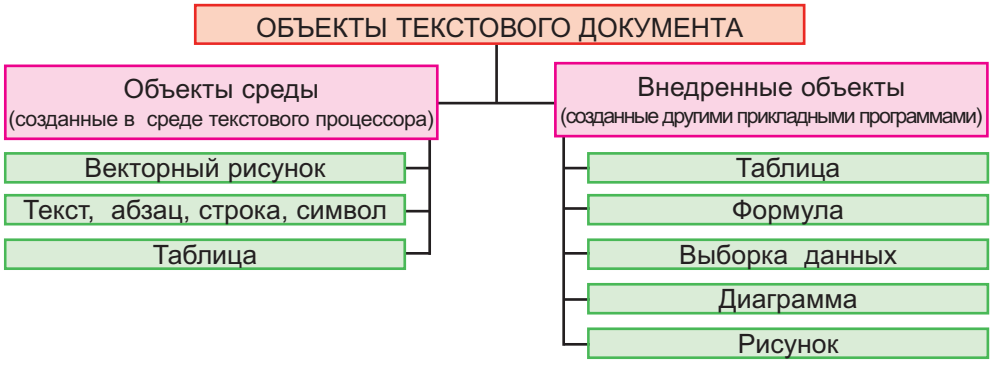
ЭЛЕКТРОННЫЙ ДОКУМЕНТ

2.1. ТЕКСТОВЫЙ ДОКУМЕНТ И ЕГО ОБЪЕКТЫ

Одной из самых распространенных областей применения компьютера является подготовка текстовых документов. Зачастую текст в процессе его создания многократно изменяется. Если текст находится на бумаге, то в нем остаются следы изменений. Чтобы избавиться от этой проблемы, на компьютере существуют специальные программы – *текстовые процессоры*, позволяющие работать с текстом.

Текст, созданный при помощи текстового процессора в совокупности с включенными в него нетекстовыми элементами (рисунками, таблицами) называют *документом*.

Прежде чем говорить о текстовом процессоре, следует иметь представление об используемых в нем объектах. Познакомимся с классификацией объектов текстового документа.



Как видно из схемы, объекты текстового документа сгруппированы по двум уровням. Объекты первого уровня подразделяются на две группы в зависимости от среды, в которой они были созданы: *это объекты самой среды и внедренные объекты.*

К *объектам среды* относятся такие объекты, для создания, редактирования и форматирования которых не требуется вызова иных программ.

К *внедренным объектам* относятся такие, которые создаются в другой прикладной среде.

Говоря о среде внедрения, рассмотрим такое сравнение. Квартира является частью среды обитания городского жителя, и все в ней предназначено для него – мебель, посуда, одежда, обувь, светильники и другие бытовые приборы… Однако до возникновения городов человека окружала живая природа. И сейчас многие горожане, тоскуя по природе, держат в своих квартирах растения, рыбок и пр. Но ведь цветок не вырастить, скажем, на полу или на столе – для нормального существования ему требуется соответствующая среда, то есть земля. Значит, для того, чтобы вырастить растение в квартире, нужно приобрести цветочный горшок, наполнить его землей и посадить туда растение. Таким образом, в данном случае городская квартира – это среда, растение – внесенный в эту среду объект, а земля – фрагмент иной среды, обеспечивающий потребности объекта.

Основанием классификации на втором уровне является тип объекта. Рассмотрим список объектов, создаваемых текстовым процессором. Сначала рассмотрим объекты подкласса “текст”. В тексте можно выделить следующие объекты: символ, слово, строка, абзац, страница.

Основными параметрами символа как объекта являются *вид начертания, кегль, цвет*.

Виды начертаний символа. На компьютере используются в основном четыре вида начертаний символа:

- Нормальное
- *Курсивное* (Italic)
- **Полужирное** (Bold)
- ***Полужирное курсивное*** (Italic Bold)

Курсив служит в тексте средством *мягкого акцентирования.* Он привлекает внимание читателя к чему-либо, отмечает особое отношение к нему.

Полужирный шрифт служит средством *сильного акцентирования*. Он может, например, указывать на непререкаемую значимость чего-либо. Кроме того, им отмечают заголовки в тексте.

UNICODE

Любой компьютер может работать только с цифрами. И для того, чтобы компьютер мог хранить в своей памяти буквы и другие символы, каждому символу ставится в соответствие некоторое число. Когда-то существовали сотни схем таких кодировок символов, однако ни одна из них не могла охватить все необходимые символы. Не было системы кодирования, охватывающей все буквы, знаки препинания и технические символы какого-то одного естественного языка.

Назрела необходимость разработки новой схемы кодирования, включающей все алфавиты современной цивилизации. В конце концов, был создан Unicode (произносится как «юникод»).

Unicode – сложный объект. В его создании принимали участие лингвисты и программисты со всех уголков нашей планеты. В основу его положен принцип кодирования каждого символа 16-битным числом. Это значит, что на каждый символ отводится по два байта. По такому принципу можно закодировать 65 536 знаков или символов. В системе кодирования Unicode каждому символу присвоен уникальный код, не зависящий от платформы, программы или языка.

Кегль – это размер шрифта. Традиционно кегль измеряется в пунктах (пт). Один пункт равен 0,35 мм. Существует ряд стандартных кеглей. Вот некоторые из них:

12 Кегль – это размер шрифта.
18 Кегль
24 Кегль
36 Кегль
48 Кегль
60 Кегль
72 Кегль

Цвет. Большинство объектов в средах прикладных программ являются цветными. Символы, линии и прочие объекты, имеющие однородную структуру, окрашиваются только в один цвет. В объектах со сложной структурой (например, автофигура, клетка, поле и т.д.) контур и фон могут быть разного цвета. 

Помимо основных, символ может иметь следующие дополнительные параметры: эффект, смещение, кернинг.

Эффект. Этот параметр определяет внешний вид знака на экране и в распечатке документа. В таблице приведены наиболее популярные виды эффектов.

Эффект	Подчеркнутый [Underline]
Эффект	Зачеркнутый [Strikethrough]
Эффект	С тенью [Shadow]
Эффект	Контур [Outline]
Эффект	Приподнятый [Emboss]
Эффект	Утопленный [Engrave]
Эффект	Малые прописные [Small caps]
ЭФФЕКТ	Все прописные [All caps]

Смещение определяет положение символа относительно базовой линии строки. Оно бывает двух типов*: нижнее и верхнее.*

Верхнее смещение относительно базовой линии

Нижнее смещение относительно базовой линии

Кернинг – это межсимвольный интервал. Различают три основных значения данного параметра: нормальный, разреженный и уплотненный.

Интервал	Уплотненный [Condensed]
Интервал	Нормальный [Normal]
Интервал	Разреженный [Expanded]

Обычно нормальное значение кернинга для данного шрифта задается автоматически. При необходимости его можно сменить на другое.

Дефис и тире

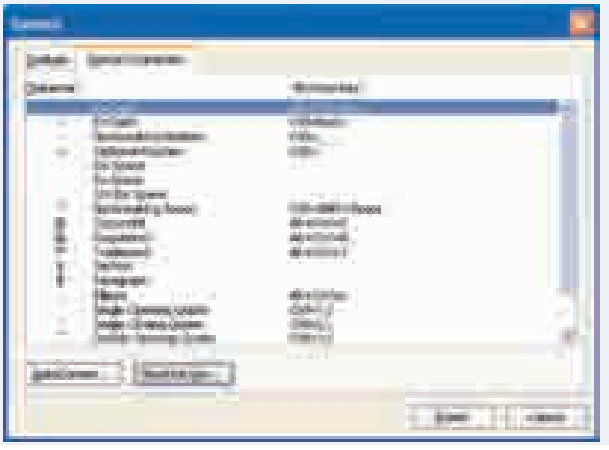
Некоторые символы имеют особое значение в тексте. Среди таких символов – *дефис* и *тире*.

Дефис – это символ, соединяющий либо две части слова, либо два слова, составляющие одно целое. Справа и слева от дефиса не оставляют пробелов. Если слово с дефисом не помещается на строке, текстовый процессор сделает в нем перенос как раз на месте дефиса. В этом случае дефис будет одновременно выполнять и роль знака переноса. Однако если речь идет о записи даты, где дефис стоит между числами годов (например, “1918-1920”), то подобную надпись переносить нежелательно. На случай таких нежелательных переносов в текстовом процессоре имеется особый символ – *неразрывный дефис* [nonbreaking hyphen].

Тире – это знак, разделяющий две части предложения. В отличие от дефиса, тире отделяется от соседних слов пробелами. В печатной продукции тире обозначается длинной горизонтальной чертой, а дефис – более короткой. Однако с клавиатуры компьютера и знак минуса, и тире, и дефис вводятся одной и той же клавишей. Поэтому для обозначения тире существует специальный символ – *длинное тире* [em dash].

Упражнение

1. Откройте какой-нибудь текстовый документ.
2. Прочтите текст. Определите, правильно ли в нем расставлены тире и дефисы.
3. Исправьте обнаруженные ошибки.
4. Для вставки длинного тире и неразрывного дефиса выполните команду Insert⇨Symbol и выберите нужный символ во вкладке «Специальные знаки» Special Characters.



1. Какие объекты присущи тексту?
2. Чем характеризуется символьный объект?
3. Перечислите виды начертаний символа.
4. Что такое эффекты? Раскройте их значение на примерах.
5. Что такое смещение и кернинг? Раскройте их значение на примерах.
6. Наберите какой-либо текст в текстовом процессоре и примените к его различным символам параметры, изученные на уроке.

2.2. СОЗДАНИЕ ТЕКСТОВОГО ДОКУМЕНТА

Процесс создания документа на компьютере включает следующие этапы: *ввод (набор) текста, редактирование, форматирование, печать.*

Прежде чем рассмотреть каждый из этих этапов, вкратце ознакомимся с некоторыми объектами текстового документа – *словом, строкой и абзацем.*

Из символов создаются слова.

Слово – это последовательность символов (букв, цифр, специальных знаков) без пробелов. Единственным самостоятельным параметром слова является количество символов в нем.

Строка состоит из слов, которые отделяются друг от друга пробелами. Без пробела за словом в строке следуют только знаки препинания, такие, как запятая, точка, точка с запятой, двоеточие, скобка. Объект "строка" наследует все параметры объекта "слово". Самостоятельным параметром строки является количество слов.

При форматировании все правила, которым подчиняются символы и слова, касаются и строк.

Абзац. Из объектов-строк образуются объекты-абзацы. Абзац вводится при помощи клавиши "Enter". Признаком окончания абзаца является символ перевода строки ¶ (в обычном режиме этот знак на экране не отражается).

Ввод текста. Ввод (набор) текста обычно производится с клавиатуры. Роль бумаги при этом выполняет экран монитора. Место для введения очередного символа указывается на экране мерцающей вертикальной чертой – *курсором*.

Правила, которые нужно выполнять при наборе текста:

1. Все знаки препинания, кроме тире, вводятся непосредственно после последней буквы слова. После знака препинания вводится пробел (для этого нужно нажать клавишу "Spacebar"). Тире отделяется пробелами с обеих сторон.
2. При наборе текста можно исправлять допущенные ошибки. Если при этом символ, который нужно заменить, находится справа от курсора, то используется клавиша "Delete" ("Del"), если слева – то клавиша "Backspace".
3. При наборе текста на компьютере не нужно обращать внимание на переход со строки на строку – достигнув конца одной строки, курсор автоматически переместится в начало следующей.
4. Для того, чтобы начать новый абзац, нажмите клавишу "Enter".

Редактирование текста. Редактирование – это изменение содержания документа. К редактированию относятся следующие операции:

- набор текста;
- исправление ошибок;
- копирование, перемещение и удаление частей текста;
- дополнение текста рисунками, таблицами и другими объектами.

Форматирование текста. Внешний вид текста очень важен для того, чтобы информацию, содержащуюся в нем можно было легко и быстро довести до сведения читателя. Для этого некоторые слова или фрагменты текста выделяют тем или иным образом.

> Форматирование – это изменение внешнего вида документа или отдельных его частей с целью облегчения его восприятия.

Слово *форматирование* происходит от слова “форма”. Форматировать – значит придавать форму чему-либо. Форматирование как операция может включать в себя ряд тех или иных способов изменения документа:

- изменение свойств символа;
- изменение свойств абзаца;
- оформление заголовков и подзаголовков;
- преобразование текста в список;
- преобразование текста в таблицу;
- вставка колонтитулов и номеров страниц, и т.д.

Чтобы изменить свойства символов в том или ином фрагменте текста, надо выделить этот фрагмент, а затем изменить необходимые параметры либо с помощью кнопок панели инструментов, либо открыв диалоговое окно “Font” (“Шрифт”).

Упражнения

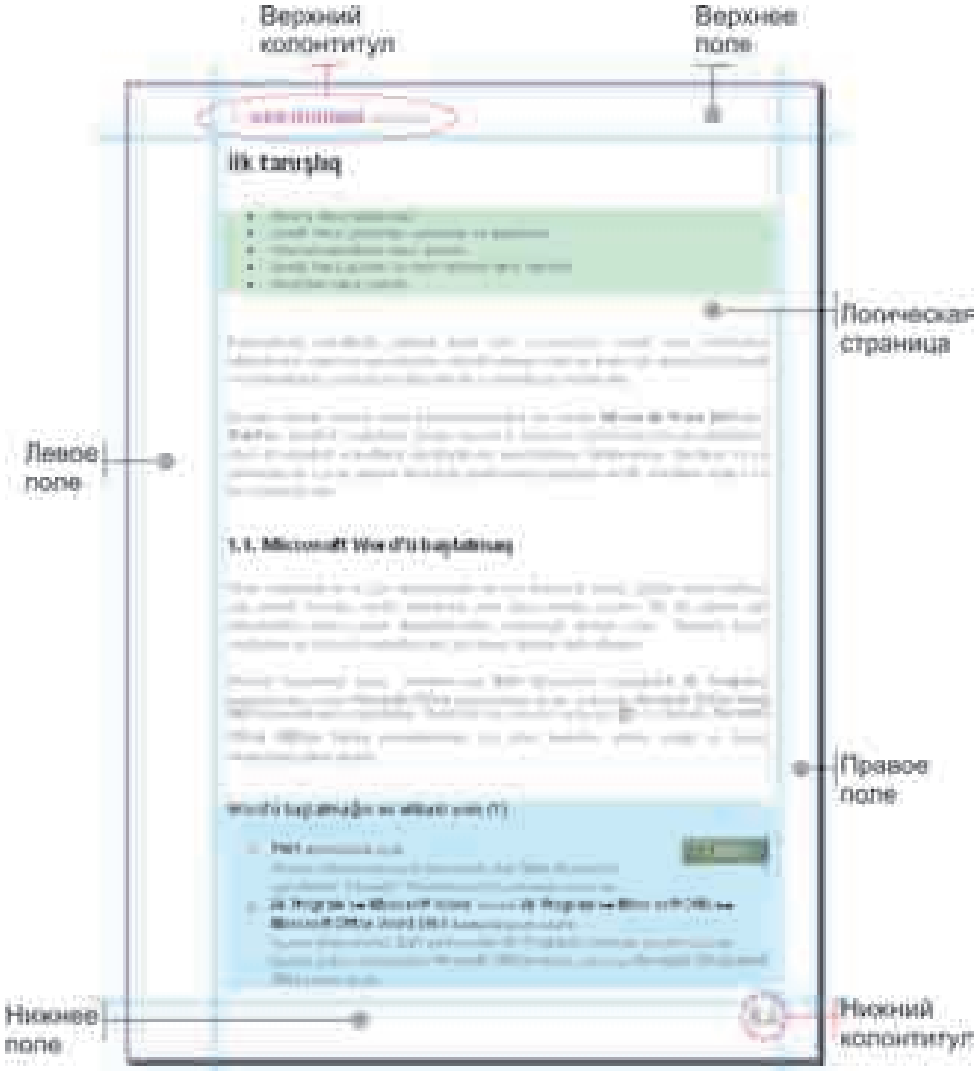
1. Откройте какой-нибудь текстовый документ.
2. Выделите произвольный фрагмент текста и измените шрифт в нем на следующий: Courier New, жирный, 16 пт, красный, все прописные.
3. Выделите другой фрагмент текста и измените шрифт в нем на следующий: Tahoma, курсивный, 10 пт, подчеркнутый.
4. Выделите еще один фрагмент текста и измените шрифт в нем на следующий: Arial, 10 пт, скрытый (“hidden”).
5. Перейдите в режим отражения непечатаемых символов. Для этого щелкните на кнопке ¶ панели инструментов. Понаблюдайте, как изменится при этом третий выделенный фрагмент текста. Какие

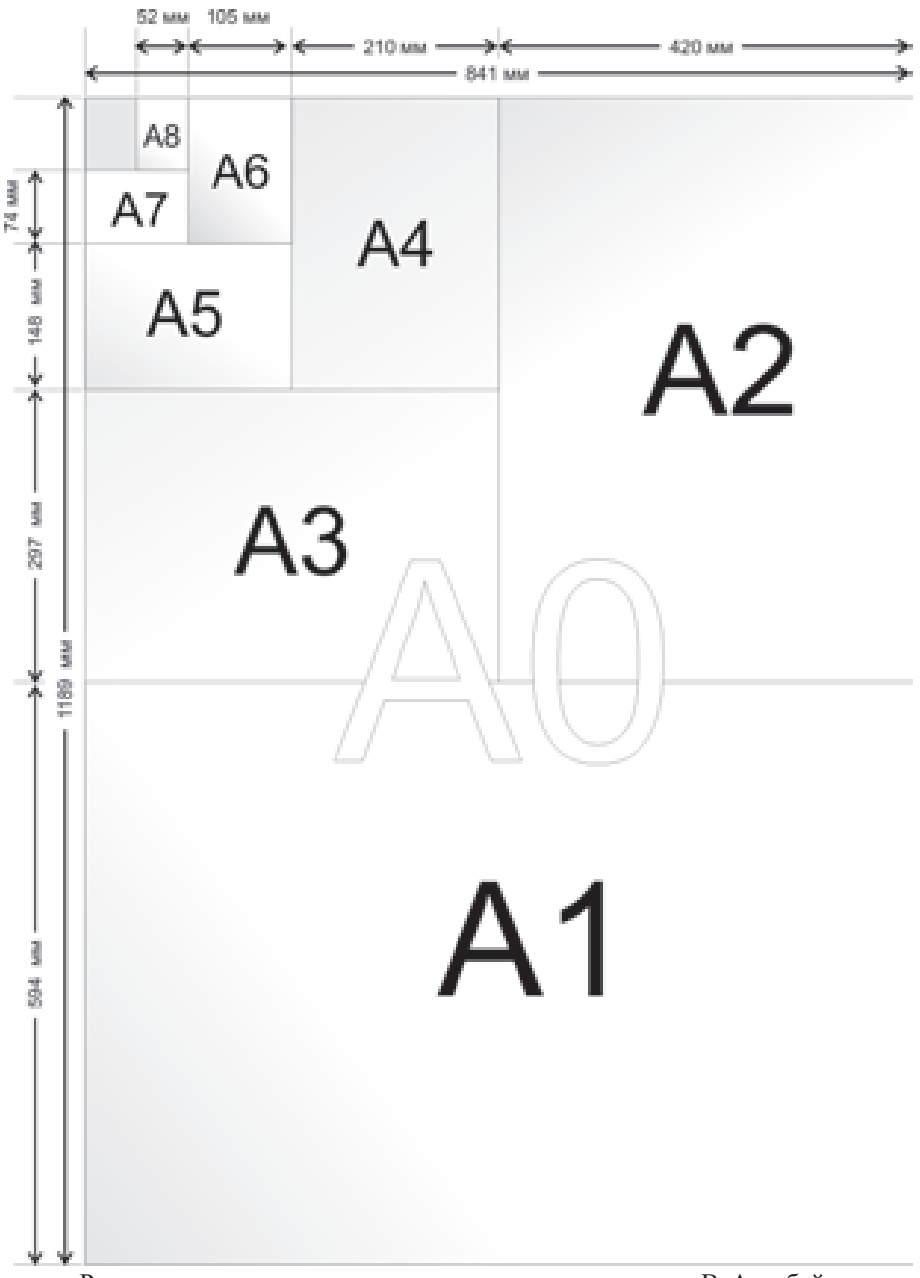
символы, не отражающиеся при печати документа, вы видите в этом режиме?

6. Выделите еще два фрагмента текста, используя клавишу “Ctrl”, и измените шрифт в них на следующий: Comic Sans MS, 20 пт, зеленый, контурный.

Каждый документ можно сохранить в файле, отобразить на экране, распечатать на бумаге. При распечатке документ на бумаге (*физической странице*) будет иметь точно такой же вид, в каком он отображается на экране.

На физической странице отводится определенное место для размещения объектов документа – оно называется *логической страницей*.





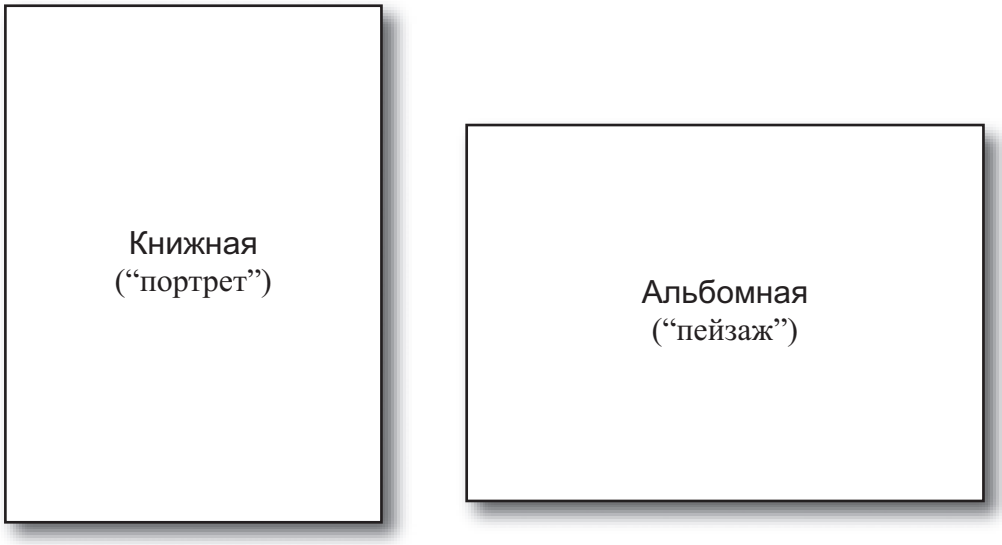
Размер печатного листа измеряется в миллиметрах. В Азербайджане в качестве основного размера взят размер 841x1189 мм, формата **A0** (точно так же, как в свое время в бывшем СССР). Производные от этого формата, более мелкие, именуются **A1, A2…. A8.** Каждый последующий формат получается делением предыдущего, более крупного, пополам.

В современном делопроизводстве в качестве основного принят формат **A4** (210x297 мм). В особых случаях (для выведения на печать крупных таблиц) используется формат **A3** (297x420 мм).

При форматировании страницы документа в любой прикладной программе обычно в первую очередь учитываются следующие параметры:

- размер страницы (листа);
- ориентация страницы;
- поля;
- колонтитулы.

Ориентация страницы определяется положением листа бумаги в пространстве. Она бывает двух типов: книжная и альбомная (“портрет” и “пейзаж”).

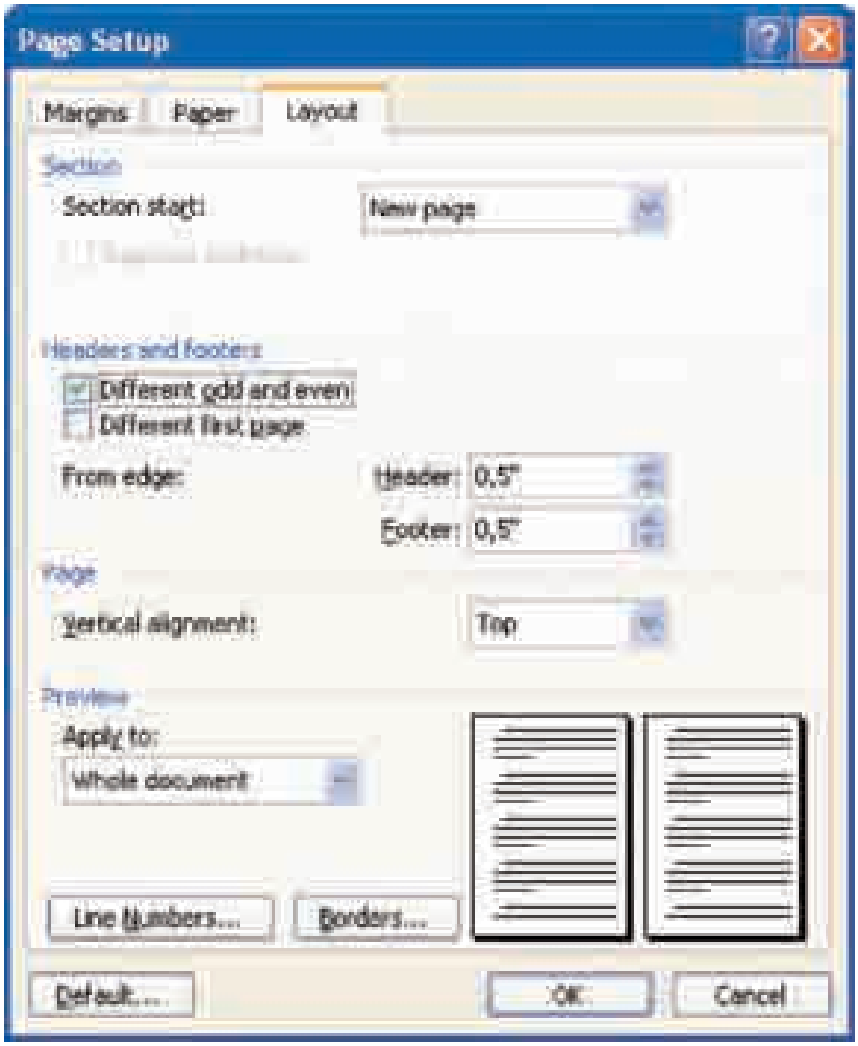


Поля – это части физического листа, предназначенные для размещения сносок, колонтитулов и другой дополнительной информации. Правое и левое поля обычно ничем не заполняются. Колонтитулы, сноски и т.д. располагаются в верхнем и нижнем полях.

Колонтитулы (от фр. “colonne” – колонна и лат. “titulus” – заголовок, письмо) – это служебная информация, размещаемая на верхнем и нижнем поле. Она может включать в себя имя автора, название (произведения, раздела, главы, параграфа), дату, номер страницы и т.д. Такие элементы облегчают работу с большими документами, а в уже изданном произведении обеспечивают удобство при чтении.

Упражнение

1. Откройте какой-нибудь текстовый документ.
2. Выполните команду File⇨Page Setup(Файл ⇨ Параметры страницы).
3. Поставьте флажок в диалоговом окне Different odd and even (Различать колонтитулы четных и нечетных страниц) вкладки Layout (Источник бумаги).



4. Выполните команду View⇨Header and footer(Вид⇨Колонтитулы).
5. Впишите в поле верхнего колонтитула нечетной страницы название:

Макет 1

6. Впишите в поле нижнего колонтитула четной страницы собственные имя и фамилию, предварительно нажав клавишу <Tab>, чтобы надпись расположилась по центру поля.
7. Закройте панель колонтитулов. Ознакомьтесь с документом и найдите в колонтитулах повторяющийся текст.

1. Какие этапы включает в себя процесс создания компьютерного документа?
2. Что такое редактирование документа и какие операции к нему относятся?
3. Что такое форматирование документа и какие операции к нему относятся?
4. Что такое логическая страница и из каких элементов она состоит?
5. Наберите какой-нибудь текст в текстовом редакторе и отформатируйте его.

3

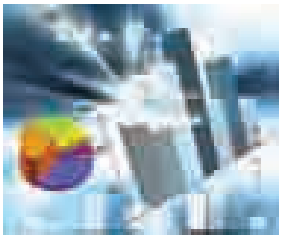
ТАБЛИЧНЫЙ ПРОЦЕССОР

3.1. НАЗНАЧЕНИЕ ТАБЛИЧНОГО ПРОЦЕССОРА

Для нормального восприятия данных очень часто пользуются таблицами. Прикладные программы, предназначенные для отображения и обработки информации, представленной в табличной форме, называют *электронными таблицами*. Иногда используют термин "*электронный процессор*". Рабочая область электронной таблицы напоминает по своей структуре шахматную доску. Она состоит из строк и столбцов, имеющих свои имена.

> Табличный процессор на компьютере служит для работы с информацией, представленной в табличной форме – в виде электронной таблицы.

Результатом работы программы является документ в форме таблицы или диаграммы. Например, в табличном процессоре можно вести классный журнал. Учителя смогут заносить в него оценки учащихся, а встроенные формулы позволят высчитывать средний балл для каждого ученика, общую успеваемость класса по предмету. Каждый раз, когда учитель будет вносить новую оценку, табличный процессор будет автоматически пересчитывать все результаты.

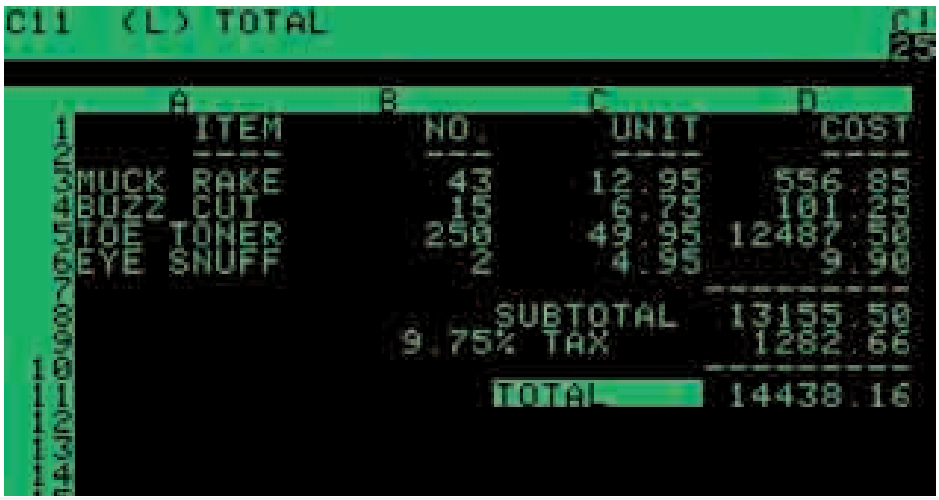


Характерной особенностью табличного процессора является то, что данные в нем и результаты действий задаются в табличной форме. Для большей наглядности эти данные можно представить в графической форме, в виде диаграммы.

По сравнению с бумажной предшественницей электронная таблица предоставляет пользователю гораздо больше возможностей для работы.

В ячейки таблицы, помимо чисел, дат и текста, можно записать также логические выражения, функции и формулы. Формулы позволяют почти мгновенно производить пересчет и выводить в соотвествующей ячейке новый результат при изменении исходных данных.

Первым табличным редактором, разработанным в 1979 году, был VisiCalc. Затем на рынок было выпущено много продукции этого класса: **SuperCalc, Microsoft MultiPlan, Quattro Pro, Lotus 1-2-3, Microsoft Excel, OpenOffice.org Calc.**



VisiCalc – первая электронная таблица

Впервые идею электронной таблицы сформулировал в 1961 году американский ученый **Ричард Матессич** в своей печатной работе “Budgeting Models and System Simulation”. Впоследствии эта концепция была разработана учеными **Пардо** и **Ландау**.

В 1979 году **Ден Бриклин** (1951) разработал совместно с **Бобом Френкстоном** (1949) программу VisiCalc, тем самым заложив основы электронных таблиц. Этот табличный редактор, разработанный для компьютера Apple II, превратил персональный компьютер из экзотической игрушки в массовое средство для бизнеса.

В настоящее время среди электронных таблиц наибольшей популярностью пользуется программа **Microsoft Excel**, входящая в пакет Microsoft Office.

Запуск программы Excel 2003. Как и для всех программ, имеющихся на компьютере, самый простой способ запуска программы Excel 2003 – использование меню Start (Пуск) в системе Windows. Для этого надо:

1. Для открытия главного меню системы Windows сделать щелчок на кнопке Start (Пуск), находящейся на панели задач.

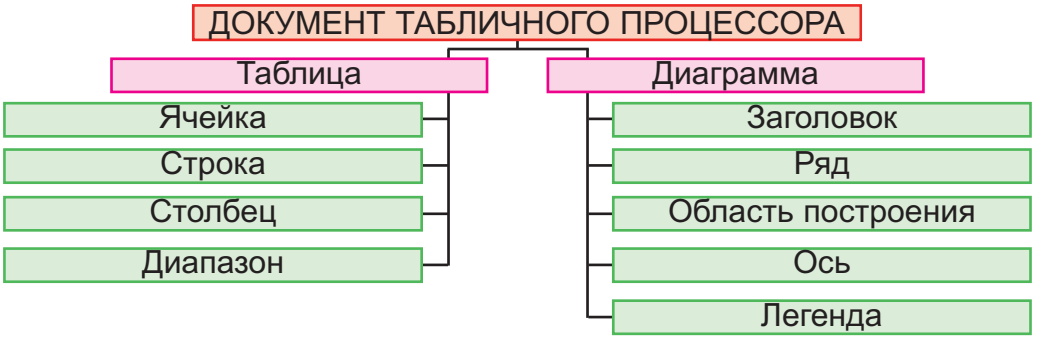
2. Из открывшегося меню выбрать элемент All Programs (Программы).
3. Из ниспадающего меню выбрать пункт Microsoft Office.
4. Из ниспадающего меню выбрать пункт Microsoft Excel 2003.

После этого начнется загрузка программы Excel 2003 и откроется окно этой программы с пустой книгой.

Файл программы Excel называется *рабочей книгой* [workbook], или просто *книгой*. Книга состоит из *рабочих листов* [worksheet].

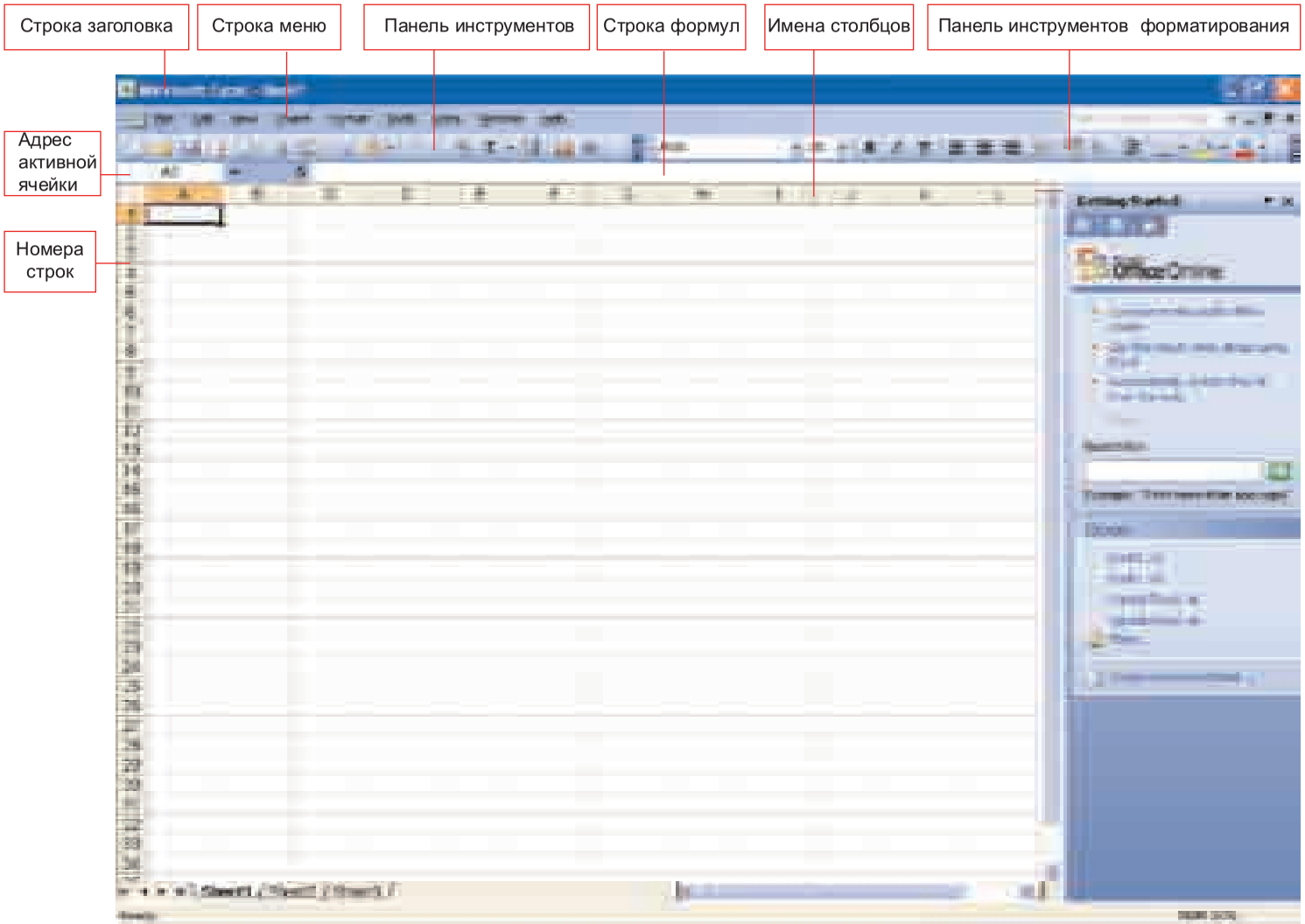
Сколько ячеек в таблице?
Если сказать, что на каждом рабочем листе есть миллионы ячеек, то это не будет ложью. На самом деле, на листе расположено 65536 строк и 256 столбцов. Если перемножить эти значения, то получится 16777216. Значит, на каждом рабочем листе более 16 миллионов ячеек. Если и этих ячеек не хватит, то следует отметить, что в каждой новой книге 3 листа. Другими словами, в вашем распоряжении в книге 50331648 ячеек и, если и этого не хватит, то в книге можно увеличить число рабочих листов.

Объекты электронной таблицы. Табличный документ бывает двух видов: *таблица* и *диаграмма*. Диаграмма является вспомогательным документом, не существующим без таблицы. В нижеследующей схеме дана классификация объектов документа табличного процессора.



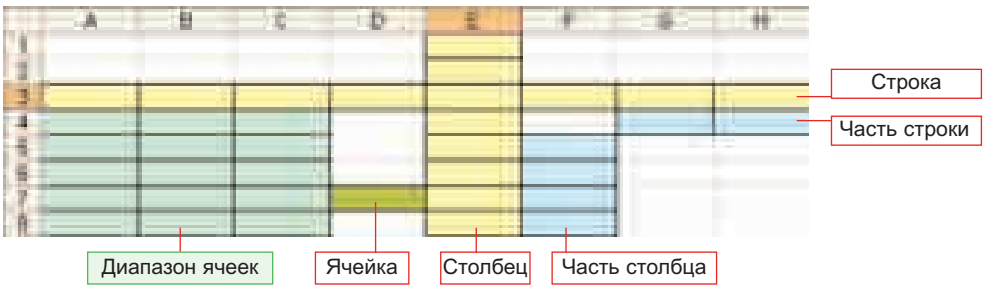
Первый уровень классификации относится к виду документа. На втором уровне показаны объекты, образующие таблицу и диаграмму. На следующих уроках будет более обстоятельно рассказано об этих объектах

1. Что такое табличный процессор?
2. Каково преимущество электронной таблицы перед обычной?
3. Как запустить электронный процессор Excel?
4. Когда была создана первая электронная таблица и как она называлась?
5. Что такое рабочая книга?
6. Запустите табличный процессор Excel вышеописанным способом. Каким еще способом можно это сделать?


Строка заголовка
Строка меню
Панель инструментов
Строка формул
Имена столбцов
Панель инструментов форматирования
Адрес активной ячейки
Номера строк

3.2. ОБЪЕКТЫ ЭЛЕКТРОННОЙ ТАБЛИЦЫ

Таблица представляет собой сложный объект, который состоит из элементарных объектов: строки, столбца, ячейки, диапазона ячеек. У каждого элементарного объекта есть имя, которое определено разработчиками электронной таблицы



Ячейка – элементарный объект, расположенный на пересечении строки и столбца электронной таблицы.

Строка – это все ячейки, расположенные горизонтально на одном уровне. Номер строки указывается целым числом, начиная с 1.

Столбец – это все ячейки, расположенные вертикально на одном уровне. Заголовки столбцов задаются буквами латинского алфавита сначала от **A** до **Z**, затем от **AA** до **AZ**, от **BA** до **BZ** и т.д. Имя последнего столбца будет – **IV**.

Диапазон ячеек – это группа смежных ячеек: строка или часть строки, столбец или его часть, а также несколько смежных ячеек, образующих прямоугольную область. Одну ячейку тоже можно считать диапазоном.

Адрес ячейки в таблице определяется ее местом в ней и содержит заголовок столбца и номер строки, на пересечении которых она находится. Вначале записывается заголовок столбца, а затем номер строки, например, **A3, D6, AB19**.

Диапазон ячеек задается указанием первой и последней его ячеек, разделенных двоеточием (:). Например, адрес диапазона, выделеннного на верхнем рисунке, будет **A4:C8**.

Основным местом хранения данных в таблице является ячейка. Для того, чтобы ввести данные в ячейку, ее надо выделить. Выделенная ячейка

берется в толстую черную рамку. Для выделения ячейки используют как мышь, так и клавиши клавиатуры.

Выделенную ячейку называют *активной* [active cell]. Имя активной ячейки можно увидеть на верхней левой стороне рабочего листа. Если выделен диапазон ячеек, то первая выделенная ячейка будет активной.

Способы выделения объектов электронной таблицы.

- Чтобы выделить ячейку, надо навести указатель мыши на нее и щелкнуть, или же с помощью клавиш управления курсором переместить курсор на нужную ячейку.
- Для выделения столбца надо щелкнуть на его заголовке.
- Для выделения строки надо щелкнуть на ее номере.
- Диапазон ячеек можно выделить несколькими способами:
  - путем протаскивания указателя мыши при нажатой левой кнопки;
  - с помощью клавиш управления курсором при нажатой клавише <Shift>;
  - вводом с клавиатуры начального и конечного адресов ячеек диапазона, разделенных двоеточием.

Данные, вводимые с клавиатуры, записываются в активную ячейку. Для того, чтобы ввести данные в ячейку, надо:

1. Навести указатель мыши на нужную ячейку и щелкнуть мышью, или же использовать для этого клавиши перемещения курсора.
2. Ввести число (например, 19 или 12,3), текст (например, Количество учеников) или формулу.

После ввода данных в ячейку для изменения активной ячейки можно использовать клавиши или комбинацию клавиш.

<Enter> . . . . . Выделяет ячейку под текущей ячейкой.
<Tab> . . . . . Выделяет ячейку справа от текущей ячейки.
<Shift+Enter> . . . . Выделяет ячейку сверху от текущей ячейки.
<Shift+Tab> . . . . . Выделяет ячейку слева от текущей ячейки.

Если ввести значение в ячейку **A1** и нажать клавишу <Enter>, то Microsoft Excel выделит ячейку **A2**.

Если ввести значение в ячейку **A2** и нажать клавишу <Tab>, то выделится ячейка **B2**.

Задание

1. Щелкните по кнопке  New Blank Document на панели инструментов. Откроется новая книга Book2.
2. Выделите ячейку **A3**.
3. Наберите с помощью клавиатуры слово **Порядковый**. Обратите внимание на то, что набранный вами текст отображается и в ячейке, и в строке формул.
4. Нажмите клавишу <Tab>. Активной станет ячейка справа **B3**. Наберите текст **Образовательное учреждение**.
5. Нажмите заново клавишу <Tab> и введите текст **Год создания**. Нажав снова клавишу <Tab> наберите: **Число учащихся**. Нажмите клавишу <Enter>. Активной станет ячейка **A4**.
6. Щелкните на ячейке **A3**. Запись **Порядковый** будет видна как в самой ячейке, так и в строке формул. Приведите указатель мыши на конец слова **Порядковый** в строке формул и щелкните мышью.

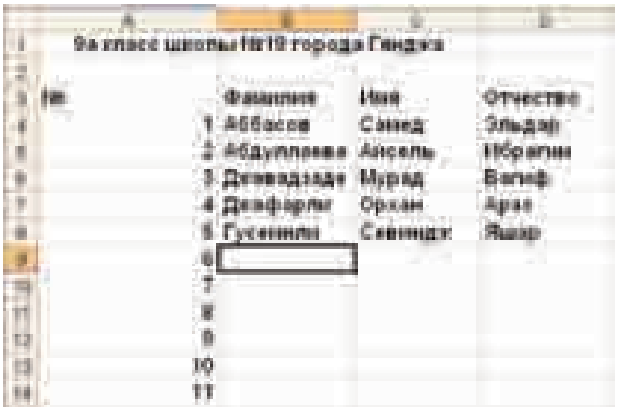
7. Нажмите клавишу Пробел и напишите слово **номер**. Затем нажмите клавишу <Enter> или значок  в строке формул. Изменится содержание ячейки **A3**.
8. Перейдите в ячейку **B3**. В ней содержится запись **Образовательное учреждение**. Введите текст **Название школы** и нажмите клавишу <Enter>. Содержимое ячейки изменится на новую запись.
9. Щелкните по ячейке **C3**. В ней имеется запись **Год создания**. Нажмите клавишу <F2>. Обратите внимание на то, что курсор ввода перешел в конец текста.
10. Используя клавишу <Backspace>, удалите существующий текст, наберите **Год образования** и нажмите клавишу <Enter>. Запись в ячейке заменится на новую, и ячейка **C4** станет активной.
11. Для перехода к клавише **A4** дважды нажмите клавишу <←> . Введите число **1** и нажмите клавишу <Enter>. Станет активной ячейка **A5**.

12. После каждого ввода номера нажмите клавишу <Enter> и введите числа

5
3
4
5

13. Для выделения столбца **A** щелкните по соответствующей букве. Обратите внимание, что все ячейки поменяют цвет, кроме первой, так как она активная.
14. Выделите 5-ю строку. Введите число **2**. Обратите внимание: это число записалось в ячейку **A5**.
15. Сохраните рабочий лист под названием **Школы**.

1. Перечислите объекты электронной таблицы.
2. Как распознать столбец, строку, ячейку?
3. Как задается адрес диапазона ячеек?
4. Что такое активная ячейка?
5. Составьте таблицу вашего класса на основе приведенного образца.



6. Составьте электронную таблицу на любую тему.

3.3. ДАННЫЕ ЭЛЕКТРОННОЙ ТАБЛИЦЫ. ФОРМУЛЫ

В табличных процессорах предусмотрены разные форматы представления данных. Форматы определяют типы данных электронной таблицы: символьные (текстовые), числовые, логические, даты и т.д. От того, какой формат выбран для той или иной ячейки, будет зависеть, какие действия табличный процессор сможет выполнять над ее содержимым.

Допустим, в ячейку помещены следующие цифры: 28051918. Как они будут восприняты табличным процессором? Если установлен *текстовый формат*, то цифры будут восприняты как символы 2,8,0,5,1,9,1,8. Если установлен *числовой формат*, то эта запись будет воспринята как число, и если задан *формат дат*, то эти цифры будут восприняты как дата 28 мая 1918 года.

Текстовый тип данных. Текстовые данные – это некоторый набор символов. Если первый из них является буквой, кавычкой, апострофом или пробелом, либо цифры чередуются с буквами, то такая запись воспринимается как текст.

Действия над символьными данными производятся аналогично действиям над объектами в текстовом процессоре.

Примеры текстовых данных:
Расписание занятий
“236
9 А класс
001

Числовой тип данных. Числовые данные представляют собой последовательность цифр, которые могут быть разделены десятичной запятой и начинаться с цифры, знака числа (“+” или “ –”) или десятичной запятой. Над числовыми данными в электронной таблице могут производится различные математические операции.

Примеры числовых данных:
232.5 –13.7 .546 +100

Если в ячейке таблицы хранится последовательность цифр, начинающаяся с кавычки, то, хотя такой набор цифр и выглядит на экране как число, на самом деле это текст. Его нельзя использовать в вычислениях. Любые текстовые данные в вычислениях всегда воспринимаются как ноль.

Существует признак, по которому можно определить, какой тип данных в ячейке: числовой или текстовый. Если в ячейку вводится текст, то он автоматически после нажатия клавиши <Enter> выравнивается по левому краю ячейки, а если вводятся числовые данные, то они выравниваются по правому краю ячейки.

Текст	12,5
.0123	0,0123
\456	-456

Логический тип данных. Логические данные используются в логических формулах и функциях. Данные этого типа отображаются в текущей ячейке следующим образом: если вводится любое отличное от нуля число (целое или дробное), то после нажатия клавиши <Enter> в этой ячейке будет выведено True [истина]. Ноль в соотвествующей ячейке отобразится как False [ложь].

Такое представление данных связано с представлением *логической переменной* в алгебре логики.

Тип данных – даты. Этот тип данных используют при добавлении числа к дате, вычисления разности двух дат, при пересчете даты, например, вперед или назад. Преобразование чисел в дату происходит автоматически, в зависимости от заданного формата. Табличный процессор дает возможность представления введенных чисел как даты в нескольких форматах. Например,

28 апреля 2008

Апрель 2008

Апрель

28.04.2008

04.2008

28 апреля

Формулы. Электронная таблица предназначена в первую очередь для автоматизации вычислений. Для этой цели в ячейки вводятся *формулы*.

> Ввод любой формулы начинается со знака равенства (=). Если он отсутствует, вводимая формула воспринимается как текст.

В формулах могут быть использованы числовые данные, знаки операций, различные функции, адреса объектов таблицы. Формулы с адресами ячеек можно сравнить с математическими формулами, в которых вместо адресов ячеек – переменные.

Адреса, которые участвуют в формулах, называются *ссылками*. Ссылки позволяют связывать между собой любые ячейки электронной таблицы и проводить необходимую обработку табличных данных.

> Ссылка – это адрес объекта (ячейки, строки, столбца, диапазона), используемый при записи формулы.

Формулы состоят из *операндов*, соединенных между собой знаками арифметических и логических операций. Операндом может быть данное, ссылка, функция.

Различают *арифметические* (алгебраические) и *логические* формулы.

Арифметические формулы. В арифметических формулах используются арифметические операции (“ + ” сложение, “ – ” вычитание, “ * ” умножение, “ / ” деление и “ ^ ” возведение в степень). При вычислении по формулам, так же, как и в математике, соблюдается порядок выполнения арифметических операций: сначала выполняется возведение в степень, затем – умножение и деление и наконец – сложение и вычитание. Операции одного уровня, например, умножение и деление, выполняются слева направо. Для изменения порядка выполнения арифметических действий используют круглые скобки. Действия над операндами, заключенными в круглые скобки, выполняются в первую очередь.

Результатом вычислений по арифметической формуле является число. При каждом изменении входящих в формулу операндов результат пересчитывается заново и отображается в соответствующей ячейке.

Логические формулы Логическая формула содержит некоторое условие и определяет, истинно оно или ложно. Истинному выражению присваивается значение “true” (“истина”, 1), ложному выражению – “false” (“ложь”, 0).

Однотипные формулы. При работе с электронной таблицей часто возникает необходимость в заполнении какого-то диапазона ячеек формулами, имеющими одинаковую структуру, но разные значения переменных, то есть когда формулы различаются ссылками. Такие формулы называются *однотипными*.

> Однотипные формулы – это формулы, имеющие одинаковую структуру, но различающиеся ссылками.

Для упрощения и ускорения ввода однотипных формул используют такой прием: ***формула вводится только в одну ячейку, далее она копируется в другие ячейки***.

Пример однотипных формул:

=A1+5	=A1*5	=A1+B3	=A1*B3	=(A1+B3)*D2
=A2+5	=B1*5	=A2+B4	=B1*C3	=(C3+D5)*F4
=A3+5	=C1*5	=A3+B5	=C1*D3	=(D4+E6)*G5
=A4+5	=D1*5	=A4+B6	=D1*E3	=(B4+C6)*E5

Абсолютная, относительная и смешанная адресация. В однотипных формулах могут использоваться различные ссылки. Существуют такие однотипные формулы, при копировании которых часть ссылок изменяется закономерным образом, другая же часть остается неизменной для всех формул.

При копировании формулы в другое место таблицы надо определить способ автоматического изменения ссылок. Для этого исползуются относительные, абсолютные и смешанные ссылки

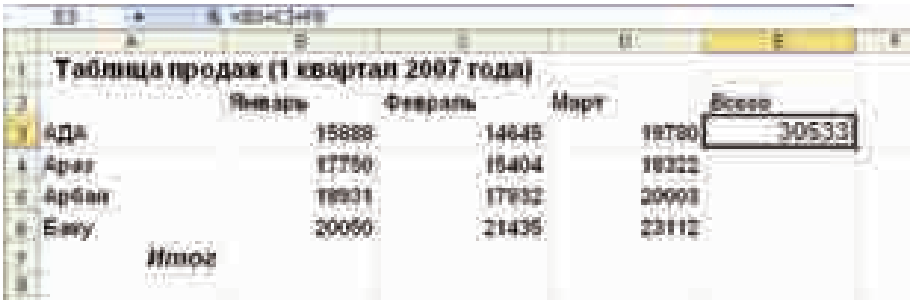
- Если формула при копировании изменяется, то используются *относительные ссылки*.
  Относительная ссылка записывается в обычной форме, например, **F3** или **E7**. При копировании во всех ячейках, куда будет помещается формула, изменятся и буква столбца, и номер строки.
- *Абсолютная ссылка* записывается в формуле в том случае, если при ее копировании не должны изменяться обе части: буква столбца и номер строки. Это указывается при помощи символа $, который ставится и перед буквой столбца и перед номером строки. Например, **F3** или же **E7**. Во всех ячейках, куда формула будет скопирована, появятся точно такие же формулы.
- *Смешанная ссылка* используется в формулах тогда, когда при копировании формулы может изменяться только какая-то одна часть ссылки – либо буква столбца, либо номер строки. При этом символ $ ставится перед той частью ссылки, которая должна оставаться неизменной. Например, **$F3** или **E$7**.

Задание

1. Запустите текстовой процессор.
2. Создайте новый документ (рабочую книгу).
3. Заполните таблицу согласно образцу.

4. Выделите ячейку **E3**. Введите с помощью клавиатуры формулу **=b3+c3+d3**. Обратите внимание на то, что набранная формула отображается как в ячейке, так и в строке формул.

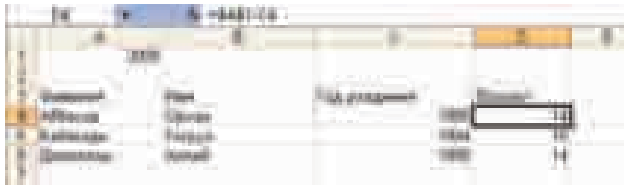
5. Нажмите клавишу <Enter> или щелкните на значке ✓ в строке формул. Результат вычисления по формуле – число 50313 появится в ячейке **E3**.
6. Щелкните по ячейке **E3**. Обратите внимание на то, что формула отображается в строке формул.



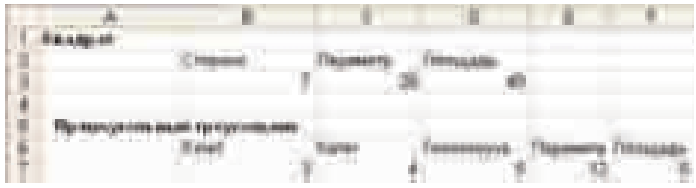
7. Щелкните по ячейке **E4**. Введите с клавиатуры знак равенства "=". Затем щелкните по ячейке **B4**. Обратите внимание: ссылка на ячейку **B4** видна как в строке формул, так и после знака равенства " =".
8. Введите с клавиатуры знак + и щелкните на ячейке **C4**, еще раз введите + и щелкните на ячейке **D4**. Формула **=B4+C4+D4** будет отображаться как в ячейке, так и в строке формул.
9. Нажмите клавишу <Enter> или нажмите на значок ✓ в строке формул. В ячейке **E4** появится значение формулы – 51476.
10. Сделайте двойной щелчок по ячейке **B4**. Поменяйте значение ячейки на **16750**. Затем нажмите клавишу <Enter>. Обратите внимание, что значение ячейки E4 изменится на 50476.
11. Выберите ячейку **E5**. Введите с клавиатуры формулу **=b4+c4+d4** и нажмите клавишу <Enter>. В ячейке **E5** появится значение 50476. Обратите внимание: отображаемое значение одинаково с содержимым ячейки **E4**, так как их содержимое вычисляется по одной и той же формуле.
12. Сделайте двойной щелчок по ячейке **E5** и замените в формуле ссылку **B4** на **b5**. Нажмите клавишу <Enter>.
13. Щелкните на ячейке **E5**. Нажмите клавишу <F2> и замените вторую ссылку **C4** на **c5**.
14. Щелкните на строке формул и замените третью ссылку **D4** на **d5**. Затем нажмите клавишу <Enter>. В ячейке **E5** отобразится значение 56866.
15. Щелкните по ячейке **E6** и введите с клавиатуры знак равенства "=". Затем щелкните по ячейке **B6**, введите +, щелкните по ячейке **C6**, введите + и щелкните по ячейке **D6**. Нажмите клавишу <Enter> и посмотрите, какое число отобразится в ячейке **E6**. Там должно быть число 64597.

1. **Назовите типы данных, используемых в табличном процессоре, и их особенности.**
2. **Что такое ссылка в электронной таблице и чем она отличается от адреса?**
3. **Что такое относительная адресация в формулах? Приведите примеры.**
4. **Что такое абсолютная адресация в формулах и как она указывается? Приведите примеры.**
5. **Объясните правила копирования формул.**
6. **Введите в ячейку A2 – значение переменной x, в ячейку B2 – значение переменной y, в ячейку C2 – значение переменной z. Получите значение математического выражения (x+y)*2+3*z в ячейке D2. Для этого запишите эту формулу в ячейку D2. Задавая различные значения для x, y, z проследите за изменением значения ячейки D2.**

7. **Создайте таблицу согласно образцу. Используя формулу, вычислите возраст учащихся.**



8. **Укажите в ячейке A2 скорость автомобиля, а в ячейке B2 – время, затраченное на прохождение определенного расстояния. Вычислите в ячейке C2 путь, пройденный автомобилем (s=v*t). Увеличьте значение ячейки B2 в 2, 3, 4 раза. Как изменится значение ячейки C2?**
9. **Создайте нижеследующую электронную таблицу. В ячейках B3, B7, C3, D7 размещены числа. Запишите в ячейку C3 формулу =4*B3, в ячейку D3 формулу =B3*B3, в ячейку E7 формулу =B7+C7+D7, а в ячейку F7 формулу =B7*C7/2. Поменяйте числа в ячейках B3, B7, C7 и D7 на другие и проследите изменения в ячейках с формулами.**



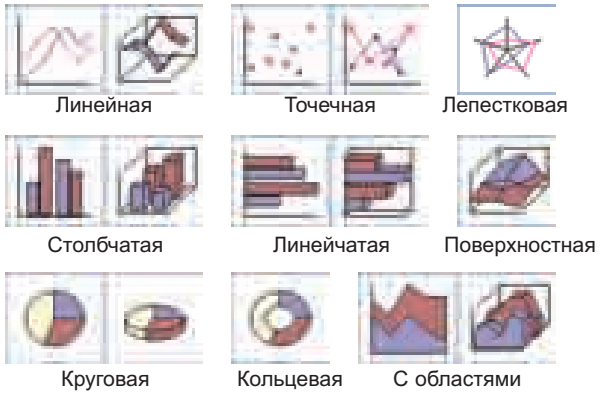
10. **Немецкий физик Г.Фаренгейт в 1724 году предложил температурную шкалу, названную его именем. Температура по шкале Фаренгейта связана с температурой по шкале Цельсия соотношением:** $t\,^{\circ}C = \frac{5}{9}(t\,^{\circ}F - 32)$. **Составьте таблицу, переводящую температуру, измеренную по шкале Фаренгейта, в температуру по шкале Цельсия.**

3.4. СОЗДАНИЕ И РЕДАКТИРОВАНИЕ ДИАГРАММЫ

Диаграмма является одним из объектов электронной таблицы. Она предназначена для представления данных в графической форме. Данные, расположенные в одном столбце или в одной строке, называются *рядом*. Для построения диаграммы вначале надо указать ряды, а затем выбрать тип диаграммы. Каждая диаграмма характеризуется следующими параметрами: *имя, тип, область, размещение*.

Имя. Каждой диаграмме дается имя, под которым она включается в состав электронный таблицы.

Тип. В табличном процессоре можно строить диаграммы различных типов. Внизу показаны основные типы диаграмм.



Познакомимся с некоторыми из них.

- *Линейная диаграмма* [line chart] знакома вам из многих предметов. Ее называют еще *графиком*. На одном чертеже можно разместить несколько графиков, каждый из которых соотвествует своему ряду данных.
- *Столбчатая диаграмма* [column chart], или *гистограмма*, может быть построена для нескольких рядов данных. Высота каждого столбика определяется значением в соответствующей ячейке.
- *Поверхностная диаграмма* [surface chart] строится только для нескольких рядов и состоит из группы цветных многослойных поверхностей.
- *Круговая диаграмма* [pie chart] используется для отображения одного ряда значений. Каждый сектор такой диаграммы отражает относительную (выраженную в процентах) долю каждого значения из ряда от общей суммы всех данных.

Область ограничивает поле чертежа построения диаграммы.

Размещение. Диаграмма может размещаться либо на том же листе, что и таблица, либо на отдельном листе.

Объекты диаграммы. Диаграмма сама является сложным объектом и состоит из элементарных объектов: *ряд, ось, заголовок, легенда, область построения.*

Ряд. Диаграмма может быть построена как по одному ряду, так и по нескольким рядам. Для выделенного диапазона ячеек построение диаграммы ведется по нескольким рядам данных. В этом случае в качестве каждого ряда принимается соответствующая строка или столбец выделенного диапазона.

Ось. Каждая из осей диаграммы характеризуется следующими параметрами: *вид, шкала, шрифт, число, выравнивание.*

- *Вид* определяет отображение внешнего вида оси на экране.
- *Шкала* определяет минимальное и максимальное значение шкалы, значение основных и промежуточных делений, точку пересечения с другими осями.
- *Число* определяет формат шкалы в соответствии с типами данных, находящихся в диапазоне.

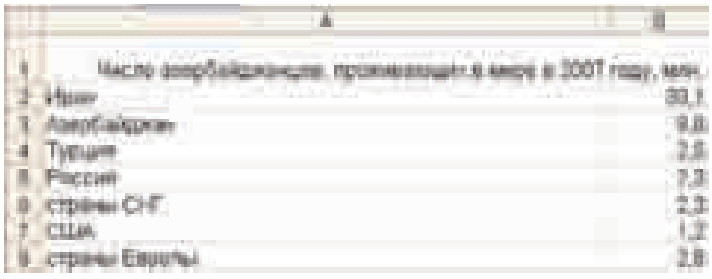
Заголовок. Заголовок размещается над диаграммой и, как правило, задается пользователем.

Легенда. Легенда диаграммы – это список названий рядов, обычно в рамочке.

Область построения. Это ограниченная осями область для размещения рядов данных. Для удобства анализа результатов на область построения может наносится сетка.

> Для выделения объекта диаграммы сначала надо сделать двойной щелчок на диаграмме, а затем щелкнуть на нужном объекте.

Создадим нижеследующую таблицу:



Для построения диаграммы к этой таблице нужно вначале выделить ячейки, которые необходимо графически представить (A2:B8), затем щелкнуть по кнопке  Chart Wizard на стандартной панели инструментов или же выбрать команды Insert⇨Chart из строки меню. В результате откроется соответствующее диалоговое окно.

На первой странице диалогового окна Chart Wizard нужно выбрать тип и вид диаграммы. При выборе конкретного типа диаграммы справа в области Chart sub-type отображаются виды этого типа диаграммы. Следует выбрать один какой-то вид. Выберем круговой тип (Pie) диаграммы и нажатием кнопки Next перейдем на следующую страницу.

На этой странице следует проверить, правильно ли указан диапазон выбранных ячеек на вкладке Data range и затем щелкнуть по кнопке Next.

На открывшейся странице определяется внешний вид диаграммы. Например, в области Chart title указывается заголовок диаграммы (**Число азербайджанцев, проживающих в мире в 2007 году, млн.**).

Если требуется отобразить значение данных на диаграмме, необходимо на вкладке Data Labels щелкнуть по кнопке Value. После определения внешнего вида диаграммы следует снова щелкнуть по кнопке Next. Откроется последняя страница диалогового окна Chart Wizard.

На этой странице определяется место, где будет размещена диаграмма: на отдельном листе (As new sheet) или же на текущем листе (As object in). Щелкнув по кнопке Finish диаграмма появится на выбранном месте.

Рассмотрим другой пример. Допустим, необходимо построить график функции $y= x^2 - 7x + 10$ на отрезке [-8; 8]. Для этого построим таблицу, содержащую значения этой функции (рис. 3.1.). Шаг изменения аргумента возьмем равным 1.

рис. 3.1.

Для построения диаграммы (графика) выделим диапазон ячеек **A4:B20** и щелкнем по кнопке Chart Wizard. Выберем тип диаграммы – XY (Scatter), а вид – или  , или  .

Затем щелкнем по кнопке Next и перейдем на новую страницу, где укажем заголовок диаграммы. На вкладке GridLines уберем все “флажки” и щелкнем по кнопке Finish. В результате получим график, изображенный на **рис.3.2**.

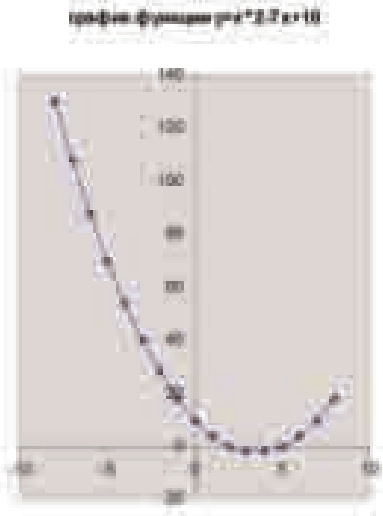
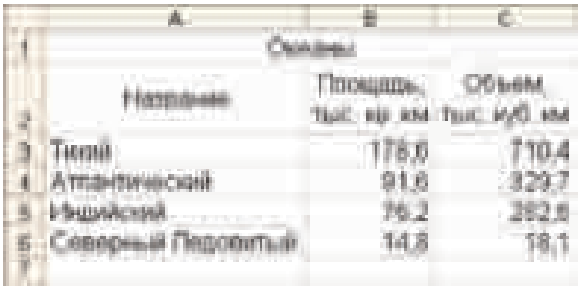
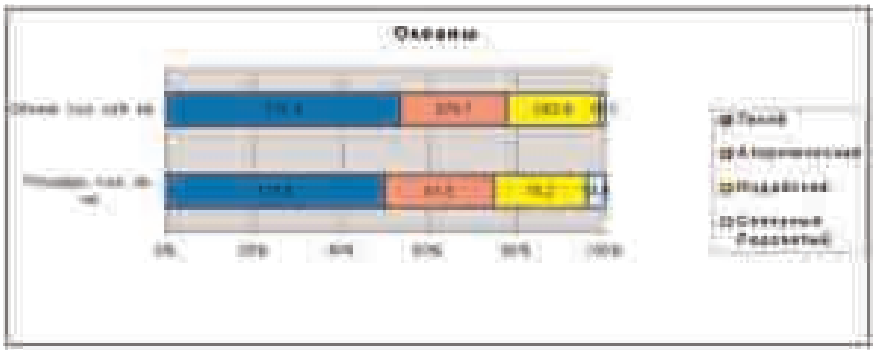


рис. 3.2.

1. Из каких объектов состоит диаграмма?
2. Назовите основные типы диаграмм в табличном процессоре.
3. Какие параметры характеризуют любую диаграмму?
4. Создайте данную таблицу и постройте соответствующую ей диаграмму.



	A	B	C
1		Океаны	
2	Название	Площадь, тыс. кв. км	Объем, тыс. куб. км
3	Тихий	178,6	710,4
4	Атлантический	91,6	329,7
5	Индийский	76,2	282,6
6	Северный Ледовитый	14,8	18,1
7			



5. Рассмотрев диаграммы, ответьте на вопросы:

- **Какая информация представлена на диаграмме? Как вы определили это?**
- **Как эту информацию можно представить в виде таблицы?**
- **Что больше всего интересует подрастающее поколение?**

3.5. ФОРМАТИРОВАНИЕ ТАБЛИЧНОГО ДОКУМЕНТА

Под *форматированием табличного документа* понимается ряд действий по изменению формы представления как самого документа, так и его объектов. Наряду со способами форматирования в текстовом процессоре, в электронной таблице имеются особые приемы форматирования:

- данные в ячейках могут быть представлены в различных форматах;
- может быть изменена высота строки и ширина столбца, где хранятся данные;
- любой объект электроной таблицы может быть взят в рамочку или же выделен специальным узором.

Форматирование произвольного объекта табличного документа производится посредством команд раздела Format из строки меню. А теперь познакомимся с форматированием отдельных ячеек, строк и столбцов.

Формат ячейки характеризуется следующими параметрами: *число, выравнивание, шрифт, рамка, вид, защита*. *Число* определяет тип данных, хранящихся в ячейке, и формат представления числовых значений. *Выравниванием* и *шрифтом* пользуются так же, как и во всех других программных средах. *Рамка* определяет внешнее обрамление ячейки (тип, толщину, штрих линии). *Вид* определяет заливку и узор фона ячейки. *Защита* определяет уровень защиты данных в ячейке. Например, можно защитить ячейку от изменения содержимого или же скрыть формулу в ней.

Формат строки позволяет регулировать высоту строки и управлять отображением строки в таблице. Высота строки регулируется автоматически или вручную. При автоматической регулировке высоты строки выбирается такое значение, чтобы все данные помещались в строке.

Формат столбца позволяет регулировать ширину столбца и управлять отображением столбца в таблице. Ширина столбца может регулироваться автоматически или вручную. При автоматической регулировке ширины столбца выбирается такое значение, чтобы все данные помещались в столбце в одну строку.

Отображение строк и столбцов. Любую строку или столбец в таблице можно спрятать. Это необходимо, когда строки или столбцы используются для записи промежуточных расчетов. Впоследствии скрытые строки и столбцы можно вновь вывести на экран.

Форматы данных. Для представления числовых данных используются различные форматы: общий, числовой, процентный, денежный, экспоненциальный и др.

- По умолчанию используется *общий формат*, и любые данные, введенные в этом формате (текст, числа, дата и т.д.), автоматически распознаются и форматируются.
- *Числовой формат* обеспечивает представление чисел в ячейках с заданным пользователем количеством десятичных знаков. Например, если выбран формат с точностью три десятичных знака после запятой, то при вводе числа 19 в ячейку на экране появится 19,000, а при вводе числа 0,12345 – число 0,123.
- *Процентный формат* обеспечивает представление числовых данных в форме процентов, со знаком “%“. Например, если установлена точность в один десятичный знак, то при вводе числа 0,257 на экране появится 25,7%, а при вводе 257 – 25700,0%.
- *Денежный формат* обеспечивает такое представление чисел, при котором каждые три разряда разделены пробелом, а следом за

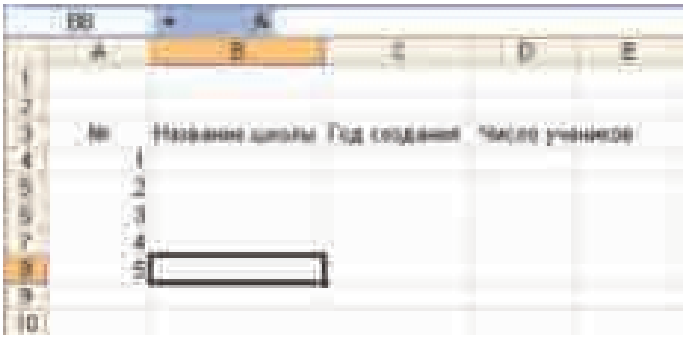
последним десятичным знаком указывается денежная единица размерности.

- *Экспоненциальный (научный) формат* обеспечивает представление вводимых чисел в виде двух частей: мантисы и порядка числа. Например, если задана точность в два знака после запятой, то число 12345 в экспоненциальном формате запишется как 1,23E+04. Здесь 1,23 – мантисса, а запись E+04 понимается как четвертая степень числа 10.

Изменение высоты строки и ширины столбца. Если ширина вводимого числа превышает ширину ячейки (столбца), то вместо числа в ячейке выводятся знаки # (решетка). При вводе текста, длина символов которого больше ширины ячейки, на экране отображается только часть введенного текста в пределах установленной ширины столбца. Количество умещающихся в ячейке символов при вводе текста зависит не только от ширины ячейки, но и от выбранного типа шрифта и его размера.

Во всех этих случаях возникает необходимость увеличить ширину столбца. Это можно сделать двумя способами:

- выбрав в меню Format команду Column⇨AutoFit⇨Selection или же команду Column⇨Width, в открывшемся диалоговом окне задать нужное значение;
- установить указатель мыши на границу раздела между заголовками столбцов (вид курсора изменится на двустороннюю стрелку) и, не отпуская левой кнопки мыши, изменить ширину столбца до нужного размера перетаскиванием границы влево или вправо.



Аналогично можно изменить и высоту строки.

В табличном процессоре есть возможность закрашивания фона объекта различными цветами и узорами.

Задание 1

1. Запустите табличный процессор и создайте файл **Школьники.xls**.
2. Заполните ячейки от **B4** до **B8**.
3. Наведите указатель мыши на правую границу заголовка столбца **B**. После того как указатель примет вид двусторонней стрелки, удерживая левую кнопку мыши, протяните его вправо для увеличения ширины столбца.
4. Щелкните по заголовку столбца **B** для его выделения.
5. Выберите в меню Format вкладку Column⇨AutoFit Selection. Программа Excel автоматически расширит ширину столбца по размеру его содержимого.
6. Для выделения строки **3** щелкните по ее заголовку.
7. В меню Format выберите вкладку Row⇨Height. Введите число **25** для установления высоты строки и нажмите ОК.
8. Вы увидите, что **3**-я строка будет в 3 раза выше других.
9. Щелкните по любой ячейке рабочего листа и отмените выделение строки. Сохраните изменения, закрыв файл.

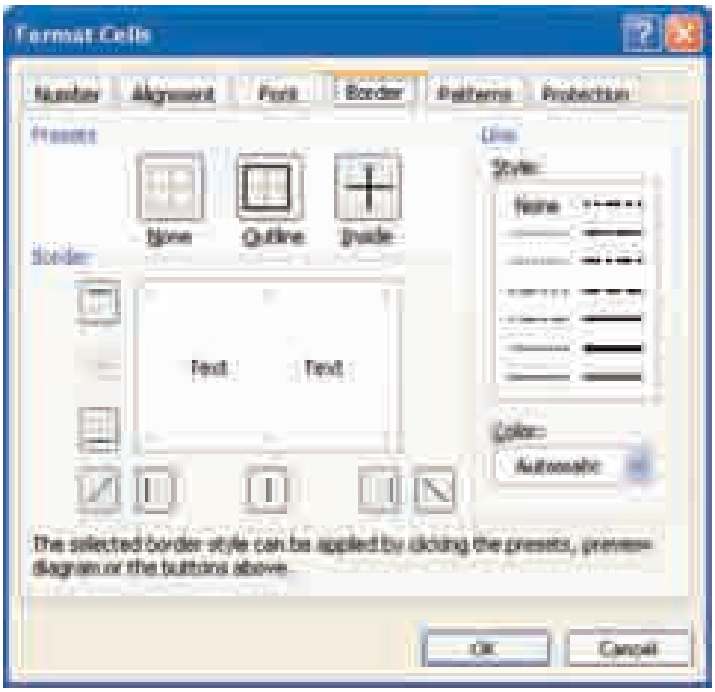
Задание 2

1. Создайте в программе Excel расписание ваших уроков, как в нижеприведенном примере.

РАСПИСАНИЕ УРОКОВ					
понедельник	вторник	среда	четверг	пятница	суббота
1 алгебра	англ. язык	алгебра	история	алгебра	англ.язык
2 литература	алгебра	история	Конституция	физика	биология
3 география	геометрия	русский язык	геометрия	русский язык	иностр.язык
4 биология	литература	химия	физ.-ра	иностр.язык	химия
5 черчение	история	информатика	экономика	история	физ.-ра
6 иностр. язык	физика	физика		англ.язык	

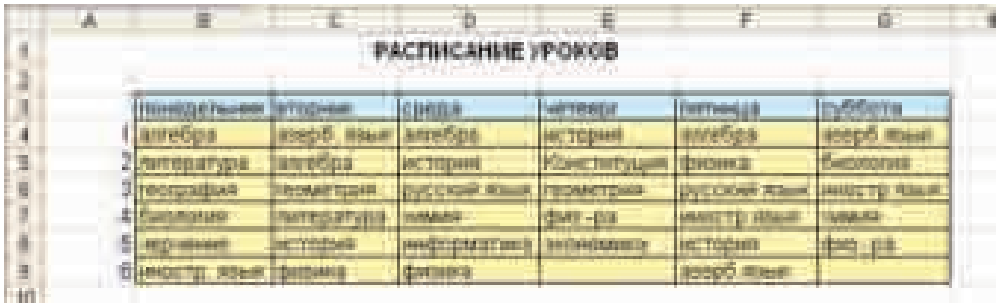
2. Чтобы разместить название таблицы (**РАСПИСАНИЕ УРОКОВ**) над таблицей посередине, выделите диапазон **A1:G1**. Выберите команду Format ⇨Cells.

3. В открывшемся диалоговом окне выберите вкладку Alignment (Выравнивание). Отметьте щелчком строку Merge cells (Объединение ячеек). Результатом этого действия будет объединение выделенных ячеек в одну – **A1**.
4. Чтобы разместить содержимое ячейки по центру, надо в этом же окне, в разделе Text alignment, выбрать в блоке Horizontal строку Center.
5. Отрегулируйте размеры строк и столбцов в таблице.
6. Выделите диапазон ячеек **B3:G3.** Выберите вкладку Patterns из диалогового окна Format Cells. Выберите цвет для диапазона (например, голубой). Щелкните на кнопке ОК. Выбранный диапазон зальется соответствующим цветом, и исчезнут границы между ячейками.
7. Для того, чтобы установить рамку для выбранных ячеек, необходимо выделить их и выбрать в диалоговом окне вкладку Borders.



Если щелкнуть по Outline, то у выбранных ячеек будут выделены внешние границы. Если необходимо отобразить и внутренние границы между ячейками, то необходимо щелкнуть по Inside. Для отображения и внешних, и внутренних границ необходимо щелкнуть последовательно по двум этим кнопкам.

8. Окрасьте диапазон ячеек **B4:G9** желтым цветом.



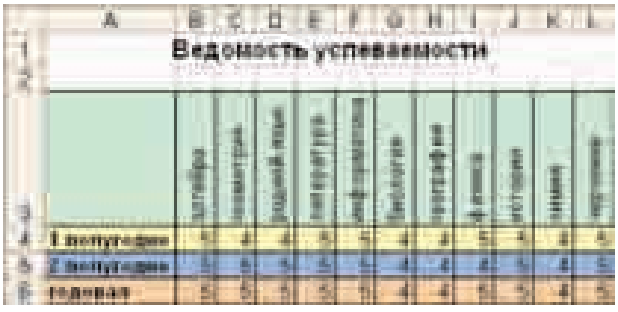
РАСПИСАНИЕ УРОКОВ

	понедельник	вторник	среда	четверг	пятница	суббота
1	алгебра	англ. язык	алгебра	история	алгебра	англ. язык
2	литература	алгебра	история	Конституция	физика	биология
3	география	геометрия	русский язык	геометрия	русский язык	иностр. язык
4	биология	литература	химия	физ.-ра	иностр. язык	химия
5	ИЗО или	история	информатика	экономика	история	физ.-ра
6	иностр. язык	физика	физика		англ. язык	

1. Что подразумевает форматирование документа?

2. Какими параметрами характеризуется форматирование ячейки?

3. Используя форматирование, создайте следующую электронную таблицу.



Ведомость успеваемости

	алгебра	геометрия	родной язык	литература	информатика	биология	география	физика	история	химия	иностранный
1 полугодие	4	4	4	4	5	4	4	5	4	4	4
2 полугодие	5	5	5	5	5	4	4	4	5	4	5
годовая	5	5	5	5	5	4	4	5	5	4	5

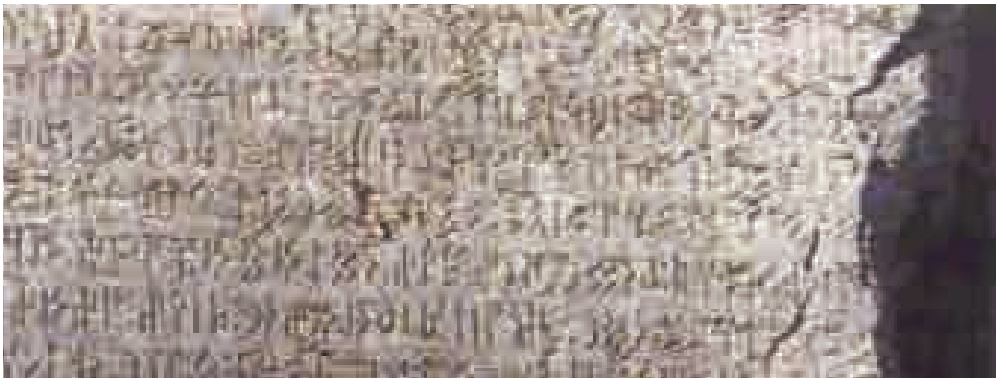
4 ИНФОРМАЦИОННОЕ ОБЩЕСТВО

4.1. ИНФОРМАТИЗАЦИЯ ОБЩЕСТВА

С момента возникновения человечества в нем всегда шла борьба сначала за ресурсы, затем – за энергию и, наконец, за информацию. На заре развития цивилизации человеку хватало элементарных знаний и умений, но затем, с увеличением количества информации, он начал ощущать недостаточность своих знаний. Для правильной обработки информации и принятия нужных решений требовалось обобщить знания и опыт людей. Для этого человек начал изобретать различные инструменты. Возникли специальные методы и приспособления для переработки информации, вызывавшие в обществе глубокие изменения – информационные революции.

В этом отношении человечество пережило четыре этапа, наиболее сильно повлиявшие на его развитие и именуемые *информационными революциями*.

Первый этап. Благодаря возникновению письменности стало возможным хранить знания и передавать их последующим поколениям.

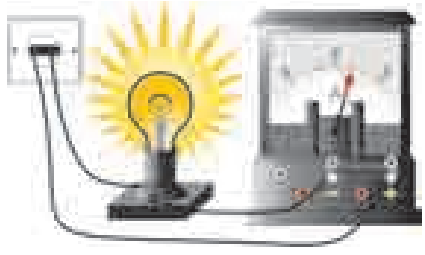


Второй этап (середина XVI века). Начался с изобретением книгопечатания. Итогом этой революции стало появление абсолютно нового способа

хранения информации. Человек получил новое средство для хранения, систематизации и распространения информации.

Это средство создало возможность для массового приобщения к духовным и культурным ценностям.

Третий этап (конец XIX века). Связан с открытием свойств электрического тока. Были созданы телеграф, телефон и радио, позволяющие передавать информацию в необходимом объеме с большой скоростью, принимать ее и хранить. Появились информационно-коммуникационные средства.



Четвертый этап (70-е гг. XX века). Изобретена технология микропроцессоров, появились персональные компьютеры.

Электрические и механические средства информации остались в прошлом – их сменили электронные средства. Эти средства позволили производить машины и приборы более миниатюрных размеров и создавать программно-управляемые устройства. Основной причиной начала четвертой революции явилось создание в 40-х гг. минувшего века электронной вычислительной машины.

Четвертая информационная революция дала человечеству толчок для перехода от *промышленного общества к информационному обществу*. Это напоминает переход человечества от аграрного общества к промышленному.

> Информационное общество – это такое общество, большая часть членов которого занимается производством, хранением, обработкой и использованием информации.

Вот некоторые характерные признаки информационного общества:

1. *С увеличением объема информации человек будет использовать для ее обработки и хранения специальные технические средства.* В информационном обществе человек или коллектив, для того чтобы принять какое-то решение, собирает определенную информацию, обрабатывает и анализирует ее. Объем информации достигает такой степени, что человек уже не способен сам ее обрабатывать – он привлекает для этой цели специальные технические средства.
2. *Пользование компьютером станет неизбежным.* В информационном обществе пользование компьютером – насущная необходимость. Оно дает возможность использовать достоверные источники информации, снижает объем бесполезной работы, ускоряет принятие оптимальных решений и автоматизирует обработку информации.
3. *Движущей силой общества станет производство информационных продуктов.* Во второй половине XX века значительная часть людей перешла из сферы производства материальных продуктов в информационную сферу. Сложилась новая социальная прослойка населения, не занятая в производстве материальных продуктов. Эти люди (педагоги, банковские служащие, программисты и т.д.) заняты обработкой информации. В новом обществе материальные блага будут по большей части “информационноемкие”. Их ценность будет зависеть от инноваций, дизайнерских решений и качества маркетинга.
4. *В информационном обществе производимым продуктом станут знания и интеллект, что, в свою очередь, увеличит долю интеллектуального труда.* Гораздо больше людей будут избирать профессии, связанные с интеллектуальным трудом.

5. *Произойдет переоценка ценностей, сформируется новый образ жизни, изменятся и развлечения, которыми человек занимается в свободное время.* Уже сейчас компьютерные игры занимают впечатляющую долю свободного времени человека. Эти игры трансформируются в систему, соединяющую игроков, географически далеких друг от друга. Растет число людей, проводящих время в Интернете. Они посещают образовательные сайты, путешествуют по виртуальным музеям, находят необходимую литературу в виртуальных библиотеках, и т.д. Особый успех у пользователей имеют службы бесед (чаты) и ICQ. С помощью этих служб люди, находящиеся далеко друг от друга, общаются в режиме реального времени.
6. *Будут развиваться компьютерная техника, компьютерные сети, информационные технологии.* Сеть Интернет будет ежемесячно расширяться на 10 –15%, число ее пользователей будет исчисляться сотнями миллионов. Использование современных мультимедийных систем, сочетающих в себе функции различных устройств (компьютера, телевизора, радио, телефона и т.д.) приведет к универсальности информационных технологий. Устройства для накопления информации уменьшатся в габаритах до того, что будут умещаться в ладони. В них, наряду с несколькими объемными энциклопедиями, будут размещены также универсальные личные сведения пользователя. При подключении этого устройства к сети можно будут получать оперативную информацию – например, прогноз погоды или сведения о пробках на дорогах.
7. *Во всех домах будет много электронных приборов и компьютерных устройств.* Вместо системы проводов дом будет оснащен одним электрическим и одним информационным кабелем. Информационный кабель будет служить для связи, трансляции телеканалов и выхода в Интернет. Специальный электронный блок будет контролировать все бытовое оборудование и жилищные системы по принципу “умный дом”. Помимо “умных домов”, появятся “умные автомобили”, оснащенные не только компьютерами, отвечающими за техническую часть автомобиля, но и системой, подключенной к городским службам информации. Такой автомобиль будет связан с “умным домом” и даже сможет управлять им.
8. *Производство энергии и материальных продуктов будет осуществляться при помощи машин, человек же будет занят преимущественно обработкой информации.* Количество людей, занятых в производстве, уменьшится – их место займут роботы и манипуляторы.

9. *В сфере образования будет создана непрерывная система обучения.* Человек получит возможность учиться всю жизнь, чтобы идти в ногу со временем, менять при необходимости профессию и занимать достойное место в обществе.
10. *Дети смогут учиться на дому посредством компьютерных программ и телекоммуникаций.* Но, наряду с этим, изменятся формы обучения в учебном процессе, в результате чего могут возникнуть проблемы, связанные с его воспитательным аспектом.
11. *Возникнет и будет развиваться рынок информационных услуг.* Информация будет иметь вид товаров или услуг. Этот продукт можно будет продавать как обычный товар.

Для перехода от промышленного общества к информационному должны были возникнуть условия информационного кризиса. Такие условия возникли в XX веке. Стало трудно ориентироваться в потоке информации, захлестнувшем людей. Появилось много излишней, ненужной информации. Переход к информационному обществу начался с использования современных средств для обработки и передачи информации в различных сферах. Этот процесс называется *информатизацией*.

Процесс *информатизации общества* обеспечивает переход от промышленного общества к информационному.

Процесс информатизации общества дает возможность каждому члену общества получать необходимую ему информацию.

До недавнего времени вместо слова “информатизация” использовалось слово “компьютеризация”. Однако компьютеризация предполагает лишь развитие и использование компьютерной техники, тогда как информатизация – более широкое понятие. В наши дни первостепенное значение имеют не столько технические средства, сколько сам социально-технический процесс. Компьютеризация – лишь часть процесса информатизации, его техническая база.

Информационная культура.

Человеческая культура определяется:

- знаниями, умениями и профессиональными навыками;
- уровнем интеллектуального, эстетического и духовного развития;
- формами и методами общения с людьми.

Личная культура человека определяется его:

- уровнем интеллектуального развития;
- характером профессиональной и творческой деятельности.

Это означает, что свой личный культурный уровень человек повышает, развивая интеллектуальные навыки, представления, мысли. Именно поэто-

му культурный уровень людей, занятых в сферах науки и творчества, должен быть высоким. Переход к информационному обществу несет с собой возникновение еще одной категории общечеловеческой культуры – *информационной культуры.*

> Информационная культура – это навыки использования компьютерной техники и современных технических средств и методов для целенаправленной работы с информацией, ее приема, обработки и распространения.

Под информационной культурой человека понимается следующее:

– освоение навыков использования различных технических устройств, от телефона до персонального компьютера и компьютерных сетей;
– способность к освоению информационных технологий;
– навыки получения информации из периодической печати и электронных коммуникаций;
– умение представить информацию в четкой и ясной форме и использовать ее с максимальной пользой;
– владение различными методами обработки информации;
– навыки работы с различными видами информации.

Информационные ресурсы. Каждое государство, общество, компания и каждый человек нуждаются в ресурсах для своей жизнедеятельности.

> Ресурсы – это источники и запасы различных средств.

В современном обществе наряду с материальными, сырьевыми, энергетическими, трудовыми и финансовыми ресурсами востребованы также и *информационные ресурсы*.

> К информационным ресурсам относятся научно-технические знания, произведения литературы и искусства и другая информация, имеющая общественно-государственное значение.

Все ресурсы, кроме информационных, расходуются по мере потребления. Например, топливо, сгорая, исчезает, финансовые ресурсы расходуются, и т.д. Информационные же ресурсы неисчерпаемы, их можно применять многократно.

1. Перечислите все информационные революции в истории человечества.
2. Каковы характерные признаки информационного общества?
3. Можно ли назвать информационным то общество, в котором мы сейчас живем?
4. Что такое информационная культура?
5. Что называется информационными ресурсами и в чем их отличие от прочих ресурсов?

4.2. СФЕРЫ ПРИМЕНЕНИЯ КОМПЬЮТЕРНОЙ ТЕХНИКИ

Первая электронно-вычислительная машина была изобретена примерно 60 лет назад. С того времени технология производства компьютеров и программного обеспечения прошла огромный путь развития. Безусловно, компьютеры коренным образом изменили наше общество. Финансы, делопроизводство, промышленность, науку, здравоохранение, образование и другие сферы в наше время просто невозможно представить без компьютеров. Сегодня для обработки информации повсеместно пользуются компьютерами.

Компьютер в образовании. Подготовка высококвалифицированных специалистов – долгий и сложный процесс. Обучение сначала в средней школе, потом в вузе занимает значительную часть человеческой жизни. Между тем в современном информационном обществе знания очень быстро устаревают. Для освоения профессиональных навыков в любой сфере деятельности человеку приходится постоянно пополнять свое базовое образование.

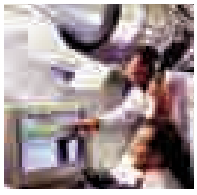
В информационном обществе гораздо важнее знать “как”, нежели “что”. Поэтому в наши дни основная задача школ и вузов состоит не в том, чтобы дать выпускнику как можно большее количество знаний, а в том, чтобы обучить его свободно получать эти знания. А это, в свою очередь, невозможно без применения в образовании современных информационных и коммуникационных технологий.

Одно из важнейших направлений применения информационных и коммуникационных технологий в образовании – использование мультимедийных возможностей компьютера. Использование мультимедийных средств усиливает наглядность, сочетая в себе логическое и образное усвоение информации, а это активизирует учебный процесс. Мультимедийные технологии предоставляют широкие возможности для построения интерактивных личностно-ориентированных учебных моделей.

Используя информационно-коммуникационные технологии, можно организовать дистанционное обучение. При дистанционном обучении педагог и учащийся находятся далеко друг от друга – учебный процесс осуществляется посредством телекоммуникаций, прежде всего сети Интернет.

Благодаря дистанционному обучению многие люди (взрослые люди, занятые профессиональными и семейными заботами, молодежь из сел и небольших городов) имеют возможность повысить свое образование на дому.

Компьютер в научных исследованиях. Современные научные исследования в различных областях знаний, проводимые крупными коллективами ученых, инженеров и конструкторов, требуют очень сложного и

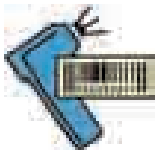


дорогостоящего оборудования. Эффективность научных разработок зависит в первую очередь от степени использования компьютерной техники. Компьютеры применяются для проведения расчетов, анализа результатов экспериментов, подготовки документов и т.д. В результате:

- во много раз сокращаются сроки проведения исследований;
- повышается точность и достоверность результатов;
- усиливается контроль над ходом экспериментов;
- увеличение числа контролируемых параметров и более точная обработка данных повышают качество и информативность экспериментов;
- итоги экспериментов выдаются оперативно и в удобной форме (например, в виде графиков, диграмм и т.д.).

Компьютер в здравоохранении. В современном обществе роль компьютеров в здравоохранении растет день ото дня. Врачи используют компьютеры в различных целях. Перечислим некоторые из них:

1. Установление диагноза, проведение осмотров и профилактических проверок;
2. Поиск доноров для трансплантации органов через компьютерные сети;
3. Возможность держать медработников в курсе новейших научных и практических достижений благодаря банку медицинских данных;
4. Определение влияния загрязнения воздуха на распространение болезней;
5. Подкрепление практических навыков медработников. При этом компьютер выступает в качестве больного, которому требуется немедленная помощь. На основании симптомов, выданных компьютером, обучающийся должен определить курс лечения. Если он ошибся, компьютер сразу показывает это;
6. Создание карт, показывающих скорость распространения эпидемий;
7. Хранение истории болезни пациентов, что освобождает врачей от бумажной работы, на которую уходит много времени, и позволяет больше времени уделять самим больным.



Компьютер в торговле. В большинстве супермаркетов на каждой единице товара имеется этикетка с *бар-кодом* (штрих-кодом), представляющим собой строку вертикальных линий разной толщины. При оплате бар-код

считывается сканером, после чего компьютер, к которому подсоединен этот сканер, сверяет данный бар-код с прайс-листом (расценками на товар) и выдает цифру нужной суммы на экран кассового аппарата. В магазинах и на складах, применяющих систему бар-кодов, информация о каждой разновидности товара хранится в *базе данных.*

Бар-код содержит в себе информацию о товаре и его производителе.

К числу наиболее распространенных типов бар-кодов относится 13-разрядный европейский EAN-13 (European Article Numbering) и применяемый в США и Канаде код UPC.

Структура кода EAN-13

1. Код страны.
2. Код производителя.
3. Код товара.
4. Регистрационная цифра.
5. Пометка о том, что данный продукт является лицензионным.

Количество цифр в коде страны и коде производителя может быть различным. Код Азербайджана в этой системе – 476.

Одно из важнейших направлений информатизации – переход денежно-кредитной и финансовой сферы на *электронную систему оплаты.*

Безналичная торговля. В торговой сфере все большее распространение получает оплата при помощи *кредитных карт*. При использовании кредитной карты нужная сумма автоматически переводится с банковского счета покупателя на банковский счет магазина.

Система безналичной торговли POS (Post of Sale System) выполняет следующие функции:

- верификация, то есть удостоверение подлинности кредитных карт;
- снятие денег со счета покупателя;
- перевод денег на счет продавца.

Информация на кредитной карте нанесена при помощи магнитной записи. Для хранения информации на кредитной карте используется магнитная карта.

На магнитной карте хранится следующая информация:

- номер личного счета;
- название банка;
- название страны;
- категория платежеспособности клиента;
- количество выданных карт и пр.

Банкоматы. Одна из наиболее распространенных сетей в мире – сеть машин для *наличного расчета*, то есть *банковских автоматов, или банкоматов* (ATM – Automated Teller Machine). Компьютеры тысяч банков по всему миру соединены между собой. Банкоматы предназначены для клиентов, имеющих банковские кредитные карточки.

Обладая кредитной картой, выданной каким-либо из банков Азербайджана, можно получить в банкомате наличные деньги, даже находясь за рубежом. Вот принцип работы банкомата:

1. Клиент вставляет карту в прорезь банкомата и вводит с его клавиатуры свой личный шифр.

2. Банкомат считывает информацию с магнитной ленты карты – имя владельца, номер счета и т.д.

3. Банкомат (точнее, его компьютер) посредством телефонной линии пересылает информацию на центральный компьютер, где хранится информация о тысячах банках.

4. Центральный компьютер проверяет счет, после чего посылает на банкомат информацию, либо разрешающую выдать наличность, либо вынуждающую отклонить запрос.

Компьютер в сельском хозяйстве. Если у фермера имеется компьютер, он может быстро и легко производить подсчеты необходимого количества семян для посева и нужного объема удобрений. С помощью компьютерной системы можно планировать сроки посевов, рассчитывать график полива, управлять кормлением скота и производить множество других полезных операций.

Технологическая революция в сельском хозяйстве происходит буквально на наших глазах: с помощью компьютеров и персональных микросенсоров стало возможным контролировать состояние каждого животного или растения, что, в свою очередь, позволяет экономить финансовые и человеческие ресурсы, повышая таким образом уровень жизни людей.

1. Как вы себе представляете применение ИКТ в образовании?
2. Что такое дистанционное обучение?
3. Для каких целей применяются компьютеры в здравоохранении?
4. Что такое бар-код и какова его структура?
5. Опишите принцип действия банкомата.
6. Что такое кредитная карта и каково ее устройство?

ПРОГРАММА

IX класс

(1 час в неделю, итого 32 часа)

I. ЯЗЫК ПРОГРАММИРОВАНИЯ PASCAL (21 час)

Классификация программного обеспечения. Языки программирования. Языки высокого уровня. Разработка программ. Редактор Turbo Pascal. Общая структура программы. Операторы. Операторы выбора. Операторы цикла. Массивы. Работа со строками. Подпрограммы. Файлы.

II. ЭЛЕКТРОННЫЙ ДОКУМЕНТ (4 часа)

Текстовой документ и его объекты. Создание текстового документа. Редактирование документа. Форматирование документа.

III. ТАБЛИЧНЫЙ ПРОЦЕССОР (5 часов)

Назначение табличного процессора. Объекты электронной таблицы. Данные электроннной таблицы. Формулы. Создание и редактирование диаграмм. Форматирование табличного документа.

IV. ИНФОРМАЦИОННОЕ ОБЩЕСТВО (2 часа)

Информатизация общества. Сферы применения компьютерной техники.

ОГЛАВЛЕНИЕ

Исмаил Джалал оглу Садыгов
Рамин Алиназим оглу Махмудзаде
Наида Ризван гызы Исаева

Информатика – **9.** Учебник для 9-го класса общеобразовательной школы. Баку, “Bakınəşr”, 2010, 128 с.

Формат 70×100 1/16. Офсетная бумага №1. Ф/п.л. 8,0. Подписано в печать. 25.05.2010. Гарнитура “Tims New Roman”. Тираж 10000. Бесплатно.