ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет автоматики и вычислительной техники

Кафедра электропривода и автоматизации промышленных установок

В.С. Грудинин, С.А. Мокрушин

Компьютерная обработка данных

Учебное пособие

Дисциплина «Компьютерная обработка данных»
Специальность 140604,
1,2 курс, д/о

Киров 2012

Печатается по решению редакционно-издательского совета Вятского государственного университета

УДК 621.377.037.3

Рецензент: кандидат технических наук, доцент кафедры РЭС
Н.А. Краев

Грудинин В.С., Мокрушин С.А. Компьютерная обработка данных: Учебное пособие. – Киров: Изд-во ВятГУ, 2012. – 100 с.

Авторская редакция

Подписано в печать Усл. печ. л.
Бумага офсетная Печать копир Aficio 1022
Заказ № Тираж Бесплатно
Текст напечатан с оригинал - макета, представленного автором

610 000 , г. Киров, ул. Московская, 36
Оформление обложки, изготовление – ПРИП ВятГУ

Информация и формы ее представления

С понятием информации связаны такие понятия, как сигнал, сообщение и данные.

- Сигнал (от латинского signum — знак) представляет собой любой процесс, несущий информацию.
- Сообщение — это информация, представленная в определенной форме и предназначенная для передачи по каналу связи.
- Данные — это информация, представленная в формализованном виде и предназначенная для обработки ее техническими средствами, например компьютером.

Различают две формы представления информации — непрерывную и дискретную. Сигнал является непрерывным, если его параметр в заданных пределах может принимать любые промежуточные значения. Сигнал называется дискретным, если в заданных пределах может принимать только отдельные фиксированные значения. Дискретность сигнала может быть по уровню и по времени. Различные виды информационных сигналов приведены на рис.1. Графики сигналов представляют: а) непрерывный по уровню и во времени Хнн; б) дискретный по уровню и непрерывный во времени сигнал Хдн; в) непрерывный по уровню и дискретный во времени сигнал Хнд; г) дискретный по уровню и по времени Хдд.

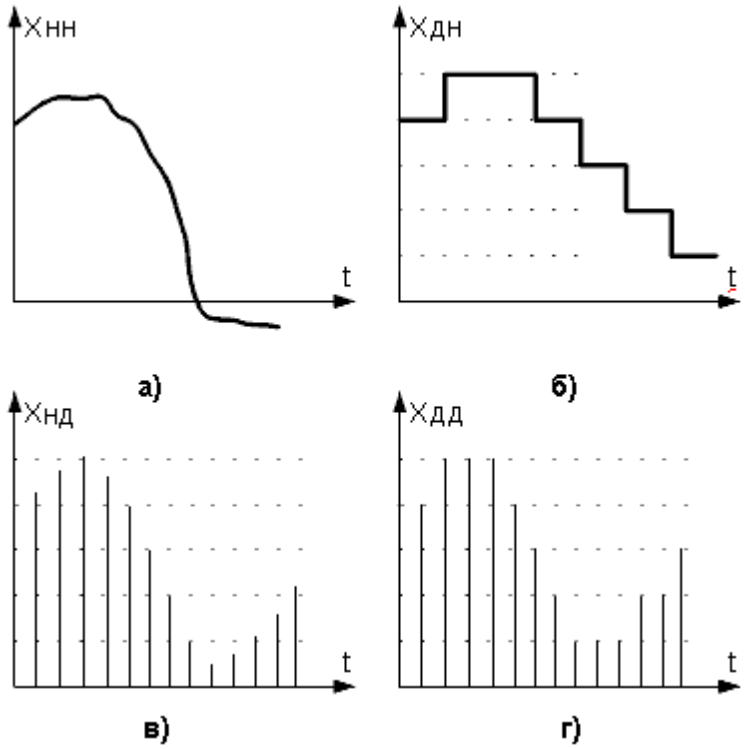


Рис.1. Виды информационных сигналов

По способу передачи и восприятия различают следующие виды информации: физическую — передаваемую физическими сигналами, например током и напряжением, визуальную — передаваемую видимыми образами и символами, аудиальную — звуками, тактильную - ощущениями, органолептическую - запахами и вкусом, машинную - выдаваемую и воспринимаемую средствами вычислительной техники, и т. д.

Понятие количества информации

Количеством информации называют числовую характеристику сигнала, отражающую

ту степень неопределенности (неполноту знаний), которая исчезает после получения сообщения в виде данного сигнала. Меру неопределенности в теории информации называют энтропией. Если в результате получения сообщения достигается полная ясность в каком-то вопросе, говорят, что была получена полная или исчерпывающая информация и необходимости в получении дополнительной информации нет. И, наоборот, если после получения сообщения неопределенность осталась прежней, значит, информации получено не было (нулевая информация).

Количество информации, которое можно получить при ответе на вопрос типа «да-нет», называется *битом.* Бит — минимальная единица количества информации, ибо получить информацию меньшую, чем 1 бит, невозможно. При получении информации в 1 бит неопределенность уменьшается в 2 раза.

Связь между количеством информации и числом состояний системы устанавливается формулой Хартли:

$$i = \log_2 N, \qquad (1)$$

где i — количество информации в битах; N —число возможных состояний.

Группа из 8 битов информации называется *байтом.* Если бит — минимальная единица информации, то байт ее основная единица. Состав байта показан на рис.2., он состоит из бит d_0 … d_7 или двух полубайт — старшего и младшего, которые часто называют ниббламИ. Ниббл (nibble) — единица измерения информации, равная четырём двоичным разрядам (битам), удобна тем, что представима одной шестнадцатеричной цифрой.

Рис.2 Состав байта

Существуют производные единицы информации: килобайт (кбайт, кб), мегабайт (Мбайт, Мб), гигабайт (Гбайт, Гб), терабайт (Тбайт, Тб).

1 кб - 1024 байта - 2^{10} (1024) байт.

1 Мб - 1024 кбайта - 2^{20} (1024 * 1024) байт.

1 Гб - 1024 Мбайта - 2^{30} (1024 * 1024 * 1024) байт.

1 Тб — 1024 Гбайта — 2^{40} (1024* 1024*1024*1024) байт.

Эти единицы чаще всего используют для указания объема памяти ЭВМ.

Архитектура ЭВМ

Архитектура любой ЭВМ определяет правила взаимодействия составных частей вычислительного средства, описание которых выполняется в той мере, в какой это необходимо для формирования правил их взаимодействия. В основе любой ЭВМ лежит процессор.

Большинство современных процессоров для персональных компьютеров в общем основаны на той или иной версии циклического процесса последовательной обработки информации, изобретённого Джоном фон Нейманом, который придумал схему постройки компьютера в 1946 году.

Архитектура процессора по фон Нейману — широко известный принцип совместного хранения программ и данных в памяти компьютера.

Наличие жёстко заданного набора исполняемых команд и программ было характерной

чертой первых компьютерных систем (например настольный калькулятор является устройством с фиксированным набором выполняемых программ). Изменение встроенной программы для такого рода устройств требует практически полной их переделки, и в большинстве случаев невозможно.

Всё изменила идея хранения компьютерных программ в общей памяти. Ко времени её появления использование архитектур, основанных на наборах исполняемых инструкций и представление вычислительного процесса, как процесса выполнения инструкций, записанных в программе, чрезвычайно увеличило гибкость вычислительных систем в плане обработки данных. Один и тот же подход к рассмотрению данных и инструкций сделал лёгкой задачу изменения самих программ.

Отличительной особенностью архитектуры фон Неймана является то, что инструкции и данные хранятся в одной и той же памяти.

Этапы цикла выполнения программы компьютером:

- Процессор выставляет число, хранящееся в регистре счётчика команд, на шину адреса, и отдаёт памяти команду чтения;
- Выставленное число является для памяти адресом; память, получив адрес и команду чтения, выставляет содержимое, хранящееся по этому адресу, на шину данных, и сообщает о готовности;
- Процессор получает число с шины данных, интерпретирует его как команду (машинную инструкцию) из своей системы команд и исполняет её с помощью арифметическо — логического устройства АЛУ;
- Если последняя команда не является командой перехода, процессор увеличивает на единицу (в предположении, что длина каждой команды равна единице) число, хранящееся в счётчике команд; в результате там образуется адрес следующей команды;
- Снова выполняется переход на начало цикла.

Данный цикл выполняется неизменно, и именно он называется процессом (откуда и произошло название устройства).

Во время процесса процессор считывает последовательность команд, содержащихся в памяти, и исполняет их. Такая последовательность команд называется программой и представляет алгоритм полезной работы процессора. Очерёдность считывания команд изменяется в случае, если процессор считывает команду перехода — тогда адрес следующей команды может оказаться другим. Другим примером изменения процесса может служить случай получения команды останова или переключение в режим обработки аппаратного прерывания.

Команды центрального процессора являются самым нижним уровнем управления компьютером, поэтому выполнение каждой команды неизбежно и безусловно. Не производится никакой проверки на допустимость выполняемых действий, в частности, не проверяется возможная потеря ценных данных. Чтобы компьютер выполнял только допустимые действия, команды должны быть соответствующим образом организованы в виде необходимой программы.

Скорость перехода от одного этапа цикла к другому определяется тактовым генератором. Тактовый генератор вырабатывает импульсы, служащие ритмом для центрального процессора. Частота тактовых импульсов называется тактовой частотой.

Тактовая частота - величина, характеризующая количество операций, выполняемых процессором за единицу времени. Под выполнением операции подразумевают обработку одного логического 0 или единицы. Тактовая частота процессора зависит не только от его собственной частоты, а и от размера/частоты кеш памяти (СОЗУ — сверхоперативной памяти), ширины общей шины и архитектуры компьютера. Структурная схема современного персонального компьютера, основанного на архитектуре фон Неймана, приведена на рис.3.

В нее входит множество дополнительных изобретений, значительно улучшающих как

вычислительные возможности, так и эргономику компьютера. Это математический сопроцессор, расширяющий вычислительные способности основного процессора, мультипроцессорное решение, мощная видеокарта, сетевой адаптер, позволяющий связывать компьютеры по сети интернет, часы с батареей, позволяющие работать в реальном времени, манипулятор — мышь, DVD, USB и др.

Рис.3. Структурная схема современного персонального компьютера

Кроме архитектуры по фон Нейману, в современных компьютерных системах применяется и другая архитектура — Гарвардская, которая основана на раздельной памяти данных и программ. Это решение позволяет повысить быстродействие системы за счет распараллеливания процесса выполнения программ (до десяти раз!).

Классификация ЭВМ

Чтобы судить о возможностях ЭВМ, их принято разделять на группы по определенным признакам, т. е. классифицировать. Признаком классификации может служить производительность, стоимость, элементная база и т. д.

С точки зрения классификации ЭВМ по таким показателям, как габариты и производительность, можно представить следующие группы:

- сверхпроизводительные ЭВМ и системы (супер-ЭВМ);
- большие ЭВМ (универсальные ЭВМ общего назначения);
- средние ЭВМ;
- малые или мини-ЭВМ, персональные компьютеры;
- микро-ЭВМ;
- микропроцессоры.

Исторически первыми появились большие **ЭВМ** (универсальные ЭВМ общего назначения), элементная база которых прошла путь от электронных ламп до схем со

сверхвысокой степенью интеграции. В процессе эволюционного развития больших ЭВМ можно выделить отдельные периоды, связываемые с пятью поколениями ЭВМ. Поколение ЭВМ определяется элементной базой (лампы, полупроводники, микросхемы различной степени интеграции), архитектурой и вычислительными возможностями.

Основное назначение больших (кластерных) ЭВМ — выполнение работ, связанных с обработкой и хранением больших объемов информации, проведением сложных расчетов и исследований в ходе решения вычислительных и информационно-логических задач.

Производительность больших ЭВМ порой оказывается недостаточной для ряда приложений, например, таких как прогнозирование метеообстановки, ядерная энергетика, оборона и т. д. Эти обстоятельства стимулировали создание сверхбольших или супер-ЭВМ. Такие машины обладают колоссальным быстродействием в миллиарды операций в секунду, основанном на выполнении параллельных вычислений и использовании многоуровневой иерархической структуры ЗУ (запоминающих устройств), требуют для своего размещения специальных помещений и крайне сложны в эксплуатации. Быстродействие супер-ЭВМ измеряется во флопсах (акроним от англ. FLoating-point Operations Per Second) — внесистемной единице, используемой для измерения производительности компьютеров и показывающей, сколько операций с плавающей запятой в секунду выполняет данная вычислительная система. В настоящее время быстродействие оценивается в петафлопсах 10^{15} операций в секунду.

Стоимость отдельной ЭВМ такого класса достигает десятков миллионов долларов. Представители этого класса ЭВМ — компьютеры фирм Cray Research, Control Data Corporation (CDC) n отечественные супер-ЭВМ семейства Эльбрус.

Например, один из самых мощных компьютеров в мире - IBM ROADRUNNER имеет производительность 1,026 петафлоп (1.026 квадрильона операций в секунду), состоит из 12960 процессоров Cell + 6948 AMD Opteron, имеет оперативной памяти 80 ТБ, потребляет 4 МВт электроэнергии, весит 225 тонн и стоит 133 миллиона долларов.

Средние ЭВМ представляют некоторый интерес в историческом плане. На определенном этапе развития ЭВМ, когда их номенклатура и, соответственно, возможности были ограниченными, появление средних машин было закономерным. Вычислительные машины этого класса обладают несколько меньшими возможностями, чем большие ЭВМ, но зато им присуща более низкая стоимость. К средним могут быть отнесены некоторые модели фирмы IBM (International Business Machinery), DEC (Digital Equipment Corporation), Hewlett Packard, COMPAREX и др.

Малые ЭВМ — это современные персональные компьютеры и ноутбуки, включая планшеты, нетбуки и субноутбуки.

К микро-ЭВМ можно отнести разнообразные наладонники, коммуникаторы, i – PAD и другие устройства связи с операционной системой типа Android.

Микропроцессоры — это своего рода миниатюрный компьютер, выполненный в виде отдельной микросхемы, но по степени интеграции имеющий внутри себя все признаки настоящего компьютера. Это такие микропроцессоры как Atmel, PIC, ARM, MSP430 и др.

Системы счисления

Различают позиционные и непозиционные системы счисления. В непозиционных системах счисления каждое число обозначается соответствующей совокупностью символов. Характерным представителем непозиционных систем является римская система счисления со сложным способом записи чисел и громоздкими правилами выполнения арифметических операций. Например, запись MCMXCVIII означает, что записано число 1998 (M — тысяча, C — сто, X — десять, V — пять, I — единица и т. д.).

Позиционные системы счисления обладают большими преимуществами в наглядности представления чисел и в простоте выполнения арифметических операций.

В позиционной системе счисления значение числа определяется не только набором входящих в него цифр, но и их местом (позицией) в последовательности цифр, изображающих это

число, например, числа 125 и 521.

Позиционной является десятичная система счисления, используемая в повседневной жизни. Помимо десятичной существуют другие позиционные системы счисления, некоторые из них нашли применение в информатике.

Количество символов, используемых в позиционной системе счисления, называется ее основанием. Его обозначают обычно буквой q. В десятичной системе счисления используется десять символов (цифр): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, и основанием системы является число десять.

Особое место среди позиционных систем счисления занимают системы со степенными весами разрядов, в которых веса смежных позиций цифр (разрядов) отличаются по величине в постоянное количество раз, равное основанию q системы счисления.

В общем случае в такой позиционной системе счисления с основанием q любое число X может быть представлено в виде полинома разложения:

$$X_{(q)} = x_{n-1}q^{n-1}+x^{n-2}q^{n-2}+...+x_1q^1 + x_0q^o + x_{-1}q^{-1}+ ...+x_{-m}q^{-m} = \text{сумма}, \quad (2)$$

где $X_{(q)}$ — запись числа в системе счисления с основанием q;
q — основание системы счисления;
X_i — целые числа, меньше q;
n — число разрядов (позиций) в целой части числа;
m — число разрядов в дробной части числа.

Например: $4295,6731_{(10)} = 4*10^3+ 2*10^2 + 9*10^1 +5*10^\circ +6*10^{-1} + 7*10^{-2} + 1*10^{-4}$.

Для обозначения используемой системы счисления ее основание указывается в индексе в круглых скобках. Изображение числа X в виде последовательности коэффициентов x полинома является его условной сокращенной записью (кодом).

$$X_{(q)} = X_{n-1}X_{n-2} \ldots X_1X_{-1}...X_{-m} \qquad (3)$$

Запятая отделяет целую часть числа от дробной и служит началом отсчета значений веса каждой позиции (разряда).

В информатике применяют позиционные системы счисления с недесятичным основанием: двоичную, восьмеричную и шестнадцатиричную, т. е. системы счисления с основанием

$q = 2^k$, где к = 1, 3, 4.

Таблица 1

Десятичное	Двоичное	Восьмиричное	Шестнадцати-ричное	Двоично - десятичное
0	0000	00	00	00000000
1	0001	01	01	00000001
2	0010	02	02	00000010
3	0011	03	03	00000011
4	0100	04	04	00000100
5	0101	05	05	00000101
6	0110	06	06	00000110
7	0111	07	07	00000111
8	1000	10	08	00001000
9	1001	11	09	00001001
10	1010	12	0A	00010000
11	1011	13	0B	00010001
12	1100	14	0C	00010010
13	1101	15	0D	00010011
14	1110	16	0E	00010100

15	1111	17	0F	00010101

Наибольшее распространение получила двоичная система счисления, см. таблицу 1. В этой системе для представления любого числа используются два символа — цифры 0 и 1. Основание системы счисления q = 2.

Произвольное число с помощью формулы (2) можно представить в виде разложения по степеням двойки:

$$X_{(2)} = X_n\text{-}1*2^{n-1}+X_{n-2}*2^{n-2}+ \dots +X_1*2^1 + X_0*2^\circ + X_{-1}*2^{-1} + \dots +X_{-m}2^{-m}$$

Тогда условная сокращенная запись в соответствии с (3) означает изображение числа в двоичной системе счисления (двоичный код числа), где x. = 0 или 1. Например:

$$13,625_{(10))}= 1*2^3 +1*2^2 + 0*2^1 + 1*2^\circ+ 1*2^{-1} +0*2^{-2} + 1*2^{-3}= 1101,101_{(2)}$$

Двоичное представление числа требует примерно в 3,3 раза большего числа разрядов, чем его десятичное представление. Тем не менее, применение двоичной системы счисления создает большие удобства для работы ЭВМ, т. к. для представления в машине разряда двоичного числа может быть использован любой запоминающий элемент, имеющий два устойчивых состояния.

Арифметические действия над одноразрядными двоичными числами выполняются по следующим правилам:

0+0=0	0*0=0
0+1=1	0*1=0
1+0=1	1*0=0
1 + 1 = 10	1*1=1.

В *восьмеричной* системе счисления алфавит состоит из восьми символов (цифр): 0, 1 ... 7. Основание системы счисления q = 8. Для записи произвольного числа в восьмеричной системе счисления необходимо по формуле (2) найти его разложение по степеням восьмерки, а затем воспользоваться условной сокращенной записью.

Например, десятичное число $28_{(10)} = 3*8^1 + 4*8^\circ = 34_{(8)}$

Восьмеричное изображение (код) основания системы счисления $q = 8_{(10)} = 10_{(8)}$

В шестнадцатеричной системе счисления алфавит включает в себя 16 символов (цифр и букв) : 0, 1 ... 9, A, B, C, D, E, F. Основание системы счисления q = 16. Для записи произвольного числа в этой системе счисления необходимо по формуле (2) найти его разложение по степеням 16, а по формуле (3) — код.

Например: $75_{(10)}= 4*16^1 + B*16^\circ = 4B_{(16)}$.

Шестнадцатеричное изображение (код) основания системы счисления $q = 16_{(10)} =10_{(16)}$.

Наряду с двоичными кодами, которыми оперирует ЭВМ, для ввода и вывода десятичных чисел (данных) используют специальное двоично-десятичное кодирование. При двоично-десятичном кодировании каждая десятичная цифра заменяется тетрадой (четверкой) двоичных цифр, а сами тетрады записываются последовательно в соответствии с порядком следования десятичных цифр. При обратном преобразовании двоично-десятичного кода в десятичный исходный код разбивается на тетрады вправо и влево от запятой, которые затем заменяются десятичными цифрами.

Таким образом, при двоично-десятичном кодировании фактически не производится перевод числа в новую систему счисления, а мы имеем дело с двоично-кодированной десятичной системой счисления.

Например, десятичное число $15_{(10)} = F_{(16)} = 17_{(8)} = 1111_{(2)} = 00010101_{(2\text{-}10)}$.

Преобразование чисел

ЭВМ работают с двоичными кодами, а пользователю удобнее иметь дело с десятичными или шестнадцатеричными. Поэтому возникает необходимость перевода числа

из одной системы счисления в другую.

Преобразование числа **X** из системы счисления с основанием **q** в систему счисления с основанием ***p*** (преобразование $X_{(q)} \rightarrow X_{(p)}$ осуществляется по правилу замещения или по правилу деления-умножения на основание системы счисления.

Правило замещения реализуется на основании формулы (2) и предусматривает выполнение арифметических операций с кодами чисел в новой системе счисления. Поэтому оно чаще всего используется для преобразования чисел из недесятичной системы счисления в десятичную.

Пример. Выполнить преобразование $X_{(2)} \rightarrow X_{(10)}$, если $X_{(2)} = 10101{,}011$

$X_{(10)} = 1*2^4 + 0*2^3 + l\text{-}2^2 + 0*2^1 + l*2° + 0*2^{-1} + l*2^{-2} + l*2^{-3} = 21{,}375.$

Правило деления-умножения предусматривает выполнение арифметических операций с кодами чисел в исходной системе счисления с основанием **q**, поэтому его удобно применять для преобразования десятичных чисел в любые другие позиционные системы счисления. Правила преобразования целых чисел и правильных дробей различны. Для преобразования целых чисел используется правило деления, а для преобразования правильных дробей — правило умножения. Для преобразования смешанных чисел используются оба правила соответственно для целой и дробной частей числа.

Правило деления используется для преобразования целого числа, записанного в q-ичной системе счисления, в p-ичную. В этом случае необходимо последовательно делить исходное q-ичное число и получаемые частные на новое основание p, представленное в q-ичной системе счисления. Деление продолжают до тех пор, пока очередное частное не станет меньше p. После замены полученных остатков и последнего частного цифрами p-ичной системы счисления записывается код числа в новой системе счисления. При этом старшей цифрой является последнее частное, а следующие за ней цифры соответствуют остаткам, записанным в последовательности, обратной их получению.

Правило умножения используется для преобразования дробного числа, записанного в q-ичной системе счисления, в p-ичную. В этом случае необходимо последовательно умножать исходную дробь и дробные части получающихся произведений на основание p, представленное в исходной q-ичной системе счисления. Целые числа получаемых произведений, замененные цифрами p-ичной системы счисления, и дают последовательность цифр в новой p-ичной системе.

Умножение необходимо производить до получения в искомом p-ичном коде цифры того разряда, вес которого меньше веса младшего разряда исходной q-ичной дроби. При этом в общем случае получается код приближенно, и всегда с недостатком значения дроби. Поэтому в случае обратного преобразования (p-ичного кода дроби в q-ичный) результат может не совпадать с исходным значением q-ичной дроби.

Пример. Выполнить преобразование $X_{(10)} \rightarrow X_{(2)}$, если $X_{(10)} = 37{,}45$

Для получения частных и остатков для целой части числа удобно использовать «правило деления», а для получения p-ичного кода дробной части числа — форму записи, известную как «правило умножения». Для целой части примера имеем:

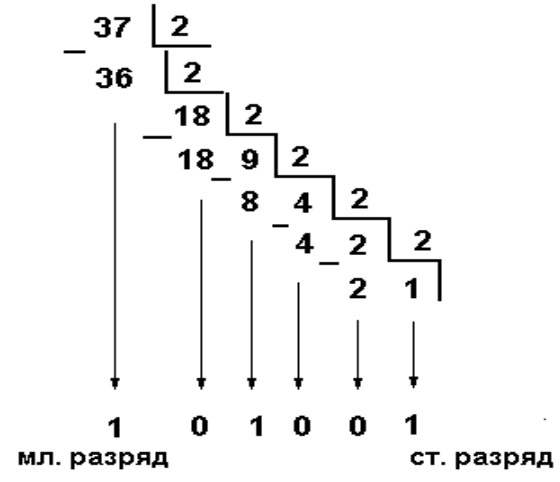


Рис.4 Преобразование целой части числа

То есть получили целое число 100101. Теперь для дробной части имеем:

0,45	0,90	0,8	0,6
* 2	* 2	* 2	* 2
0,90	1,8	1,6	1,2
0, 0	1	1	1 . . .

Рис.5.Преобразование дробной части числа

То есть получили дробное число 0,0111... Итак, в результате преобразования получаем 37,45 = 100101,0111... $_{(2)}$ Как следует из примера, процесс перевода дробной части можно продолжить до бесконечности. ЭВМ оперирует числами, представленными конечными наборами цифр, поэтому дроби округляют в соответствии с правилами преобразования и весом младшего разряда исходной дроби.

Преобразование чисел из двоичной системы счисления в восьмеричную, шестнадцатеричную и обратно осуществляется по упрощенным правилам с учетом того, что основания этих систем счисления кратны целой степени 2, т. е. $8=2^3$, а $16=2^4$. Это означает, что при преобразовании восьмеричного кода числа в двоичный, необходимо каждую восьмеричную цифру заменить трехзначным двоичным кодом (триадой), а при преобразовании шестнадцатеричного кода числа в двоичный необходимо каждую шестнадцатеричную цифру заменить четырехзначным двоичным кодом (тетрадой).

При преобразовании двоичного кода в восьмеричный или шестнадцатеричный двоичный код делится соответственно на триады или тетрады влево и вправо от запятой, разделяющей целую и дробные части числа. Затем триады (тетрады) заменяются восьмеричными (шестнадцатеричными) цифрами.

Например: $1CD,4_{(16)} = 000111001101,0100_{(2)} = 715,2_{(8)}$.

Если при разбиении двоичного кода в крайних триадах (тетрадах) недостает цифр до нужного количества, они дополняются нулями. Соответственно, «лишние» нули слева и справа, не вошедшие в триады (тетрады) отбрасываются.

Формы представления данных в ЭВМ

В ЭВМ используются следующие формы представления данных:

- числа с фиксированной точкой (запятой);
- числа с плавающей точкой (запятой);
- десятичные числа;
- символьные данные.

При представлении числа X в форме с фиксированной точкой указываются знак числа (sign X) и модуль числа (mod X) в q-ичном коде. Иногда такую форму представления чисел называют естественной формой. Место точки (запятой) постоянно для всех чисел и в процессе решения задач не меняется. Знак положительного числа кодируется цифрой «0», а знак отрицательного числа — цифрой «1».

Код числа в форме с фиксированной точкой, состоящий из кода знака и q-ичного кода его модуля, называется прямым кодом q-ичного числа. Разряд прямого кода числа, в котором располагается код знака, называется знаковым разрядом кода. Разряды прямого кода числа, в которых располагается q-ичный код модуля числа, называются цифровыми разрядами кода.

При записи прямого кода знаковый разряд располагается левее старшего цифрового

разряда и обычно отделяется от цифровых разрядов точкой.

Пример разрядной сетки ЭВМ для размещения чисел в форме с фиксированной точкой имеет следующий вид:

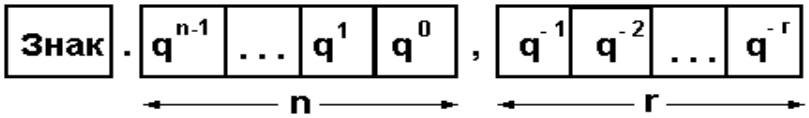


Рис.6 Формат числа с фиксированной точкой

На рисунке 6 показано **n** разрядов для представления целой части числа и **r** разрядов — для дробной части числа. Данная форма представления чисел в ЭВМ практически не используется, за исключением микроконтроллеров, где требуется получить максимальную скорость выполнения программы, но при этом инженеру — программисту необходимо следить за возможным переполнением разрядной сетки.

При заданных **n** и **r** диапазон изменения чисел, коды которых могут быть представлены в данной разрядной сетке, определяется соотношением:

$$q^{-r} < |X| < q^{n} - q^{-r}$$

От этих недостатков в значительной степени свободны ЭВМ, использующие форму представления чисел с плавающей точкой, или нормальную форму.

В нормальной форме число представляется в виде произведения $X = m*q^{p}$,

где m — мантисса числа;

q — основание системы счисления;

p — порядок.

Для задания числа в нормальной форме требуется задать знаки мантиссы и порядка, их модули в q-ичном коде, а также основание системы счисления. Нормальная форма представления чисел неоднозначна, ибо взаимное изменение m **и p** приводит к плаванию точки (запятой). Отсюда произошло название формы представления чисел.

Например, десятичное число 73,28 в нормальной форме может быть записано в следующих вариантах:

$X_{(10)} = 73{,}28*10^{\circ} = 7{,}328*10^{1} = 0{,}7328*10^{2} = 0{,}07328\text{-}10^{3}$ и т. д.

Для однозначности представления чисел в **ЭВМ** используется ***нормальная нормализованная форма,*** в которой положение точки всегда задается перед значащей цифрой мантиссы, т. е. выполняется условие

$$q^{-1} < |m| < q^{\circ} = 1 .$$

В общем случае разрядную сетку ЭВМ для размещения чисел в нормальной форме **можно** представить в виде, изображенном на рис. 7.

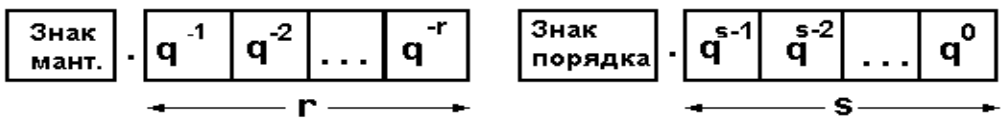


Рис.7. Представление числа в нормальной форме

Разрядная сетка содержит:

- разряд для знака мантиссы;
- r цифровых разрядов для q-ичного кода модуля мантиссы;
- разряд для кода знака порядка;

- s разрядов для q-ичного кода модуля порядка.

В конкретной **ЭВМ** диапазон представления чисел с плавающей точкой зависит от основания системы и числа разрядов для представления порядка.

Точность вычислений при использовании формата с плавающей точкой определяется числом разрядов мантиссы. Она увеличивается с увеличением числа разрядов.

При представлении информации в виде десятичных многоразрядных чисел каждая десятичная цифра заменяется двоично-десятичным кодом. Для ускорения обмена информацией, экономии памяти и удобства операций над десятичными числами часто используются специальные форматы их представления: ***зонный*** и ***упакованный***. Зонный формат используется в операциях ввода-вывода десятичных данных, упакованный формат — для выполнения арифметических операций. Для этого в **ЭВМ** имеются специальные команды упаковки и распаковки десятичных чисел.

В ***зонном*** формате для представления десятичных цифр используется двоичный код обработки информации. Каждая цифра в этом коде занимает байт, причем в левую тетраду (зону) записывается код 1111 = F.

Для обработки символьной информации в ЭВМ используется двоичный код обработки информации — ДКОИ, вариантом которого является КОИ-7, в котором для кодирования используется только 7 разрядов. Другим вариантом кодирования является американский код обмена информацией — ASCII.

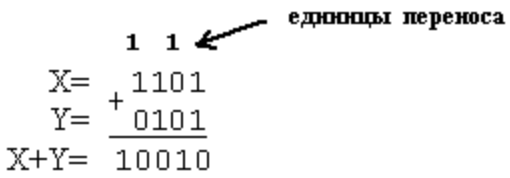
Двоичная арифметика

Так как процессор компьютера предназначен для выполнения математических операций, рассмотрим несколько примеров того, как это происходит с битами двоичных чисел.

- **Сложение двоичных чисел**

При сложении двоичных чисел в каждом разряде производится сложение цифр слагаемых и переноса из соседнего младшего разряда, если он имеется. При этом необходимо учитывать, что 1+1 дают нуль в данном разряде и единицу переноса в следующий.

Пример. Выполнить сложение четырехразрядных двоичных чисел:
X=1101, Y=0101; Вычислить Z = X+Y.

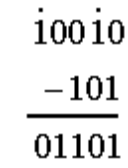


Результат Z = 1101+101=10010.

- **Вычитание двоичных чисел**

При вычитании двоичных чисел в данном разряде при необходимости занимается 1 из старшего разряда. Эта занимаемая 1 равна двум 1 данного разряда.

Пример. Заданы двоичные числа X=10010 и Y=101. Вычислить Z = X-Y.



Результат Z = 10010 - 101=1101.

- **Вычитание двоичных чисел с помощью дополнительного кода**

Реально в структуре процессора вычитатель как таковой отсутствует, а операция вычитания производится с помощью сумматора. Чтобы такое стало возможным, необходимо перевести вычитаемое число в дополнительный код. Дополнительный код числа получается путем его инверсии с добавлением единицы к младшему разряду. Если взять предыдущий пример, то вычитаемое число было в пятиразрядном представлении 00101, тогда его дополнительный код 11010 + 1 = 11011. Отсюда операция Z = X-Y заменяется на
Z = X+доп.код(Y):

```
  10010
+ 11011
-------
1 01101
  ↖отбрасывается
```

Как мы видим, результат, если отбросить старшую единицу, как не вошедшую в разрядную сетку, аналогичен.

- **Умножение двоичных чисел**

Умножение двоичных чисел производится по тем же правилам, что и для десятичных с помощью таблиц двоичного умножения и сложения.
Пример. Задано X = 1001 Y = 101, вычислить Z = X*Y.

```
   1001
 ×  101
-------
   1001
  1001
 ------
 101101
```

Результат Z = 1001*101 = 101101.

- **Деление двоичных чисел**

Деление двоичных чисел производится по тем же правилам, что и для десятичных. При этом используются таблицы двоичного умножения и вычитания.
Пример. X = 1100.011, Y = 10.01, вычислить Z = X/Y.

```
_110001.1 | 1001
 1001     |------
  _1101   | 101.1
   1001
   _1001
    1001
       0
```

Результат Z = 1100.011 : 10.01=101.1.

Логические побитовые операции

- **Побитовое отрицание**

Побитовое отрицание (или побитовое НЕ) — это унарная операция, действие которой эквивалентно применению логического отрицания к каждому биту двоичного представления операнда. Другими словами, на той позиции, где в двоичном представлении операнда был 0,

в результате будет 1, и, наоборот, где была 1, там будет 0. Операция обозначается либо чертой сверху, либо знаком восклицания, либо как not. Например, если X = 01011, то

$$\text{not}(X) = !X = \overline{X} = 10100$$

- **Побитовое И**

Побитовое И — это бинарная операция, действие которой эквивалентно применению логического И к каждой паре битов, которые стоят на одинаковых позициях в двоичных представлениях операндов. Другими словами, если оба соответствующих бита операндов равны 1, результирующий двоичный разряд равен 1; если же хотя бы один бит из пары равен 0, результирующий двоичный разряд равен 0. Операция обозначается либо знаком & , либо знаком ^ , либо как and, либо как знак умножения в логических выражениях.

Пример:
```
  01011
^ 11001
  01001
```

- **Побитовое ИЛИ**

Побитовое ИЛИ — это бинарная операция, действие которой эквивалентно применению логического ИЛИ к каждой паре битов, которые стоят на одинаковых позициях в двоичных представлениях операндов. Другими словами, если оба соответствующих бита операндов равны 0, двоичный разряд результата равен 0; если же хотя бы один бит из пары равен 1, двоичный разряд результата равен 1. Операция обозначается либо знаком || , либо знаком V , либо как or, либо как знак сложения в логических выражениях.

Пример:
```
  01011
V 11001
  11011
```

- **Побитовое исключающее ИЛИ**

Побитовое исключающее ИЛИ (или побитовое сложение по модулю два) — это бинарная операция, действие которой эквивалентно применению логического исключающего ИЛИ к каждой паре битов, которые стоят на одинаковых позициях в двоичных представлениях операндов. Другими словами, если соответствующие биты операндов различны, то двоичный разряд результата равен 1; если же биты совпадают, то двоичный разряд результата равен 0. Операция обозначается либо знаком ⊻
, либо как xor.

Пример:
```
  01011
⊻ 11001
  10010
```

В компьютерной графике «исключающее ИЛИ» применяется при выводе спрайтов на картинку — повторное её применение убирает спрайт с картинки. Благодаря инволютивности эта операция нашла применение в криптографии как простейшая реализация идеального шифра (шифра Вернама). «Исключающее ИЛИ» также может использоваться для обмена двух переменных, используя алгоритм обмена при помощи исключающего ИЛИ.

- **Битовые сдвиги**

К битовым операциям также относят битовые сдвиги. При сдвиге значения битов копируются в соседние по направлению сдвига. Различают несколько видов сдвигов — логический, арифметический и циклический, в зависимости от обработки крайних битов.

Также различают сдвиг влево (в направлении от младшего бита к старшему) и вправо

(в направлении от старшего бита к младшему).

При логическом сдвиге значение последнего бита по направлению сдвига теряется (копируясь в специальный дополнительный бит переноса), а первый приобретает нулевое значение.

Логические сдвиги влево и вправо используются для быстрого умножения и деления на 2, соответственно.

Арифметический сдвиг аналогичен логическому, но значение слова считается знаковым числом представленному дополнительным кодом. Так при правом сдвиге старший бит сохраняет свое значение. Левый арифметический сдвиг идентичен логическому.

При циклическом сдвиге, значение последнего бита по направлению сдвига копируется в первый бит (и копируется в бит переноса).

Также различают циклический сдвиг через бит переноса (флаг переноса, находящийся в регистре флагов процессора) — при нём первый бит по направлению сдвига получает значение из бита переноса, а значение последнего бита сдвигается в бит переноса.

Аппаратные основы ЭВМ

В основе процессора ЭВМ лежит АЛУ — арифметическо — логическое устройство. Оно выполняет основные математические операции над двоичными числами, а также и логические. Это устройство выполнено на электронных логических элементах.

Логическими элементами компьютеров являются электронные схемы И, ИЛИ, НЕ, И—НЕ, ИЛИ—НЕ и другие (называемые также вентилями), а также триггер.

С помощью этих схем можно реализовать любую логическую функцию, описывающую работу устройств компьютера. Обычно у вентилей бывает от двух до восьми входов и один или два выхода.

Чтобы представить два логических состояния — “1” и “0” в вентилях, соответствующие им входные и выходные сигналы имеют один из двух установленных уровней напряжения. Например, +5 вольт и 0 вольт.

Высокий уровень обычно соответствует значению “истина” (“1”), а низкий — значению “ложь” (“0”).

Каждый логический элемент имеет свое условное обозначение, которое выражает его логическую функцию, но не указывает на то, какая именно электронная схема в нем реализована. Это упрощает запись и понимание сложных логических схем.

Работу логических элементов описывают с помощью таблиц истинности. Рассмотрим основные простейшие логические элементы.

- **Схема НЕ**

Схема НЕ (инвертор) реализует операцию отрицания. Связь между входом x этой схемы и выходом z можно записать соотношением z = !x или "инверсия x".

Если на входе схемы 0, то на выходе 1. Когда на входе 1, на выходе 0. Условное обозначение на структурных схемах инвертора — на рис. 8.

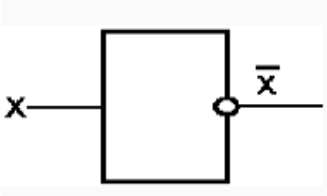


Таблица истинности схемы **НЕ**

x	$\overline{x}$
0	1
1	0

Рис. 8. Обозначение и таблица истинности элемента НЕ.

- **Схема И**

Схема И реализует конъюнкцию двух или более логических значений. Условное

обозначение на структурных схемах схемы И с двумя входами представлено на рис. 9.
Единица на выходе схемы И будет тогда и только тогда, когда на всех входах будут единицы. Когда хотя бы на одном входе будет ноль, на выходе также будет ноль.

Связь между выходом z этой схемы и входами x и y описывается соотношением: z = x . y (читается как "x и y"). Операция конъюнкции на структурных схемах обозначается знаком "&" (читается как "амперсэнд"), являющимся сокращенной записью английского слова and.

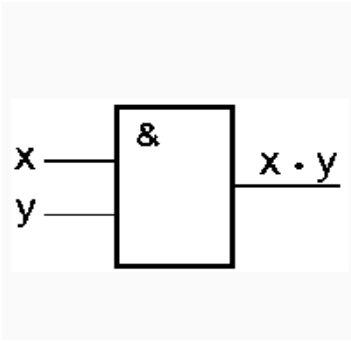


Таблица истинности схемы **И**

x	y	x · y
0	0	0
0	1	0
1	0	0
1	1	1

Рис. 9. Обозначение и таблица истинности элемента И.

- **Схема ИЛИ**

Схема ИЛИ реализует дизъюнкцию двух или более логических значений. Когда хотя бы на одном входе схемы ИЛИ будет единица, на её выходе также будет единица.

Условное обозначение на структурных схемах схемы ИЛИ с двумя входами представлено на рис. 10. Знак "1" на схеме — от устаревшего обозначения дизъюнкции как ">=1" (т.е. значение дизъюнкции равно единице, если сумма значений операндов больше или равна 1). Связь между выходом z этой схемы и входами x и y описывается соотношением: z = x v y (читается как "x или y").

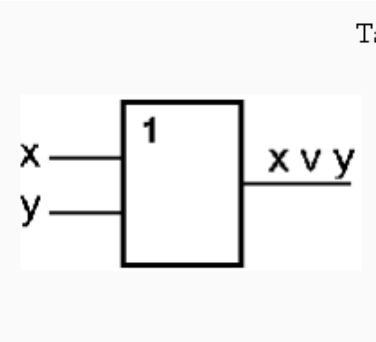


Таблица истинности схемы **ИЛИ**

x	y	x v y
0	0	0
0	1	1
1	0	1
1	1	1

Рис. 10. Обозначение и таблица истинности элемента ИЛИ.

- **Триггер**

Термин триггер происходит от английского слова trigger — защёлка, спусковой крючок. Самый распространённый тип триггера — так называемый RS-триггер (S и R, соответственно, от английских set — установка, и reset — сброс). Он имеет два симметричных входа S и R и два симметричных выхода Q и , причем выходной сигнал !Q является логическим отрицанием сигнала .
На каждый из двух входов S и R могут подаваться входные сигналы в виде кратковременных импульсов. Наличие импульса на входе считается единицей, а его отсутствие — нулем.

На рис.11 показана реализация триггера с помощью вентилей ИЛИ—НЕ и соответствующая таблица истинности. В реализации компьютера триггеры играют роль элементов памяти, так как обладают свойством хранения бит.

Проанализируем возможные комбинации значений входов R и S триггера. Если на

входы триггера подать S="1", R="0", то (независимо от состояния) на выходе Q верхнего вентиля появится "0". После этого на входах нижнего вентиля окажется R="0", Q="0" и выход станет равным "1".

Точно так же при подаче "0" на вход S и "1" на вход R на выходе появится "0", а на Q — "1".

Если на входы R и S подана логическая "1", то состояние Q и не меняется.

Подача на оба входа R и S логического "0" может привести к неоднозначному результату, поэтому эта комбинация входных сигналов запрещена.

Поскольку один триггер может запомнить только один разряд двоичного кода, то для запоминания байта нужно 8 триггеров, для запоминания килобайта, соответственно, 8 * 210 = 8192 триггеров. Современные микросхемы памяти содержат миллионы триггеров.

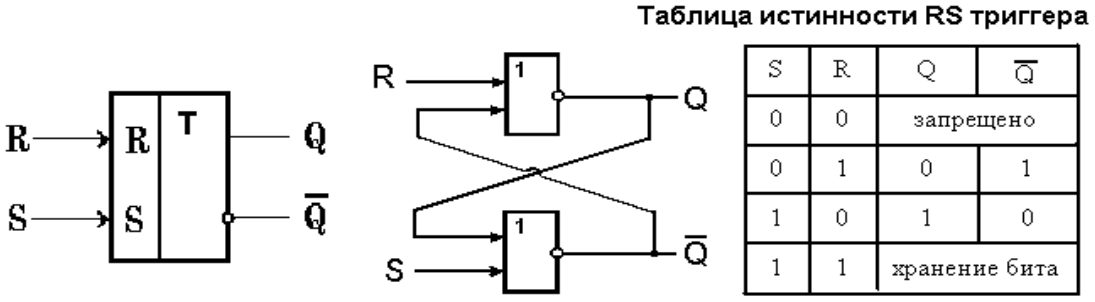


Таблица истинности RS триггера

S	R	Q	$\overline{Q}$
0	0	запрещено	
0	1	0	1
1	0	1	0
1	1	хранение бита	

Рис. 11. Обозначение и таблица истинности RS триггера.

- **Сумматор**

Сумматор служит, прежде всего, центральным узлом арифметико-логического устройства компьютера, однако он находит применение также и в других устройствах машины. Многоразрядный двоичный сумматор, предназначенный для сложения многоразрядных двоичных чисел, представляет собой комбинацию одноразрядных сумматоров, с рассмотрения которых мы и начнём. Условное обозначение одноразрядного сумматора представлено на рис.12.

При сложении чисел A и B в одном i-ом разряде приходится иметь дело с тремя цифрами:

1. цифра a_i первого слагаемого;
2. цифра b_i второго слагаемого;
3. перенос p_{i-1} из младшего разряда.

В результате сложения получаются две цифры:

1. цифра c_i для суммы;
2. перенос p_i из данного разряда в старший.

Таким образом, одноразрядный двоичный сумматор - это устройство с тремя входами и двумя выходами, работа которого может быть описана таблицей истинности приведенной на рис.12.

Если требуется складывать двоичные слова длиной два и более бит, то можно использовать последовательное соединение таких сумматоров, причём для двух соседних сумматоров выход переноса одного сумматора является входом для другого.

Например, схема вычисления суммы C = (c3 c2 c1 c0) двух двоичных трехразрядных чисел A = (a2 a1 a0) и B = (b2 b1 b0) может иметь вид представленный на рис.13.

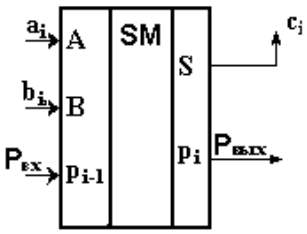


Таблица истинности сумматора

Входы			Выходы	
Первое слагаемое	Второе слагаемое	Перенос	Сумма	Перенос
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Рис. 12. Обозначение и таблица истинности сумматора.

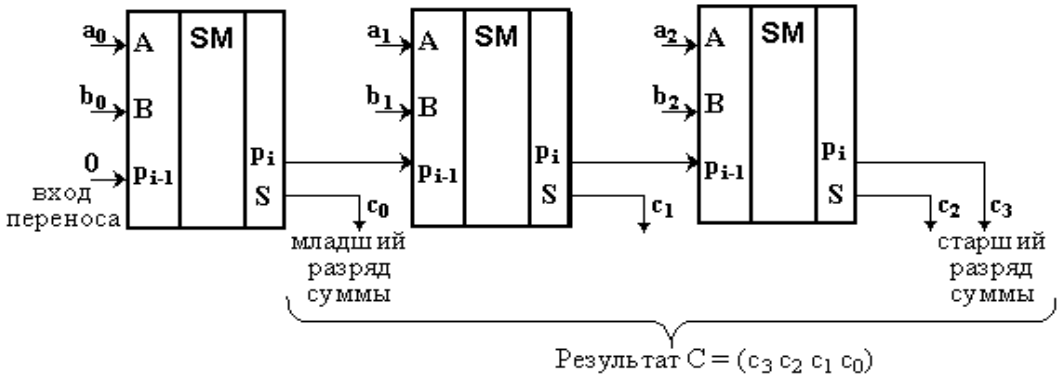


Рис. 13. Принципиальная схема трехразрядного сумматора.

- **Регистр**

Для работы с битами и байтами необходимо объединение отдельных триггеров в линейку из нескольких штук. Такой набор, совместно с общей управляющей логикой, называется регистром. Восьмиразрядные регистры исторически явились основой цифровых микропроцессорных систем.

- **Микропроцессор**

Микропроцессор, иначе, центральный процессор - Central Processing Unit (CPU) - функционально законченное программно-управляемое устройство обработки информации, выполненное в виде одной или нескольких больших (БИС) или сверхбольших (СБИС) интегральных схем.

Для МП на БИС или СБИС характерны:

- простота производства (по единой технологии);
- низкая стоимость (при массовом производстве);
- малые габариты (пластина площадью несколько квадратных сантиметров или кубик со стороной несколько миллиметров);
- высокая надежность;
- малое потребление энергии.

- Микропроцессор выполняет следующие функции:
- чтение и дешифрацию команд из основной памяти;
- чтение данных из оперативного запоминающего устройства (ОЗУ) и регистров адаптеров внешних устройств;
- прием и обработку запросов и команд от адаптеров на обслуживание внешнего устройства (ВУ);
- обработку данных и их запись в ОЗУ и регистры адаптеров ВУ;
- выработку управляющих сигналов для всех прочих узлов и блоков ПК.

Рассмотрим один из самых первых микропроцессоров класса x86 - Intel 8086, являвшийся основой самых первых персональных компьютеров IBM PC. Условное обозначение и внутренняя структура микропроцессора представлены на рис. 14.

На принципиальной схеме показаны номера выводов микропроцессора и их условные обозначения, например вывод Reset – сброс, Ready – готов, INTA,INTR – прерывания, AD – шина адреса или данных, в зависимости от сигнала мультиплексирования ALE. Остальные выводы и сигналы на них относятся к системным и управляющим.

В структуру входит арифметическо - логическое устройство, которое выполняет арифметические и логические действия. Результат выполнения операции помещается в шестнадцатиразрядный аккумулятор AX, состоящий из двух байт — старшего - AH и младшего — AL. Регистр – аккумулятор («накопитель»), предназначен для временного хранения операнда или промежуточного результата действий производимой в АЛУ. Разрядность регистра равна разрядности информационного слова.

Одновременно по результату выставляются флаги состояния, входящие в регистр FLAGS. Кроме аккумулятора в состав микропроцессора входят регистры общего назначения (РОН), это BX, CX и DX. Регистры общего назначения, число которых может изменятся от 4 до 64, определяют вычислительные возможности МП. Их функция – хранение операндов. Но могут выполнять также и роль регистров. Все РОН доступны программисту, который рассматривает их как сверхоперативное запоминающее устройство.

Каждый из РОН имеет некоторые специальные функции, например BX часто используется для хранения адреса таблиц (база), CX — как счетчик, DX — как расширение аккумулятора

Арифметическо - логическое устройство состоит из двоичного сумматора со схемами ускоренного переноса, сдвигающего регистры и регистров для временного хранения операндов. Обычно это устройство выполняет по командам несколько простейших операций: сложение, вычитание, сдвиг, пересылку, логическое сложение (ИЛИ), логическое умножение (И), сложение по модулю 2.

Регистр флагов — FLAGS состоит из флага переноса CF, если таковой случился, флага четности PF, если результат вычисления четный, флага нуля ZF, если получился нуль, флага знака SF и т.д., что всегда позволяет оценить произведенную операцию.

Устройство управления и синхронизации управляет работой АЛУ и внутренних регистров в процессе выполнения команды. Согласно коду операций, содержащемуся в команде, оно формирует внутренние сигналы управления блоками МП. Адресная часть команды совместно с сигналами управления используется для считывания данных из определенной ячейке памяти или для записи данных в ячейку. По сигналам УУ осуществляется выборка каждой новой, очередной команды.

Счетчик команд IP (программный счетчик) содержит адрес ячейки памяти, в которой помещены байты выполняемой команды.

Регистр команд принимает и хранит код очередной команды, адрес которой находится в счетчике команд. По сигналу УУ в него передается из регистра хранимая там информация.

Сегментные регистры DS, CS, ES и SS необходимы для адресации сегментов памяти ОЗУ, так как вся память делится на сегменты.

Регистры смещения SP, BP, SI и DI служат для указания точного адреса внутри конкретного сегмента памяти ОЗУ.

Регистры стека делятся на стек и указатель стека SP. В МП стек – набор регистров,

хранящих адреса команд возврата при обращении к подпрограммам или состояние внутренних регистров при обработке прерываний.

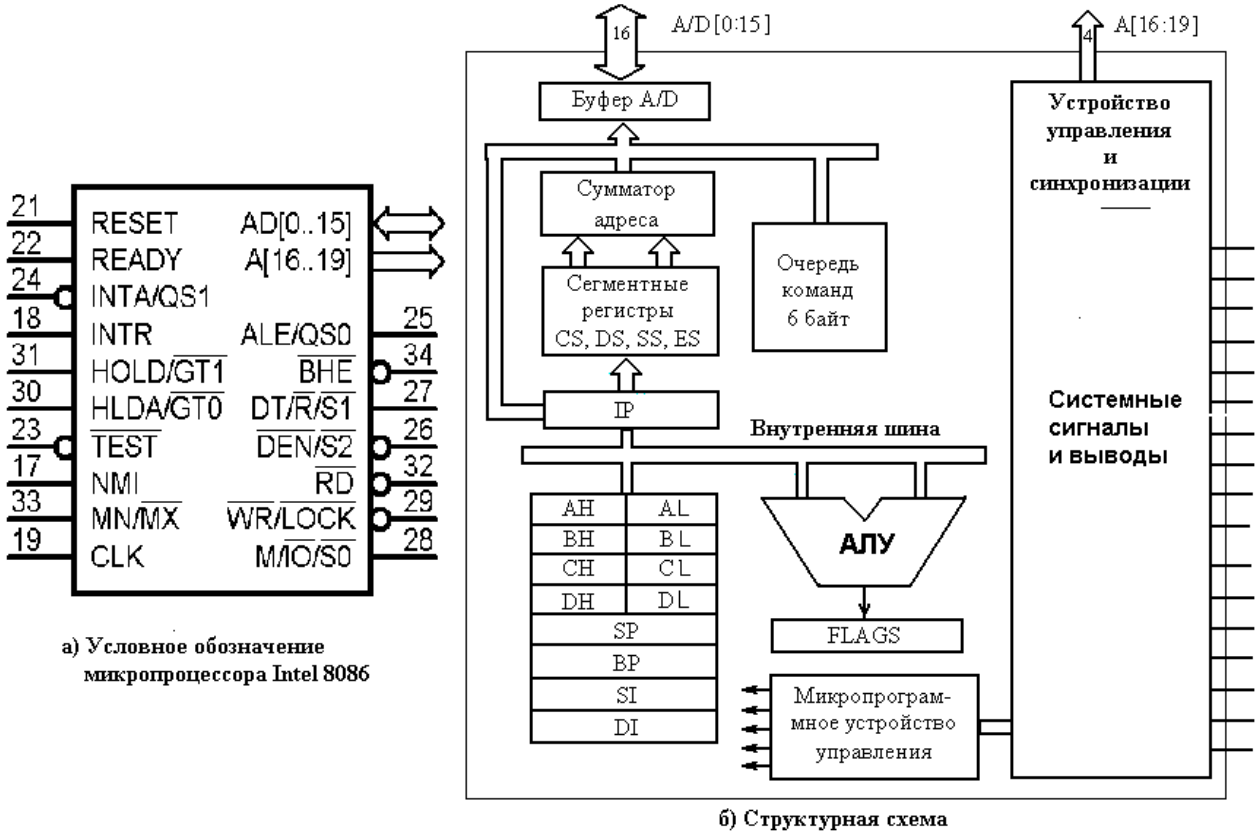


Рис.14. Принципиальная и структурная схема микропроцессора компьютера

Стек находится в ОЗУ, занимая там отведенную для него зону. Для обращения к нему необходим специальный регистр – указатель стека.

Указатель стека хранит адреса последней занятой ячейки стека, которую называют вершиной. Содержащее в указателе число указывает, где находится вершина стека. Когда в стек записывается очередное слово, то число в указателе стека соответственно увеличивается. Извлечение слова из стека сопровождается, наоборот, уменьшением числа, заполняющего указатель стека.

Как мы видим по приведенному описанию микропроцессора, несмотря на простоту идеи фон Неймана, ее техническая реализация достаточно сложна.

Компьютерные сети

В настоящее время широко распространены локальные вычислительные сети (ЛВС), в которых компьютеры поддерживают совместную работу. **ЛВС** обычно развертываются в рамках некоторой организации. Поэтому их иногда называют корпоративными системами или сетями. Компьютеры при этом, как правило, находятся в пределах одного здания или соседних зданий.

Собственными ресурсами компьютера традиционно управляет операционная система, которая, как правило, реализует и функции сетевого управления.

Для унификации сетевых функций и обеспечения возможности взаимодействия любых вычислительных систем Международная Организация по Стандартизации **(International Standard Organization — ISO)** разработала эталонную модель взаимодействия открытых систем **(Open System Interconnection — OSI).**

Эталонная модель **OSI** определяет следующие семь функциональных уровней:

- физический **(physical layer);**
- управления линией передачи или канальный **(data link);**
- сетевой **(network layer);**
- транспортный **(transport layer);**
- сеансовый **(session layer);**
- представления данных **(presentation layer);**
- прикладной, или уровень приложений **(application layer).**

Физический уровень обеспечивает интерфейс между **ЭВМ** сети и средой передачи сигналов. На физическом уровне это **сетевые адаптеры** и канал передачи данных. Управление физическим каналом сводится к выделению начала и конца кадра с передаваемыми данными, а также к формированию и приему сигналов определенной физической природы (ток, напряжение, свет, радиосигнал).

Функции ***канального уровня*** состоят в управлении вводом-выводом информации в канале связи. Для повышения надежности передачи данных канальный уровень может предусматривать введение избыточных кодов, повторную передачу данных и другие методы. Формируемые этим уровнем данные группируются в так называемые ***кадры*** или ***фреймы.*** Обмен данными между двумя объектами канального уровня может вестись одним из трех способов: *дуплексным* (одновременно в обоих направлениях), *полудуплексным* (попеременно в обоих направлениях) или *симплексным* (в одном направлении).

Сетевой уровень обеспечивает передачу сетевых пакетов между узлами сети. Здесь решаются задачи выбора, управления входящим потоком, буферизации пакетов и т. д. Основная аппаратура этого уровня – **коммутаторы** и **маршрутизаторы**.

Основной функцией ***транспортного уровня*** является доставка сообщений (транспортных блоков), которые состоят из сетевых пакетов. Транспортный уровень занимается согласованием различных сетевых уровней с помощью соответствующих ***шлюзов*** (согласование принципиально *различных* сетей) и ***мостов*** (согласование *однотипных* сетей).
Для контроля того, что все отправленные пакеты приняты и в них нет ошибок, применяется метод посылки квитанций — ***квитирование.*** Квитанции подтверждают прием. В настоящее время существует несколько классов сервиса, предоставляемого транспортным протоколом, которые различаются возможностями приоритетной передачи сообщений, защиты от ошибок, а также засекречивания данных с помощью шифрования.

Сеансовый уровень предназначен для организации сеансов связи между объектами сети более высоких уровней При установлении сеансов связи контролируется полномочие объекта по доступу к другому объекту. Данный уровень, как и транспортный, предусматривает несколько классов услуг **(A, B, C** и **D).**

Уровень представления данных описывает методы преобразования информации (шифрование, сжатие, перекодировка), передаваемой объектам прикладного уровня: пользователям и программам.

Прикладной уровень отвечает за поддержку прикладного ПО пользователя. На **этом** уровне реализуются три основные службы: передача и управление файлами, **передача в** обработка заданий, а также виртуальный терминал.
Предложенная семиуровневая модель описывает общие принципы сетевого объединения компьютеров. Для описания взаимодействия программных и аппаратных элементов уровней используются протоколы и интерфейсы.

Протоколом называется свод правил взаимодействия объектов однотипного уровня, а также форматы передаваемых между объектами блоков данных

Интерфейсы описывают процедуры взаимодействия объектов смежных уровней и форматов информации, передаваемой между этими объектами.

Разработка силами **ISO** множества рекомендаций по организации сетевого обмена между компьютерами внесла существенный вклад в создание как глобальных, так и локальных сетей, но принятие международных стандартов не устранило полностью разнообразия архитектур реальных существующих сетей.

Отличия сетей друг от друга вызваны особенностями используемого аппаратного и программного обеспечения, различной интерпретацией рекомендаций фирмами-разработчиками, различием требований к системе, например, требованиями защищенности информации, скорости обмена, безошибочности передачи данных и т. д..

Аппаратные средства ЛВС

Основными аппаратными компонентами **ЛВС** являются:

- рабочие станции;
- серверы;
- сетевые адаптеры;
- сетевые устройства;
- кабели, антенны и т.д.

Рабочие станции (РС) — это, как правило, персональные **ЭВМ,** которые являются рабочими местами пользователей сети.

Иногда в РС, непосредственно подключенной к сети, могут отсутствовать накопители на магнитных дисках или флеш - памяти. Такие РС называют ***бездисковыми рабочими станциями*** («тонкий клиент»)**.** В этом случае загрузка в РС операционной системы производится с файл-сервера. Основным преимуществом бездисковых РС является низкая стоимость, а также высокая защищенность от несанкционированного проникновения в систему пользователей и компьютерных вирусов. *Недостаток* бездисковой РС заключается в невозможности работать в автономном режиме (без подключения к серверу), а также иметь свои собственные архивы данных и программ.

Серверы в **ЛВС** выполняют функции распределения сетевых ресурсов. Обычно его функции возлагают на достаточно мощную ЭВМ. В одной сети может быть один или несколько серверов. Каждый из серверов может быть отдельным или совмещенным с РС. В последнем случае не все, а только часть ресурсов сервера оказывается общедоступной.

При наличии в **ЛВС** нескольких серверов каждый из них управляет работой подключенных к нему РС. Совокупность компьютеров сервера и относящихся к нему РС часто называют **доменом.** Иногда в одном домене находится несколько серверов. Обычно один из них является главным, а другие — выполняют роль резерва (на случай отказа главного сервера) или логического расширения основного сервера.

РС и серверы соединяются друг с другом посредством ***линий передачи данных,*** например витой парой. Подключение компьютеров к линии осуществляется с помощью ***сетевых адаптеров.*** В последнее время стали применяться беспроводные Wi-Fi сети, средой передачи данных в которых является радиоканал. В подобных сетях компьютеры устанавливаются на небольших расстояниях друг от друга: в пределах одного или нескольких соседних помещений.

К **сетевым устройствам** относят модемы, трансиверы, репитеры, концентраторы, коммутаторы, маршрутизаторы, мосты, шлюзы, а также различные разъемы (коннекторы, терминаторы).

Модем используется в качестве устройства подключения **ЛВС** или отдельного компьютера к глобальной сети через телефонную связь. ***Трансивер*** — это приемопередатчик, преобразующий форму и характер сигналов. ***Репитер*** предназначен для усиления сигнала и соединения сегментов сетей. ***Концентраторы*** и ***Коммутаторы*** необходимы для организации сегментов сети компьютеров либо с широкополосным доступом, либо с адресным. ***Маршрутизаторы*** являются более сложным устройством, предназначены для определения оптимального маршрута прохождения информации в разветвленных сетях. ***Мосты*** необходимы для объединения ЛВС с разными протоколами. ***Шлюзы*** служат для объединения ЛВС с глобальными. ***Терминаторы*** служат для подключения к открытым кабелям сети, а.также для заземления (так называемые терминаторы с заземлением).

Структурно - функциональная организация ЛВС

Топология ЛВС

Конфигурация или топология соединения элементов в сеть определяет такие важнейшие характеристики сети, как ее надежность, производительность, стоимость, защищенность и т. д.

Одним из подходов к классификации топологий **ЛВС** является выделение двух основных классов топологий: ***широковещательных*** и ***последовательных.***

В ***широковещательных*** конфигурациях каждый персональный компьютер передает сигналы, которые могут быть восприняты остальными компьютерами. К таким конфигурациям относятся топологии «общая шина», «дерево», «звезда с пассивным центром».

В ***последовательных*** конфигурациях каждый физический подуровень передает информацию только одному персональному компьютеру. Примерами таких конфигураций являются: произвольная (произвольное соединение компьютеров), иерархическая, «кольцо», «цепочка», «звезда с интеллектуальным центром», «снежинка» и другие.

Наиболее широко распространены базовые топологии **ЛВС:** «звезда», «общая шина» и «кольцо», рис. .

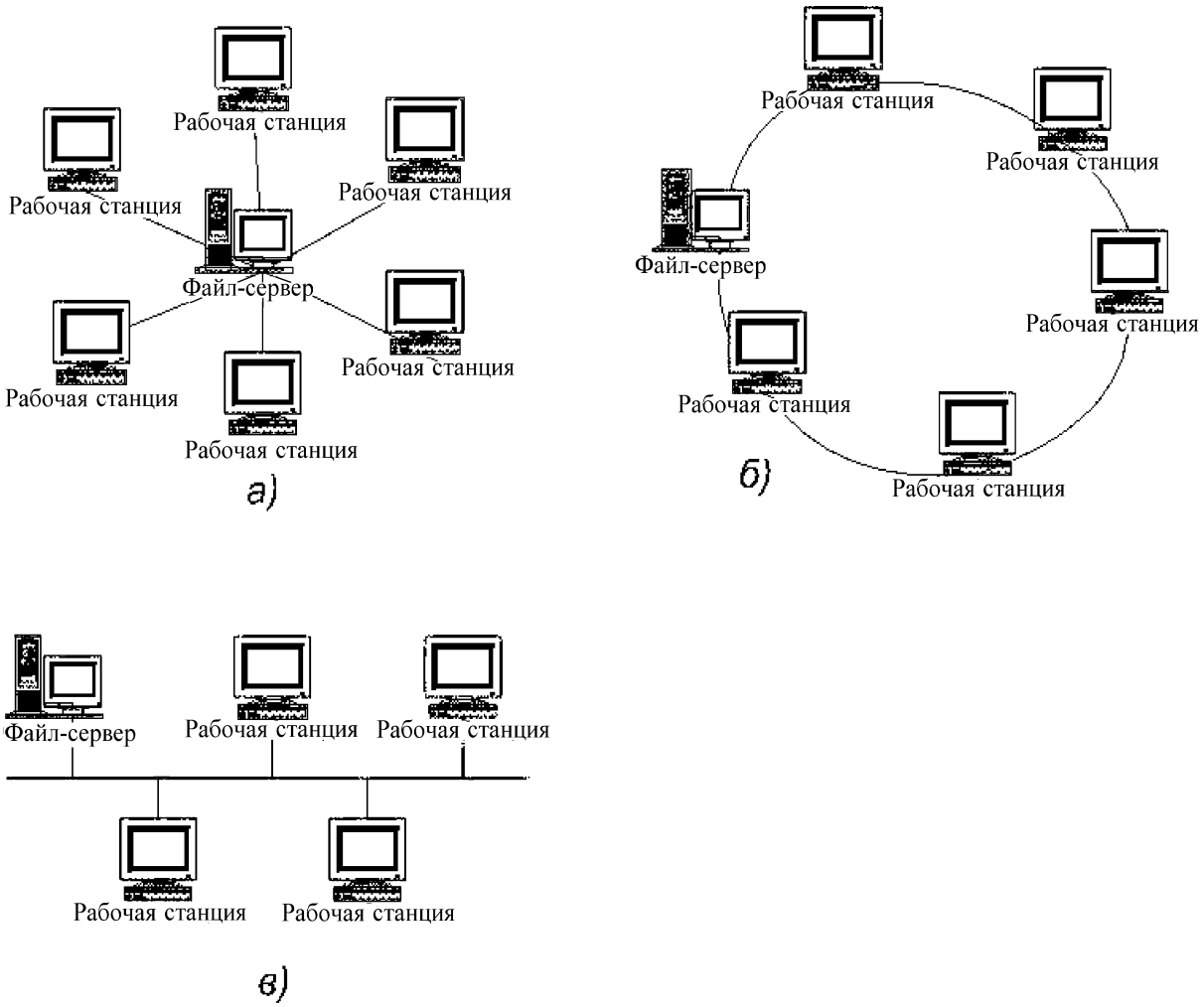


Рис. Базовые сетевые топологии

В случае ***топологии - «звезда»*** каждый компьютер через сетевой адаптер подключается отдельным кабелем к центральному узлу. Центральным узлом служит либо файл – сервер, либо соединитель или активный повторитель.

Недостатком такой топологии является низкая надежность, так как выход из строя

центрального узла приводит к остановке всей сети, а также обычно большая протяженность кабелей (это зависит от реального размещения компьютеров

Топология -«общая шина» предполагает использование одного кабеля, к которому подключаются все компьютеры. Информация по нему передается компьютерами поочередно.

Достоинством такой топологии является, как правило, меньшая протяженность кабеля, а также более высокая надежность чем, у «звезды», так как выход из строя отдельной станции не нарушает работоспособности сети в целом. Недостатки состоят в том, что обрыв основного кабеля приводит к неработоспособности всей сети, а также слабая защищенность информации в системе на физическом уровне.

При ***кольцевой топологии*** данные передаются от одного компьютера другому по эстафете. Если некоторый компьютер получает данные, предназначен ему, он передает их дальше по кольцу.

Достоинством кольцевой топологии является более высокая надежность систем при разрывах кабелей, чем в случае топологии с общей шиной, так как к каждой РС есть два пути доступа. К недостаткам топологии следует отнести большую протяженность кабеля, невысокое быстродействие по сравнению со «звездой», а также слабая защищенность информации.

Топология реальной ЛВС может в точности повторять одну из приведенных или включать их комбинацию. Структура сети в общем случае определяется следущими факторами: количеством объединяемых компьютеров, требованиями по надежности и оперативности передачи информации, экономическими соображениями и т. д.

Существует два основных принципа управления в локальных сетях: централизованный файл-серверный и децентрализованный одноранговый.

В сетях с ***централизованным управлением*** функции управления обменом данными возложены на файл-серверы. Преимуществом централизованных сетей является высокая защищенность сетевых ресурсов от несанкционированного доступа, удобство администрирования сети, возможность создания сетей с большим числом узлов. Основной недостаток состоит в уязвимости системы от сбоев файл-сервера

Децентрализованные (одноранговые) сети не содержат в своем составе выделенных серверов. Функции управления сетью в них поочередно передаются от одной РС к другой.

Для организации обмена между компьютерами **ЛВС** используются стандартные протоколы, разработанные Международным институтом инженеров по электротехнике и радиоэлектронике IEEE (Institute of Electrical and Electronical Engineers).
Это стандарты IEEE802.3, IEEE802.4 и IEEE802.5, которые соответствуют методам доступа к сетевым каналам данных: Ethernet, Arcnet и Token Ring.

Метод доступа **Ethernet** был разработан фирмой Xerox. Обеспечивает высокую скорость передачи и надежность. Является методом множественного доступа с прослушиванием несущей и разрешением конфликтов (CSMA/CD — Carier Sense Multiple Access with Collision Detection). Суть метода состоит в том, что РС начинает передачу в том случае, если канал свободен, в противном случае передача сообщений задерживается на некоторое время (для каждой станции свое). Возможные случаи одновременной передачи данных распознаются автоматически аппаратным способом.

Метод доступа **Arcnet.** Разработан фирмой Datapoint Corp. Используется в топологии «звезда». Сообщения от одной РС к другой по этому методу доступа передаются с помощью ***маркера,*** который создается на одной из РС. Если РС хочет передать сообщение, то она дожидается прихода маркера и присоединяет к нему свое сообщение, снабженное адресами отправителя и получателя.

Метод доступа **Token Ring.** Разработан фирмой **IBM** для кольцевой топологии. Имеет сходство с методом Arcnet. Основное его отличие состоит в том, что имеется механизм приоритета, благодаря которому отдельные РС могут получать маркер быстрее других и удерживать его дольше.

Архитектура Internet

Internet представляет собой Всемирную сеть, информация в которой хранится на серверах. Серверы имеют свои адреса и управляются специализированными программами. Они позволяет пересылать почту и файлы, производить поиск в базах данных и т. п. Обмен информацией между серверами сети выполняется по высокоскоростным каналам связи. Доступ отдельных пользователей к информационным ресурсам Internet обычно осуществляется через *провайдера* или *корпоративную сеть*. В качестве провайдера выступает некоторая организация, имеющая каналы для соединения с клиентами и выхода во Всемирную сеть.

Рассмотрим структурную схему построения Internet (WAN). На рис. показана архитектура сети. В качестве магистрали передачи данных используются телефонные линии, радиоканалы, оптоволоконные и спутниковые каналы связи.

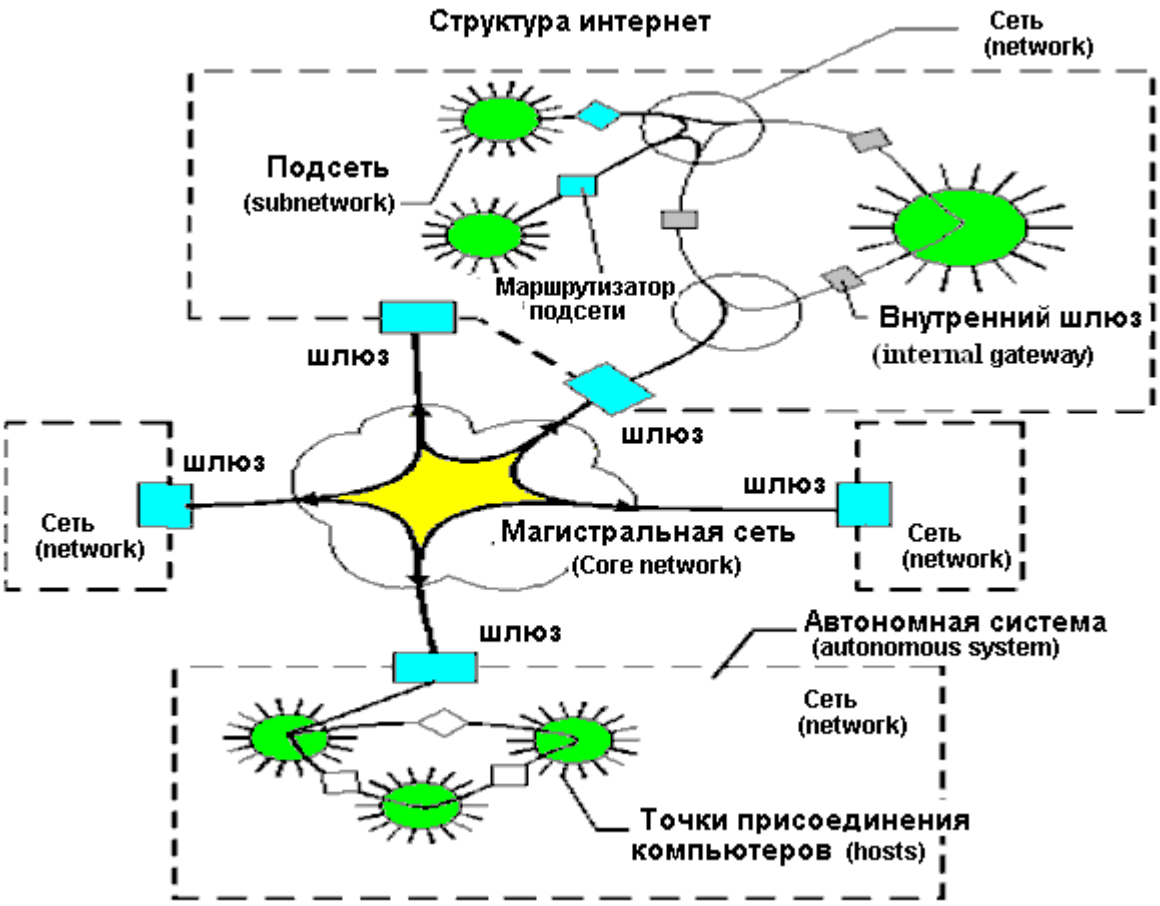


Рис. Архитектура интернет

Любая организация для подключения к Internet использует ***шлюз*** (gateway). На нем устанавливается программное обеспечение, осуществляющее обработку всех сообщений, проходящих через шлюз. Каждый шлюз имеет свой IP-адрес. Шлюзы могут быть внешними и внутренними. Внутренние шлюзы поддерживают связь между собой с помощью внутреннего шлюзового протокола **IGP (Internal Gateway Protocol).** ***Внешние шлюзы*** применяются в больших сетях, подобных **Internet,** настройки их постоянно меняются из-за изменений в мелких подсетях. Связь между внешними шлюзами осуществляется через внешний шлюзовый протокол **EGP (Exterior Gateway Protocol).**

Подключаемая к интернету сеть может быть как городской (MAN), так и локальной

(LAN). Отдельные подсети снабжаются маршрутизаторами подсетей.

Если поступает сообщение, адресованное локальной сети, к которой подключен шлюз, то оно передается в эту локальную сеть. Если сообщение предназначено для другой сети, то оно передается следующему шлюзу. Каждый шлюз имеет информацию обо всех остальных шлюзах и сетях. Когда сообщение посылается из локальной сети через шлюз в Internet, то при этом выбирается самый «быстрый» путь. Шлюзы обмениваются друг с другом информацией о маршрутизации и состоянии сети, используя специальный шлюзовый протокол.

Некоторые компании могут выступать в качестве провайдера. ***Провайдер*** имеет свой шлюз в **Internet** и позволяет другим компаниям и отдельным пользователям подключаться к Сети через этот шлюз. Кроме информации о маршрутизации сообщений, шлюзу необходимы данные о параметрах подсетей, подключенных к более крупной сети, для корректировки маршрутов передачи сообщений в случае сбоев в отдельных частях сети.

Протоколы обмена и адресация

Подключение пользователя к **Internet** может осуществляться разными способами, отличающимися по стоимости, удобству и объему предоставляемых услуг. Этими способами являются:

- электронная почта **(E-mail);**
- телеконференции **(TelePresence);**
- программы удаленного доступа и администрирования (**Remote Control**);
- поиск и передача двоичных файлов **(FTP);**
- поиск и передача документов с помощью гипертекстовых ссылок **(WWW,** или **Всемирная паутина).**

Создание и развитие этих способов связано сложилось исторически. Каждый из них характеризуется своими возможностями и различием в организации протоколов обмена информацией. В общем случае под протоколом понимается набор инструкций, регламентирующих работу взаимосвязанных систем или объектов в сети.

Электронная почта (E-mail) — наиболее простой и доступный способ доступа в сети **Internet.** Она позволяет выполнять пересылку любых типов файлов (включая тексты, изображения, звуковые вставки) по адресам электронной почты в любую точку планеты за короткий промежуток времени в любое время суток. Для передачи сообщения необходимо знать только электронный адрес получателя. Работа электронной почты основана на последовательной передаче информации по сети от одного почтового сервера к другому, пока сообщение не достигнет адресата. К достоинствам электронной почты относятся высокая оперативность и низкая стоимость. Недостаток электронной почты состоит в ограниченности объема пересылаемых файлов.

TelePresence разработана как система обмена аудио и видео информацией. Она позволяет всем пользователям **Internet** участвовать в групповых дискуссиях, называемых телеконференциями, в которых обсуждаются всевозможные проблемы. Информация, посылаемая в телеконференции, становится доступной любому клиенту Сети, обратившемуся в данную телеконференцию. В настоящее время телеконференции позволяют передавать файлы любых типов, включая текстовые, графические и аудио файлы. Для работы с телеконференциями наиболее часто используются такие средства как Skype, Google и Microsoft.

Remote Control — это программы, позволяющие управлять удаленным компьютером в локальной сети или через интернет, а также использовать ресурсы удаленного компьютера. Другими словами — это протокол удаленного терминального доступа в сети. Программы позволяют отображать рабочий стол, включать, выключать, блокировать и обмениваться данными от локального компьютера к удаленному компьютеру в Сети. Ранее подобие реализации этих функций осуществлялось через TelNet.

FTP — это протокол Сети для работы с любыми типами файлов: текстовыми и

бинарными, являющийся примером системы с архитектурой «клиент-сервер». FTP-сервер устанавливается на удаленном компьютере для того, чтобы предоставлять пользователям возможность просматривать файловую систему и копировать требуемые файлы. Для реализации связи по протоколу FTP на удаленной компьютерной системе должна функционировать программа — FTP-сервер. Достоинством данного протокола является возможность передачи файлов любого типа — текстов, изображений, исполняемых программ. К недостатку протокола FTP следует отнести необходимость знания местоположения отыскиваемой информации.

WWW (**World Wide Web** — Всемирная паутина) представляет собой самое современное средство организации сетевых ресурсов. Она строится на основе гипертекстового представления информации. Гипертекст — это текст, содержащий ссылки на другие части данного документа, на другие документы, на объекты нетекстовой природы (звук, изображение, видео), а также система, позволяющая читать такой текст, отслеживать ссылки, отображать картинки и проигрывать звуковые и видеовставки. Гипертекст с нетекстовыми компонентами (звук, видео) называется *гипермедиа.* Конечной целью WWW является объединение всех ресурсов сети (файлов, текстов, баз данных, программ-серверов) в единый всемирный гипертекст.

Работа сети Internet основана на использовании семейства коммуникационных протоколов — Протокол управления передачей данных/Протокол Internet (Transmission Control Protocol/Internet Protocol — **TCP/IP**), который используется для передачи данных в глобальной сети и во многих локальных сетях. TCP/IP - семейство протоколов. В состав его входят протоколы, которые можно разделить по назначению на следующие группы:

- транспортные протоколы, служащие для управления передачей данных между двумя компьютерами;
- протоколы маршрутизации, обрабатывающие адресацию данных и определяющие кратчайшие доступные пути к адресату;
- протоколы поддержки сетевого адреса, предназначенные для идентификации компьютера по его уникальному номеру или имени;
- прикладные протоколы, обеспечивающие получение доступа к всевозможным сетевым услугам;
- шлюзовые протоколы, помогающие передавать по сети сообщения о маршрутизации и информацию о состоянии сети, а также обрабатывать данные для локальных сетей;
- другие протоколы, не относящиеся к указанным категориям, но обеспечивающие клиенту удобство работы в сети.

Архитектура **TCP/IP** построена на основе эталонной модели, однако в ней первые три уровня OSI-модели объединены в один .

Любой документ или сообщение отправляется в сеть из прикладной программы (уровень приложений). Затем через линию связи (транспортный уровень) сообщение попадает на узел сети Internet и далее с помощью сетевых программ (сетевой интерфейс) передается в линию связи узлов глобальной сети (физический уровень). Программы каждого уровня по-своему обрабатывают сообщение или передаваемый документ, не зная ничего о его содержании.

В **Internet** каждому компьютеру назначается свой уникальный *сетевой адрес* — IP-адрес, имеющий длину 32 бита и состоящий из 4 частей по 8 битов. Каждая часть может принимать значения от 0 до 255 и отделяется от других частей точкой. Например, **94.105.195.17 и 147.115.3.27** представляют два IP-адреса.

Сетевой адрес состоит из двух частей: адреса сети и адреса хоста в этой сети. Под х***остом*** понимается компьютер, включенный в сеть и предоставляющий различные сетевые услуги. Благодаря такой структуре IP-адреса компьютеры в разных сетях могут иметь одинаковые адреса.

Для обеспечения максимальной гибкости IP-адреса подразделяются на классы A, B и C, в зависимости от количества локальных сетей и компьютеров в них. Классы IP-адресов определяют размер локальной сети организации. В зависимости от класса полный 32-битный

адрес по-разному разбивается на 8-битные составляющие. При этом первые от одного до трех битов в начале IP-адреса идентифицируют соответствующий класс. Структура IP-адресов представлена в таблице .

Таблица

Класс А	0	Адрес сети (7 бит)		Адрес хоста (24 бита)
Класс В	1	0	Адрес сети (14 бит)	Адрес хоста (16 битов)
Класс С	1	1	0 Адрес сети (21 бит)	Адрес хоста (8 битов)

По первому числу IP-адреса можно определить тип класса, к которому относится организация:

Адреса класса А — числа от 0 до 127.
Адреса класса В — числа от 128 до 191.
Адреса класса С — числа от 192 до 223.

Адрес сети ***класса А*** позволяет идентифицировать более 16 миллионов компьютеров в локальной сети организации, но при этом может существовать не более 128 локальных сетей данного класса. Адрес сети ***класса В*** позволяет выделить большее количество локальных сетей, но с меньшим числом компьютеров в самой сети. И, наконец, сети ***класса С*** могут иметь максимум 254 компьютера, но таких сетей может быть свыше 2 миллионов.

При посылке сообщения в **Internet** IP-адрес используется для указания отправителя и получателя. Клиенту нет необходимости запоминать сетевые адреса, поскольку в сети используют ***доменные имена,*** которые преобразуются доменной системой имен в IP-адреса.

Адреса в **Internet** строятся по доменной системе адресации **(domain name system, DNS).** Это означает, что адрес пользователя состоит из двух частей: идентификатора пользователя и названия домена, разделенных символом @:
Идентификатор пользователя>@<название домена>

Идентификатор пользователя и название домена могут состоять из сегментов, разделяемых точкой. В адресе допускается использование латинских букв, цифр и некоторых других символов. Например:

Ivan.Ivanov@mycomputer-vyatsu.kirov.ru

В примере идентификатор пользователя состоит из двух сегментов, а название домена — из четырех. Обычно сегменты домена или ***поддомены*** образуют иерархическую структуру: первый слева поддомен, как правило, является названием компьютера, которому присвоен этот адрес, следующий относится к названию организации, где находится этот компьютер, а крайний правый (поддомен верхнего уровня) является сокращенным обозначением страны. Приведенный адрес означает, что он принадлежит Иванову Ивану, сотруднику Вятского Государственного университета г. Кирова в России, имеющему компьютер с именем **mycomputer.** Идентификаторы пользователей могут быть любыми: полное имя и фамилия, инициалы, фамилия с инициалами, прозвища, а также названия организаций или отделов. При этом на одном компьютере может быть произвольное (ограниченное допустимым количеством IP-адресов) число зарегистрированных пользователей со своими адресами или пользователь может иметь несколько адресов на домене (один, например, для личной переписки, а другой — для официальной).

Поддомен верхнего уровня, обозначающий страну, состоит обычно из двух букв: ru - Россия, su - территория республик бывшего Союза, ca - Канада, uk - Великобритания, ua — Украина, de - Германия и т. д.

В США традиционно используется другая система. Поддомен верхнего уровня состоит из трех букв и обозначает принадлежность владельца адреса к одному из следующих классов:

com — коммерческие организации;
edu — учебные и научные организации;
gov — правительственные учреждения;
mil — военные организации;
net — сетевая администрация;
org — прочие организации.

Программы просмотра Web-документов

Для работы в WWW на компьютере необходимо иметь специальную программу — браузер (browser). **Браузер** — это прикладная программа, взаимодействующая с WWW и позволяющая получать из сети различные документы, просматривать и редактировать х содержимое. Браузеры предоставляют возможность работы с документами, содержащими текстовую и мультимедийную информацию.

В WWW документы, как правило, содержат гипертекст (текст с гиперссылками). В отличие от обычных текстов, документы в сети содержат команды, задающие их структуру, включая ссылки на другие документы. Это позволяет браузеру отформатировать документ для его отображения на экране в соответствии с возможностями конкретного компьютера. Поскольку в составе Internet используются разнородные аппаратно-программные средства, то для разработки Web-страниц был принят универсальный язык разметки гипертекста — HTML (HyperText Markup Language).

В состав HTML входит набор команд, используемых для описания структуры документа. С помощью HTML документ разбивается на соответствующие логические компоненты: абзацы, заголовки, списки и т д. Конкретные атрибуты форматирования документа (основного текста и выделенных компонентов) при его просмотре определяются используемым браузером.

Наиболее распространенными браузерами являются: Internet Explorer, Opera, Firefox, Google Chrome, Mozilla и др.

Операционные системы и сервисные программы

Для обеспечения работы компьютера и всех его компонент требуется системное программное обеспечение. Это, в первую очередь, операционная система (ОС) и базовая система ввода-вывода BIOS. Базовая система BIOS находится в ПЗУ компьютера и устанавливает связь между материнской платой (материнские платы все разные, но «точки входа» к аппаратуре все одинаковы) и операционной системой на самом низком уровне, одновременно выполняя при старте процедуру POST - проверки работоспособности аппаратуры.

Операционная система представляет собой комплекс управляющих и обрабатывающих программ, выступающих как интерфейс между аппаратурой компьютера и пользователем, и обеспечивающих эффективное использование ресурсов вычислительной системы для выполнения программ пользователя.

Основными функциями ОС являются следующие:

- Прием команд и их обработка;
- Загрузка в ОЗУ программ и запуск их на исполнение;
- Обслуживание операций ввода — вывода;
- Выполнение системных сервисов — как API — интерфейса прикладного программирования (от английского Application Program Interface);
- Обеспечение функций работы с файлами;
- Обеспечение мультизадачной работы системы — то есть возможности одновременного запуска нескольких программ на одном компьютере;
- Обеспечение контроля и сохранности данных, защита ядра ОС от сбоев;
- Авторизация пользователей и их аутентификация по правам доступа к системе;
- Обеспечение сетевого взаимодействия компьютеров между собой.

Наиболее популярными в настоящее время являются операционные системы семейства Windows компании Microsoft. Это Windows XP, Windows 7, Windows 8 и UNIX - подобные операционные системы Linux и системы FreeBSD.

ОС Windows XP

Windows XP представляет из себя модульную операционную систему, состоящую из отдельных взаимосвязанных относительно простых модулей. Основными модулями являются: уровень аппаратных абстракций HAL (Hardware Abstraction Layer), ядро (Kernel), исполняющая система (Executive), защищенные подсистемы (protected subsystems) и подсистемы среды (environment subsystems), рис.15.

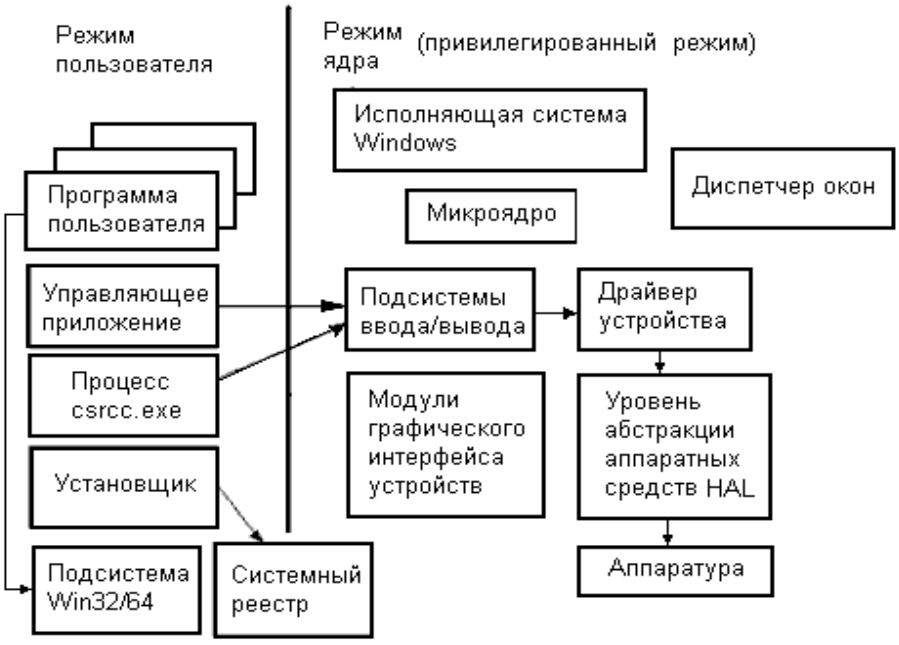


Рис.15. Архитектура операционной системы класса Windows XP

Ядро (микроядро) ОС является ее «сердцем» и первую очередь занимается планированием действий компьютерного процессора. Если компьютер содержит несколько процессоров, то ядро синхронизирует их работу, чтобы достичь максимальной производительности системы.

Ядро осуществляет диспетчеризацию потоков (нитей) управления, которые являются основными объектами в планируемой системе. Нити управления определяются в контексте процесса, который включает адресное пространство и набор доступных процессу объектов. Объектами являются управляемые операционной системой ресурсы. Ядро производит диспетчеризацию нитей управления таким образом, чтобы максимально загрузить процессоры системы и обеспечить первоочередную обработку нитей с более высоким приоритетом (всего 32 уровня приоритета).

Исполняющая система, в состав которой входит ядро и уровень аппаратных абстракций HAL, обеспечивает общий сервис системы, который могут использовать все подсистемы среды. Каждая группа сервиса находится под управлением одной из отдельных составляющих исполняющей системы (Win 32/64 Executive):

- диспетчера процессов (Process Manager);
- диспетчера виртуальной памяти (Virtual Memory Manager);
- диспетчера объектов (Object Manager);
- средства вызова локальных процедур (Local Procedure Call Facility);

- диспетчера ввода/вывода (I/O Manager);
- мониторы безопасности (Security Reference Monitor).

Диспетчер процессов создает, отслеживает и удаляет процесс согласно его приоритету.
Диспетчер виртуальной памяти выделяет виртуальную память процессу в ОЗУ и следит за тем, чтобы каждому процессу выделялось отдельное адресное пространство.
Диспетчер объектов создает и поддерживает объекты, следит за их атрибутами защиты. Объектами считаются каталоги, файлы, процессы, события и т.д.
Диспетчер ввода/вывода управляет всеми операциями ввода/вывода, организуя взаимодействие систем через драйверы устройств.
Монитор безопасности обеспечивает слежение за санкционированием доступа к объектам и реализует модель безопасности Windows.

ОС UNIX

Свободнораспространяемая операционная система UNIX (LINUX) является мультипрограммной и многопользовательской. Центральной частью UNIX — систем является макроядерное ядро (Kernel). Оно состоит из большого количества модулей и с точки зрения архитектуры является монолитным. В нем можно выделить три подсистемы: управления процессами, управления файлами и операциями ввода/вывода. Файловая система UNIX имеет древовидную структуру. Узлами дерева являются каталоги, а листья дерева соответствуют файлам. Каждому зарегистрированному пользователю соответствует свой так называемый «домашний» каталог (home). К каталогам других пользователей доступ ограничен (все зависит от привилегий). Имеется суперпользователь — администратор системы UID с неограниченным доступом.

Традиционным способом взаимодействия пользователя с UNIX является командный язык shell (для Linux – bash). В последнее время широко используется графический интерфейс (GUI), подобный интерфейсу Windows. В системах Linux — это KDE и GNOME.

Операционная система FreeBSD также является UNIX — подобной, но с микроядерным ядром. Основным разработчиком этой операционной системы является университет Беркли, Калифорния.

Сервисные программы

Хотя разработчики операционных систем стараются выпускать самодостаточные продукты, но нормальная работа систем невозможна без сервисного обслуживания и сервисных программ. К узкопрофессиональным программам относится огромное множество программ специального назначения, ориентированных на специалистов в определенной области. Например, для подготовки каких-то текстовых документов, например, заявлений, отчетов, деловых писем и т.д. используют специальные программы - **текстовые редакторы.** Примеры: Word, Writer. Разновидностью текстовых редакторов являются издательские системы, используемые при издании книг, журналов, газет, рекламных объявлений.

Очень часто человек сталкивается с необходимостью выполнить какие-то расчеты или другие операции над данными в табличной форме. Вообще, таблицы сопровождают нас всю жизнь - расписание занятий, экзаменационная ведомость, расписание поездов, турнирная таблица футбольного чемпионата и т.д. Для автоматизированной обработки данных в табличной форме используют специальные программы - **электронные таблицы**. Примеры: Excel, Quattro и др.

Производители программного обеспечения стараются объединить программы разного функционального назначения, но входящие в какую-то одну сферу деятельности человека, в один пакет программ. Типовым решением такого подхода является пакет Microsoft Office. Рассмотрим состав его структуры:

-

На ЭВМ широко используются **файловые менеджеры**, такие как Norton Commander, FAR Manager, Windows Commander предоставляющие пользователю значительно большие

удобства, чем, например стандартный проводник Windows.

Для работы с интернет — документами используются программы — **браузеры,** например Internet Explorer. Opera, Google Chrome, Mozilla FireFox и др.

Для обслуживания компьютера широко используются различные **утилиты** (лат. utilitas — польза), которые можно разделить на следующие группы:

1. Программы контроля, тестирования и диагностики, которые используются для проверки правильности функционирования устройств компьютера и для обнаружения неисправностей в процессе эксплуатации; указывают причину и место неисправности;
2. Программы-драйверы, которые расширяют возможности операционной системы по управлению устройствами ввода-вывода, оперативной памятью и т.д.; с помощью драйверов возможно подключение к компьютеру новых устройств или нестандартное использование имеющихся;
3. Программы-упаковщики (архиваторы), которые позволяют записывать информацию на дисках более плотно, а также объединять копии нескольких файлов в один архивный файл. Наиболее используемыми архиваторами являются arj, zip и rar, Они позволяют разархивировать файлы, т.е. производить обратную операцию по извлечению отдельных файлов из архива.
4. Антивирусные программы, предназначенные для предотвращения заражения компьютерными вирусами и ликвидации последствий заражения вирусами
5. Программы оптимизации и контроля качества дискового пространства;
6. Программы восстановления информации, форматирования, защиты данных;
7. Коммуникационные программы, организующие обмен информацией между компьютерами;
8. Программы для управления памятью, обеспечивающие более гибкое использование оперативной памяти;
9. Программы для записи CD-ROM, CD-R и многие другие.

Проблемы защиты информации

Развитие новых информационных технологий и всеобщая компьютеризация привели к тому, что одной из самых актуальных проблем современности стала информационная безопасность. Под угрозой безопасности информации понимаются события или действия, которые могут привести к искажению, несанкционированному использованию или разрушению информационных ресурсов, программных и аппаратных средств.

В настоящее время для обеспечения защиты информации требуется системный подход, так как злоумышленники стремятся любыми средствами добыть важную для них информацию.

Рассмотрим виды угроз безопасности информации:

- утечка конфиденциальной информации;
- несанкционированное использование информационных ресурсов;
- ошибочное использование информационных ресурсов;
- несанкционированный обмен информацией между абонентами;
- отказ от информации;
- нарушение информационного обслуживания;
- незаконное использование привилегий.

Утечка конфиденциальной информации — это бесконтрольный выход конфиденциальной информации за пределы компьютерной системы, например, по
визуально-оптическим, акустическим, электромагнитным и другим каналам.

Несанкционированный доступ — это противоправное преднамеренное овладение конфиденциальной информацией лицом, не имеющим права доступа к охраняемым сведениям. Наиболее распространенными путями несанкционированного доступа к информации являются:

• перехват электронных излучений;
• принудительное электромагнитное облучение (подсветка) линий связи с целью получения информации от паразитной модуляции сигнала;
• применение подслушивающих устройств (закладок);
• дистанционное фотографирование;
• перехват акустических излучений и восстановление текста принтера;
• чтение остаточной информации в памяти системы после выполнения санкционированных запросов;
• копирование носителей информации с преодолением мер защиты;
• маскировка под зарегистрированного пользователя;
• маскировка под запросы системы;
• использование программных ловушек;
• использование недостатков языков программирования и операционных систем;
• незаконное подключение к аппаратуре и линиям связи;
• злоумышленный вывод из строя механизмов защиты;
• расшифровка специальными программами зашифрованной информации;
• информационные инфекции.

Большинство из перечисленных технических путей несанкционированного доступа поддаются надежной блокировке при правильно разработанной и реализуемой на практике системе обеспечения безопасности. Но борьба с информационными инфекциями представляет значительные трудности, так как существует и постоянно разрабатывается огромное множество вредоносных программ, цель которых — порча информации в компьютерах.

Вредоносные программы классифицируются следующим образом:

- Логические бомбы, которые используются для искажения или уничтожения информации, например: программист, предвидя свое увольнение, вносит в программу расчета заработной платы определенные изменения, которые начинают действовать, когда его фамилия исчезнет из набора данных о персонале фирмы. Часто встречаются в архивах – ArcBomb.
- Троянский конь — программа, выполняющая в дополнение к основным санкционированным действиям некоторые вредоносные. Троянский конь представляет собой дополнительный блок команд, тем или иным образом вставленный в легальную программу и срабатывающий при наступлении какого-то условия (даты, времени, по команде извне и т.д.). Злоумышленник, составивший и внедривший троянского коня, может, например, удаленно управлять зараженным компьютером, красть конфиденциальную информацию - пароли, номера банковских счетов и т.д. Часто используется технология Rootkit – скрытие присутствия в системе.
- Вирус — программа, которая может заражать другие программы путем включения в них модифицированной копии, обладающей способностью к дальнейшему размножению. Имеются следующие виды вирусов:
    1) Файловые вирусы – внедряются в исполняемые файлы и совершают вредоносные действия;
    2) Макровирусы – используют макросы офисных программ и заражают документы;
    3) Загрузочные вирусы – проникают в загрузочный (boot) сектор;
    4) Скриптовые вирусы – заражают командные или служебные программы операционной системы, могут заражать HTML – страницы, если в них разрешено исполнять скрипты (Visual Basic Script, Java Script и т.д.);
    5) Зомби – программа, внедренная и управляемая по сети извне, незаметно для пользователя компьютера, предназначенная для атак на другие компьютеры;

6) Malware – вредоносная программа, специально разработанная для нанесения вреда компьютеру для вывода его из строя.

- Червь — программа, распространяющаяся через сеть и не оставляющая своей копии на магнитном носителе. Червь использует механизмы поддержки сети для определения узла, который может быть заражен. Затем с помощью тех же механизмов передает свое тело или его часть на этот узел и либо активизируется, либо ждет для этого подходящих условий. Часто черви предназначены для замусоривания системы и затормаживания ее работы. Имеются следующие виды червей:
  7) Интернет – черви, например почтовые черви с зараженными письмами;
  8) Сетевые (LAN) черви – ищут компьютеры в сети и заражают их;
  9) Эмуляторы DDoS – атак, то есть атака сервера запросами со многих компьютеров (распределенная атака) для создания ситуации «отказ в обслуживании» и вывода оборудования из рабочего состояния;
  10) Дроппер – программа – инсталлятор вируса в систему.
- Захватчик паролей — это программа, специально предназначенная для воровства паролей. При попытке обращения пользователя к терминалу системы на экран выводится ложная информация. Пытаясь организовать вход, пользователь вводит имя и пароль, которые пересылаются владельцу программы - захватчика, далее ввод и управление возвращаются к операционной системе, но конфиденциальные данные уже похищены. Разновидностью захватчика паролей являются программы – фишинги и фарминги, специально предназначенные для похищения финансовой информации.

Методы и средства защиты информации

Обеспечение информационной безопасности основывается на следующих принципах:

- Системный подход к построению системы защиты, означающий оптимальное сочетание взаимосвязанных организационных программных, аппаратных, физических и других свойств, подтвержденных практикой создания отечественных и зарубежных систем защиты и применяемых на всех этапах технологического цикла обработки информации.
- Принцип непрерывного развития системы. Способы реализации угроз информации в ИС непрерывно совершенствуются, и поэтому обеспечение безопасности не может быть одноразовым актом. Это непрерывный процесс, заключающийся в обосновании и реализации наиболее рациональных методов, способов и путей совершенствования систем безопасности, непрерывном контроле, выявлении ее узких и слабых мест, потенциальных каналов утечки информации и новых способов несанкционированного доступа.
- Разделение и минимизация полномочий по доступу к обрабатываемой информации и процедурам обработки, т. е. предоставление как пользователям, так и самим работникам ИС, минимума строго определенных полномочий, достаточных для выполнения ими своих служебных обязанностей.
- Обеспечение надежности системы защиты, т. е. невозможность снижения уровня надежности при возникновении в системе сбоев, отказов, преднамеренных действий взломщика или непреднамеренных ошибок пользователей и обслуживающего персонала.
- Обеспечение контроля за функционированием системы защиты, т.е. создание средств и методов контроля работоспособности механизмов защиты.
- Обеспечение всевозможных средств борьбы с вредоносными программами.

К методам и средствам обеспечения безопасности информации можно отнести:

- Методы физического преграждения пути злоумышленнику к защищаемой информации (к

аппаратуре, носителям информации и т.д.).

- Управление доступом — идентификацию пользователей, персонала и ресурсов системы, проверку полномочий, регистрацию обращений к защищаемым ресурсам, сигнализацию, отключение, отказ в запросе и т.п. при попытках несанкционированных действий.
- Шифрование — криптографическое закрытие информации.
- Программные средства — это специальные программы и программные комплексы, предназначенные для защиты информации.

Так как первые два метода относятся к организационным мероприятиям, то более подробно рассмотрим методы шифрования и программные методы.

Шифровально - криптографическая система защиты информации основана на использование специальных алгоритмов. Действие таких алгоритмов запускается уникальным числом, называемым шифрующим ключом.

Стойкость любой системы защищенной компьютерной сети определяется степенью секретности используемого в ней ключа. Но при этом этот ключ должен быть известен другим пользователям сети, чтобы они могли свободно обмениваться зашифрованными сообщениями.

Современная криптография знает два типа криптографических алгоритмов: классические алгоритмы, основанные на использовании закрытых, секретных ключей, и новые алгоритмы с открытым ключом, в которых используются один открытый и один закрытый ключ (эти алгоритмы называются также асимметричными). Кроме того, существует возможность шифрования информации и более простым способом — с использованием генератора псевдослучайных чисел.

Существует довольно много различных алгоритмов криптографической защиты информации. Среди них можно назвать алгоритмы DES, Rainbow (CIIJA); FEAL-4 и FEAL-8 (Япония); B-Crypt (Великобритания); алгоритм шифрования по ГОСТ 28147 — 89 (Россия) и ряд других.

Наиболее перспективными системами криптографической защиты данных сегодня считаются асимметричные криптосистемы, называемые также системами с открытым ключом. Идея состоит в том, что ключ, используемый для зашифровывания, отличен от ключа расшифровывания. При этом ключ зашифровывания не секретен и может быть известен всем пользователям системы. Однако расшифровывание с помощью известного ключа зашифровывания невозможно. Для расшифровывания используется специальный, секретный ключ.

Известно несколько криптосистем с открытым ключом. Например, система RSA. RSA— это система коллективного пользования, в которой каждый из пользователей имеет свои ключи зашифровывания и расшифровывания данных, причем секретен только ключ расшифровывания. Широко используется для цифровых подписей, подтверждающих подлинность передаваемых документов и сообщений.

Недостатками системы типа RSA являются: длина ключа порядка 300— 600 бит и низкая скорость работы.

Теперь рассмотрим программную защиту. В первую очередь это антивирусные программы и файерволы. Наиболее известны среди них следующие (по данным портала comss.ru):

- Avast! Internet Security – 21 место за 2012 г.
- Avira Antivir – 15 место за 2012 г.
- Comodo Internet Security– 2 место за 2012 г.
- Dr.Web – 19 место за 2012 г.
- Emisoft Anti-Malware и Internet Security – 1 и 4 место за 2012 г.
- Eset NOD32 – 20 место за 2012 г.
- G Data Internet Security – 11 место за 2012 г.
- Kaspersky Internet Security – 18 место за 2012 г.

- McAfee – 13 место за 2012 г.
- Norton Antivirus – 7 место за 2012 г.
- Panda Software – 14 место за 2012 г.

Все они являются резидентными (работают на уровне системных программ операционной системы), выполняют поиск и удаление вирусов, осуществляют эвристический анализ файлов и системных областей. Лучшие из них имеют базу данных на более чем 800 тысяч сигнатур различных вирусов и постоянно ее обновляют. Стандартными действиями при обнаружении опасности являются:

- Информировать пользователя.
- Вылечить зараженные файлы.
- Удалить зараженные файлы.
- Переименовать зараженные файлы.
- Переместить в карантин.

Как правило, задачи управления и контроля на предприятии решаются административной группой, в которую входят администратор безопасности, менеджер безопасности и операторы. Основной задачей специалиста по информационным технологиям, входящему в эту группу, является обеспечение информационной системы наиболее эффективным антивирусным программным обеспечением, регулярное обновление антивирусных баз данных, создание резервных копий важной информации, создание точек восстановления системы, поддержка службы паролей и т.д.

Программы подготовки документов

Текстовые процессоры

Одним из актуальных направлений использования персональных компьютеров является подготовка документов. Для этого предназначены специальные программы – текстовые редакторы и текстовые процессоры. Эти две разновидности программ похожи. Различие заключается в функциональности и наборе поддерживаемых типов файлов. Текстовые редакторы предназначены исключительно для набора и редактирования текста и не включают в себя средств сложного форматирования. Наиболее известный пример текстового редактора – Блокнот, входящий в состав ОС Windows.

Весь текст в редакторе отображается одним и тем же шрифтом (как правило, моноширинным), отдельные фрагменты нельзя выделить ни курсивом, ни полужирным, нельзя задать для разных абзацев разные отступы или окрасить их фон в разные цвета. Более того, даже если мы зададим в редакторе настройки отображения для всего текста, они не сохранятся в файле и не будут видны на другом компьютере. Дело в том, что файлы, с которыми работают текстовые редакторы (они имеют расширение TXT), ничего, кроме последовательности символов, не содержат.

Возможностей текстовых редакторов в большинстве случаев недостаточно. Именно поэтому так широко распространены текстовые процессоры, позволяющие не только набирать и корректировать текст, но и оформлять его.

Одной из задач специалиста по информатике является выбор программы подготовки текстов, максимально функциональной для обслуживаемого направления, но максимально бюджетной. С этих позиций необходимо провести оценку достоинств и недостатков ПО этого направления.

Стандартом де-факто для текстовых процессоров в настоящее время стал Microsoft Word. Кроме него, имеются альтернативные продукты, например, OpenOffice.org Writer или программа WordPad, входящая в состав всех версий Windows – ее возможностей вполне хватает для решения многих задач.

Часто текстовые процессоры входят в состав пакетов офисных приложений, в то

время как самостоятельные программы используются реже, так как «пакетный» текстовый процессор оказывается более функциональным – за счет интеграции с входящими в пакет приложениями. Кроме того, вместе с офисным пакетом обычно поставляются галерея изображений и набор шрифтов.

Рассмотрим ставший стандартом де-факто пакет Microsoft Office. Он поставляется в нескольких редакциях, основные компоненты пакета следующие:

- Microsoft Office Word — текстовый процессор. Позволяет подготавливать документы различной сложности. Поддерживает OLE (включение в документ объектов – документов из других программ), подключаемые модули сторонних разработчиков, шаблоны и многое другое. Продукт занимает ведущее положение на рынке текстовых процессоров, и его форматы используются как стандарт де-факто в документообороте большинства предприятий.
- Microsoft Office Excel — табличный процессор. Поддерживает все необходимые функции для создания электронных таблиц любой сложности. Последняя версия использует формат OOXML с расширением «.xlsx», более ранние версии использовали двоичный формат с расширением «.xls».
- Microsoft Office Outlook — персональный коммуникатор. В состав Outlook входят: календарь, планировщик задач, записки, менеджер электронной почты, адресная книга. Поддерживается совместная сетевая работа.
- Microsoft Office PowerPoint — приложение для подготовки презентаций.
- Microsoft Office Access — приложение для управления базами данных.
- Microsoft Office InfoPath — приложение сбора данных и управления ими.
- Microsoft Office Communicator — предназначен для организации всестороннего общения между людьми посредством простого обмена мгновенными сообщениями, а также проведения голосовой и видеобеседы.
- Microsoft Office Publisher — приложение для подготовки публикаций.
- Microsoft Office Visio — приложение для работы с бизнес-диаграммами и техническими диаграммами — позволяет преобразовывать концепции и обычные бизнес-данные в диаграммы.
- Microsoft Office Project — управление проектами.
- Microsoft Query — просмотр и отбор информации из баз данных.
- Microsoft Office OneNote — приложение для записи заметок и управления ими.
- Microsoft Office Picture Manager — работа с рисунками.

Ранее в Microsoft Office входило приложение Microsoft FrontPage (программа для создания сайтов), однако Microsoft приняла решение исключить это приложение из Office и прекратить его разработку. В Microsoft Office 2007 программа FrontPage была заменена на Microsoft SharePoint Designer.

Хотя каждый текстовый процессор обычно имеет свой формат, большинство, как правило, способно открывать и сохранять файлы других форматов. Вот некоторые из них.

- RTF – формат, разработанный Microsoft специально для форматированных текстов. Его поддержку обеспечивают практически все текстовые процессоры, а для самых простых (например, WordPad) он является основным.
- DOC – формат, который использовался как основной в Microsoft Word до версии 2003 включительно. Он читается практически всеми текстовыми процессорами, и сохранение документов в этом формате в большинстве случаев доступно.
- DOCX пришел на смену DOC в Microsoft Office Word 2007 и, по замыслу разработчиков, должен был получить широкое распространение: спецификации его открыты, так что обеспечить его поддержку теоретически ничто не мешает. На практике же он оказался весьма сложным, поэтому совместимых с ним программ немного.
- ODT – открытый формат, разработанный раньше DOCX. Он поддерживается заметно

большим числом приложений; в частности, OpenOffice.org Writer использует его в качестве основного.

- HTML – формат, в котором хранятся веб-страницы в Интернете. Он хорош прежде всего своей универсальностью. Использует язык гипертекстовой разметки и не является основным ни для одного текстового процессора, но большая их часть с HTML совместима.

Хотя программам от Microsoft Office нет равноценной альтернативы, но задачей специалиста по информатике является обеспечение функционирования компьютерных систем с оптимальным соотношением цена – качество. С этих позиций рассмотрим функциональность свободнораспространяемых текстовых процессоров, например, OpenOffice.org Writer, IBM Lotus Symphony, WordPad, Angel Writer, PolyEdit и AbiWord.

Все необходимые средства оформления страниц и текста предоставляют только OpenOffice.org Writer и «Документы IBM Lotus Symphony». WordPad не имеет средств оформления страниц. Со стилями оформления работают все перечисленные, за исключением Angel Writer, PolyEdit и WordPad.

Функция проверки орфографии есть во всех процессорах, кроме WordPad. Но в Angel Writer, Atlantis Nova и «Документы IBM...» русскоязычных словарей для проверки нет. С проверкой русской пунктуации и синтаксиса не справляется ни одна программа.

Лучшая поддержка распространенных форматов на уровне чтения у «Документы IBM...», Writer и AbiWord. Writer не поддерживает запись в DOCX, а «Документы IBM...» – в DOCX и HTML. Зато обе они позволяют экспортировать текст в файлы PDF. По количеству входящих в комплект дополнительных файлов и модулей лучше всего Writer и «Документы IBM...». Writer единственный имеет набор шаблонов на русском языке и редактор формул, но уступает «Документам IBM...» по количеству и качеству изображений в клипарт-галерее.

С точки зрения удобства управления и быстроты освоения - интерфейс почти всех перечисленных программ русифицирован, кроме Atlantis Nova. Для PolyEdit русификатор надо устанавливать отдельно. Наиболее прост в работе элементарный WordPad. Наиболее сложным в освоении является текстовый процессор OpenOffice.org Writer: большую часть его функций приходится выискивать в огромном меню. Текстовый процессор «Документы IBM...» удобнее, так как имеет совершенно иной интерфейс, в котором актуальные в данный момент функции открываются на боковой панели. Например, если редактируется текст, то справа появляется панель с настройками свойств текста.

Подготовка схем в системе Visio

Важным элементом в работе инженера и организатора является проектирование разнообразных *схем* — организационных, структурных, маршрутных, логических и т. п. Их общей чертой является наличие некоторого стандартного для каждой предметной области набора типовых элементов (например, блок-схемы алгоритмов).

Вычерчивание схем возможно средствами любого графического редактора или универсального текстового процессора класса Microsoft Word. Однако в указанных целях гораздо удобнее воспользоваться системой Microsoft Visio. В ее состав входят несколько десятков комплектов трафаретов *(stencils),* ориентированных на весьма разнообразные применения. Пользователь, выбрав нужный трафарет, переносит из него в рабочий лист необходимые блоки, а затем размножает их, масштабирует, перемещает, соединяет линиями и стрелками, снабжает надписями и т. п.

Visio является полноценным Windows-приложением, и разработанные в ней схемы переносимы в другие приложения (в частности, поддерживается механизм OLE). Технология работы, содержание меню, состав панели инструментов и назначение «горячих клавиш» максимально приближены к принятым в продуктах Microsoft Office. Ценным достоинством Visio является возможность изготовления с ее помощью крупномасштабных плакатов, склеиваемых из листов стандартного формата.

Экспорт и импорт файлов для работы с другими приложениями производится в форматах CGM, EPS, PCX, TIF- BMP, WMF через меню *File (Файл)*. Соответственно *Visio* можно использовать для преобразования форматов загруженных файлов. По команде *Tools \ Options (Сервис | Параметры)* можно корректировать пути для записи формируемых рисунков.

Рабочее окно программы Visio имеет стандартный вид продуктов фирмы Microsoft и легко осваивается самостоятельно по аналогии с другими программами, рис.16.

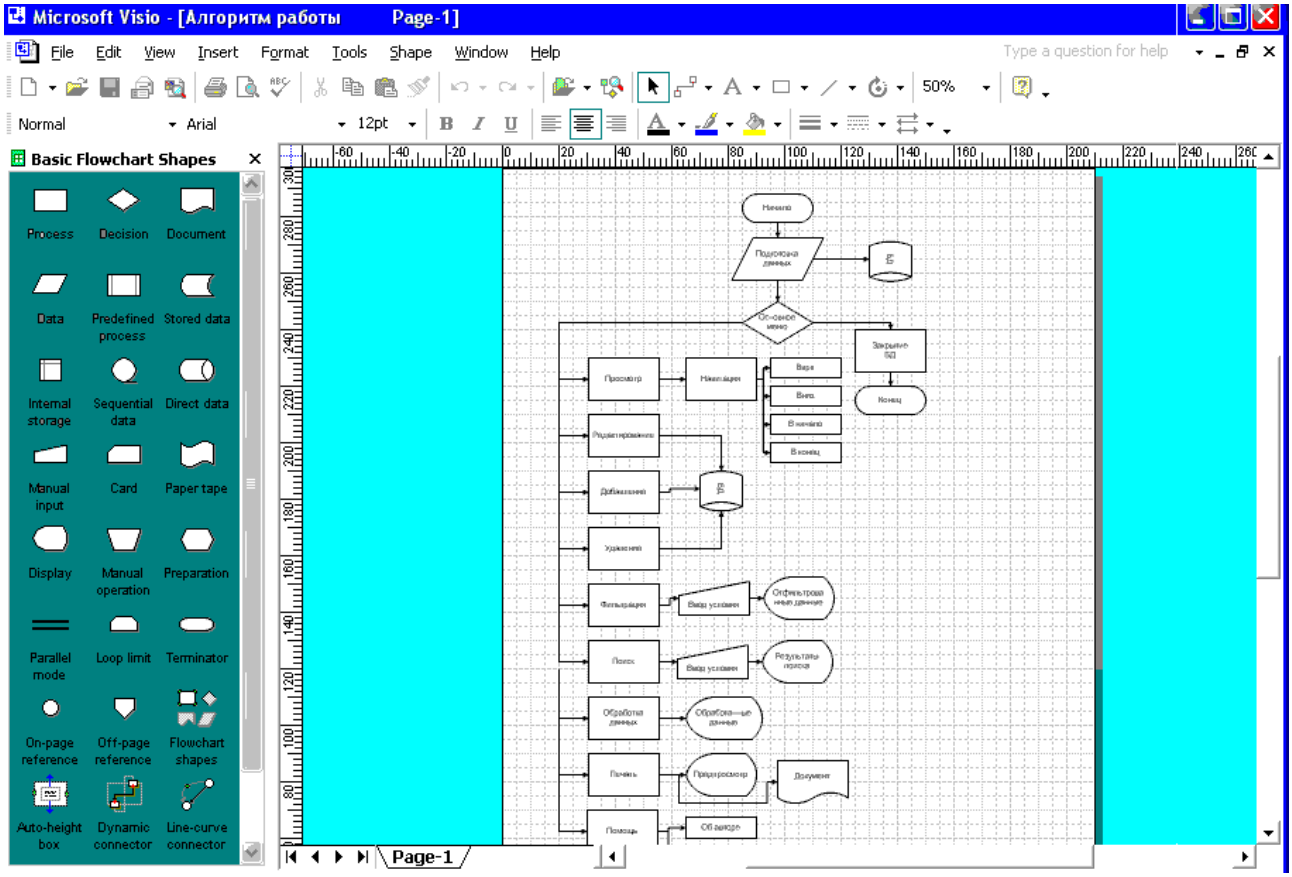


Рис.16. Основное окно разработки графического документа в Microsoft Visio.

Однако для работы с системой существенно знание базовых определений и основных библиотечных трафаретов.

Базовые определения Visio

- *Фигура (Shape)* — базовый графический элемент, из комбинации которых создается схема (drawing).
- *Трафарет (Stencil)* — совокупность базовых элементов, объединяемых общностью предметной области.
- *Стиль (Style)* — совокупность форматов (размер шрифта, толщина и цвет линий, заполнение), имеющая имя и сохраняемая со стандартом или схемой.
- *Шаблон (Template)* — файл, содержащий стили и установки для некоторого вида схем и открывающий соответствующие трафареты. Создание новой схемы начинается с вызова необходимого трафарета или стандарта.

Стандартные трафареты Visio

Перечислим некоторые из категорий трафаретов, сохраняя для удобства работы с системой и английские названия.

- Block diagram — три комплекта геометрических фигур, стрелок и соединителей в большом числе вариантов, с оттенением и объемной перспективой.

- Building Plan – средства построения разнообразных планов: помещений, начиная с плана жилого дома, кончая планом сайтов, размещения электрооборудования и т.д.
- Database – средства для построения информационных моделей баз данных.
- Electrical Engineering — построение электрических и логических схем, инверторы, схемы И, ИЛИ, диод, триод, «земля», батарея, резистор, конденсатор, сигналы.
- Flowchart — элементы схем алгоритмов (все стандартизованные обозначения, внутри- и межстраничные соединители, средства разветвления и соединения потоков, базы данных).
- Forms and Charts — разнообразные графики и рамки для текста, шаблоны для таблиц и графиков (в частности, заготовки для логарифмической сетки), текстовые блоки для набора разными шрифтами.
- Map — обширная группа наборов географических карт, изображающих контуры штатов США, стран, регионов и континентов, а также обозначений на картах и схемах и знаков дорожного движения, включая 3D элементы.
- Mechanical Engineering – средства для рисования чертежей механики.
- Network — элементы сетей связи и компьютерных сетей: устройства ЭВМ; малые и большие ЭВМ разных типов, типичные сетевые структуры; спутниковая связь и т.д.
- Organization Chart — средства построения планов организации процесса менеджмента, включая мастера - помощника.
- Process Engineering – средства составления инженерных планов производства, техпроцесса, работы оборудования.
- Project Schedule — средства сетевого планирования: календарный план, PERT-диаграммы и графики Ганта.
- Software – средства представления взаимодействия и проектирования программного обеспечения.
- Web Diagram — разработка плана сайта, планов взаимодействия сайтов и т.д.

Разумеется, полнота наборов не гарантируется (в частности, для электрических и функциональных схем вычислительных машин она явно недостаточна). Однако в Visio имеются средства создания авторских шаблонов, что снимает упомянутое затруднение.

Этапы подготовки для решения задач на ЭВМ

На компьютерной технике могут решаться задачи различного характера, например, научно – инженерные разработки программного обеспечения, моделирования процессов, управления производством, обучения, проектные разработки изделий разнообразного применения и т.д.

Если касаться научно – инженерных задач, то для решения их требуется выполнить следующие этапы:

- Постановка задачи;
- Математическая формулировка решения;
- Выбор и обоснование метода решения;
- Разработка алгоритма вычислительного процесса;
- Выбор языка программирования и написание программы;
- Отладка программы;
- Решение задачи на программном уровне и анализ результата.

При решении различных задач некоторые этапы могут и отсутствовать, но, в общем случае, этапы должны быть соблюдены и часто требуют итерационного подхода, так как, например, анализ результата может привести к необходимости пересмотра метода и алгоритма решения.

Постановка задачи предполагает формулировки конечных целей, а, если речь идет о

работе с заказчиком на разработку программного продукта, то утверждение исходных и выходных данных, словесное описание алгоритма, утверждение экранных форм и отчетных документов, согласование сроков не только разработки, но и тестирования и сопровождения программы. В конечном результате должен быть составлен и подписан договор на проведение работ, имеющий юридическую силу.

Математическая формулировка, выбор и обоснование метода решения. Данные этапы предполагают применение разработчиком математического аппарата к решению поставленной задачи и обоснованный выбор метода обработки информации. Дело в том, что обрабатываемая информация может потребовать, например:

- вычисления формульных выражений;
- сохранения полученных и исходных данных на диск и считывания их;
- сортировки данных – по возрастанию, убыванию, алфавиту и т.д.;
- поиска данных;
- определения экстремальных значений – максимума, минимума и т.д.;
- выдачи заказчику полученных данных, как в виде таблиц, так и графиков;
- печати документов по определенному шаблону;
- транспонирования данных из одной программы в другую, например в Excel;
- сетевого обмена, например, по сети Интернет и др.;
- программа может потребовать решения задач телеуправления, сбора данных с каких-то приборов или датчиков, что влечет за собой необходимость написания драйверов связи с оборудованием и так далее.

Ошибки вычислений

При вычислении формульных выражений необходимо учитывать возможность возникновения ошибок вычисления, например, рассмотрим задачу нахождения одного из корней уравнения:

$x^2 + 0.4002\,x + 0/00008 = 0$, используя вычисления до четвертой значащей цифры.

Применяя формулу:

$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a},$$

Получим ответ: x = -0.00015.

Если же при вычислении учитывать восемь значащих цифр, то x = -0.0002, таким образом, результат имеет ошибку 25%. Поэтому разработка программы вычислении формульных выражений должна проводиться с анализом ошибок вычисления, по крайней мере необходимо предусмотреть «прогон» программы с различным числом значащих цифр исходных данных.

Ошибки бывают относительные и абсолютные. **Абсолютной ошибкой** считается разность между истинным значением величины и ее приближенным (например, округленным). **Относительной ошибкой** считается отношение абсолютной ошибки к приближенному значению величины.

Кроме этого, наблюдаются ошибки в исходной информации, например, число π невозможно представить точно ограниченным числом цифр, или показания измеряемой величины зависят от точности прибора. Отсюда наблюдаются **ошибки ограничения** и **ошибки округления**. Далее, при программировании формул могут наблюдаться **ошибки распространения**, или ошибки математических операций. Например:

- абсолютная ошибка суммы: $\mathbf{e_{x+y} = e_x + e_y}$,
- абсолютная ошибка разности: $\mathbf{e_{x-y} = e_x - e_y}$,
- абсолютная ошибка произведения: $\mathbf{e_{x*y} = x*e_y + y*e_x}$,

- абсолютная ошибка от деления: $\mathbf{e_{x/y} = e_x/y - e_y*x/y^2}$.

Отсюда при выполнении длинных вычислений может наблюдаться накопление ошибки расчета, что необходимо учитывать при программировании и анализе результата.

Сохранение данных

Задача сохранения данных на жесткий диск компьютера предполагает работу с файловой системой компьютера. Файлом называется логически связанная совокупность данных определенной длины, имеющая свое имя. Файл может хранить текст исполняемой программы, документы, графические изображения и т.д. Для любой операционной системы имеются стандартные расширения файлов, указывающих на их содержимое. Например, некоторые расширения для ОС Windows:

- arj – архив архиватора ARJ;
- asm – текст программы на языке ассемблер;
- bat – командный файл;
- bin – двоичный файл;
- bmp – графическое изображение;
- com – исполняемая программа с абсолютным адресом загрузки;
- doc – файл текстового документа;
- exe – исполняемая программа;
- ini – информационный файл;
- pas – текст программы на языке Паскаль;
- sys – системный файл (драйвер).

Несомненными достоинствами любой ОС является то, что она освобождает пользователя от работ по физическому размещению файлов и следит за их целостностью. Для доступа к файлам ОС использует таблицу размещения файлов – FAT, FAT32 (File Allocation Table). FAT содержит информацию о расположении файлов, свободном месте на диске, неисправных блоках, код формата диска и т.д.

Более современный способ размещения файлов - отказоустойчивая система NTFS. Данный способ более надежен, так как он основан на транзакциях – любое действие с файлом, если оно верно, то утверждается, а если неверно, то происходит откат действия и файл приводится в исходное состояние. Далее, в NTFS ведется постоянное журналирование файлов с помощью механизма MFT (Meta File Table), более надежного, чем в системе FAT, размер файлов теоретически неограничен и поддерживаются длинные имена файлов.

С другой стороны, кроме задачи инженера в области информационных технологий выбора и настройки ОС, типов файлов, выбора способа размещения файлов, умения создавать элементарные программы для работы с файлами и т.д., может возникнуть проблема хранения больших структурированных массивов данных. Отсюда возникает необходимость применения баз данных (БД), применения систем управления базами данных (СУБД), что значительно усложнит решение исходной задачи и потребует дополнительных знаний.

Сортировка данных

Упомянутая ранее задача сортировки данных требует от инженера знания математических подходов к этой проблеме. Критерием выбора соответствующего метода служит его эффективность (скорость) или простота реализации. Наиболее известны следующие методы сортировки для чисел:

- Метод простого выбора. Ищется максимальный элемент и меняется местами с последним элементом списка. Далее процедура повторяется с оставшимися элементами. Метод эффективен при отсутствии одинаковых элементов.
- Метод простого обмена (пузырьковый метод). Метод заключается в

последовательном сканировании списка по парам. Каждая пара чисел меняется местами, если левое число больше правого. В результате самое большое число переместится в конец списка. Затем действие повторяем для оставшихся пар.

- Сортировка методом слияний. Основана на разбиении исходного списка (массива) на несколько частей, которые сортируются по отдельности, а потом «сливаются» и т.д.
- Для сортировки баз данных все значительно проще. Здесь применяется индексирование и SQL - запросы.

Поиск данных.

Поиск данных для списка (массива) чисел также может быть основан на простейших математических методах:

- Линейный поиск. Основан на сканировании списка данных до тех пор, пока очередной элемент не будет равен ключу поиска.
- Метод половинного деления. Делим список пополам. Ищем нужный элемент, если не найден, делим оставшуюся часть пополам и снова ищем, если не находим, процедуру деления и поиска повторяем. Метод более эффективен.
- Для баз данных, как правило, используются готовые операторы поиска и SQL - запросы.

Определение экстремальных значений

Определение экстремальных значений для массива чисел может проводиться аналогично методам поиска данных, а для функциональных зависимостей применяются специальные численные методы. Подробное изложение численных методов не входит в данное учебное пособие, поэтому ограничимся только их перечислением:

- Метод равномерного поиска.
- Метод поразрядного приближения.
- Метод дихотомии.
- Метод золотого сечения.
- Метод квадратичной интерполяции – экстраполяции.
- Метод координатного спуска.

Остальные требования, входящие в выбор и обоснование метода решения, зависят от аппаратуры, выбранного программного обеспечения, среды программирования и т.д.
Далее рассмотрим задачу разработки алгоритмов.

Алгоритмизация и программирование

Алгоритм и его свойства

Решение задач на компьютере основано на понятии алгоритма. Алгоритм – это точное предписание, определяющее вычислительный процесс, ведущий от задаваемых начальных данных к конечному результату. Разработка алгоритма является сложным и трудоемким процессом. Алгоритмизация – это техника разработки (составления) алгоритма для решения задач на вычислительном устройстве — ЭВМ, ПЛК или микроконтроллере.

Для записи алгоритма решения задачи применяются следующие формы их представления:

- Словесно - формульное описание;
- Блок-схема (схема графических символов);
- Алгоритмические языки;
- Операторные схемы;
- Псевдокод;

- Графы.

Формульно - словесный способ записи алгоритма характеризуется тем, что описание осуществляется с помощью слов и формул, как было показано в предыдущем разделе. Содержание последовательности этапов выполнения алгоритмов записывается на естественном профессиональном языке предметной области в произвольной форме.

Графический способ описания алгоритма (блок - схема) получил самое широкое распространение. Для графического описания алгоритмов используются схемы алгоритмов или блочные символы (блоки), которые соединяются между собой линиями связи, рис.17. Условные обозначения блоков приведены в ГОСТ 19.701-90.

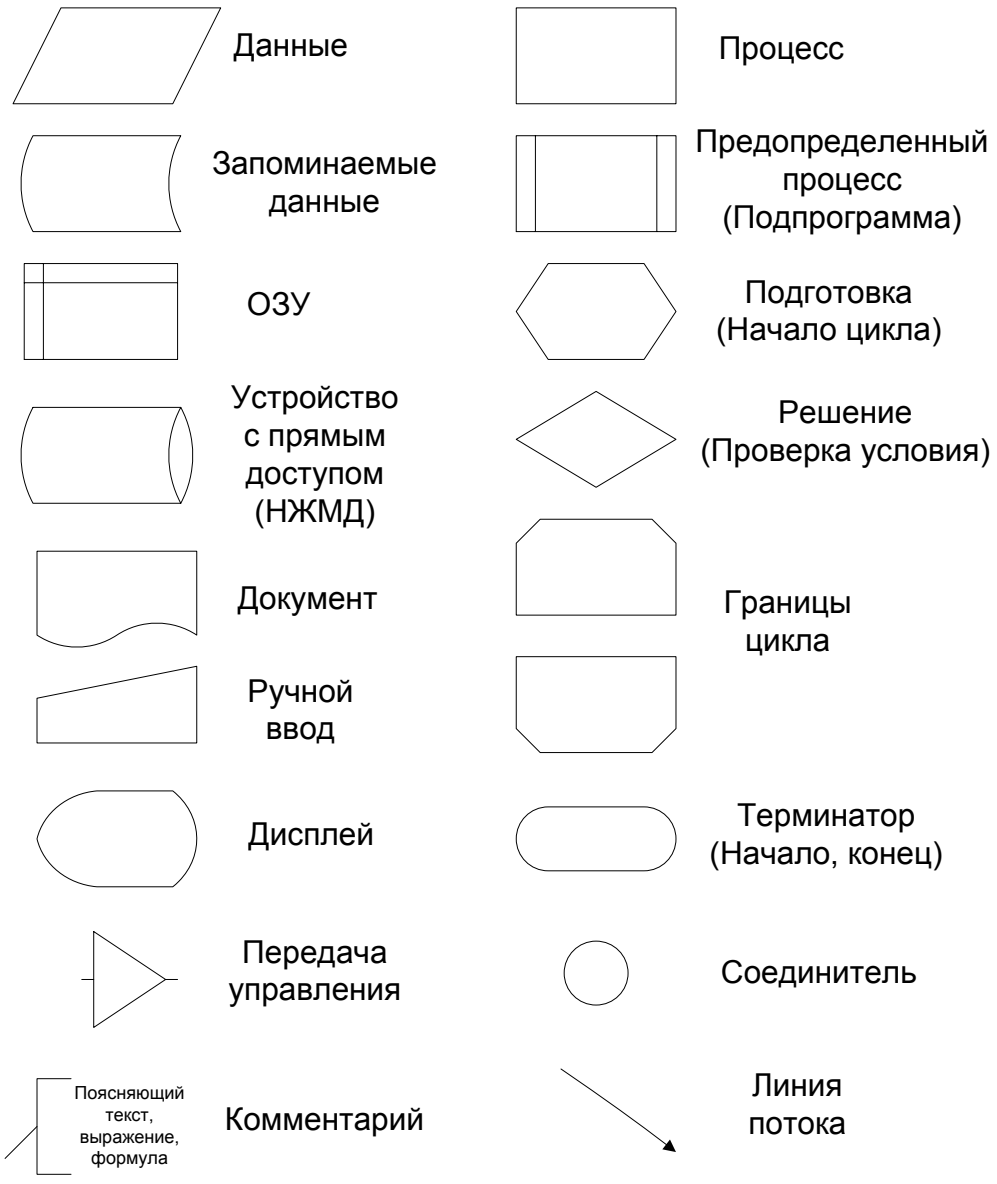


Рис.17. Условные обозначения основных алгоритмических блоков

Порядок выполнения этапов алгоритма указывается стрелками, соединяющими блоки, которые размещаются сверху вниз и слева на право. Нумерация блоков производится в порядке их размещения в схеме.

Алгоритмические языки - это специальное средство, предназначенное для записи алгоритмов в аналитическом виде. Алгоритмические языки близки к математическим выражениям и к естественным языкам. Каждый алгоритмический язык имеет свой словарь.

Алгоритм, записанный на алгоритмическом языке, выполняется по строгим правилам этого конкретного языка.

Операторные схемы алгоритмов. Суть этого способа описания алгоритма заключается в том, что каждый оператор обозначается буквой (например, А – арифметический оператор, Р – логический оператор и т.д.).

Операторы записываются слева направо в последовательности их выполнения, причем, каждый оператор имеет индекс, указывающий порядковый номер оператора. Алгоритм записывается в одну строку в виде последовательности операторов.

Псевдокод – система команд абстрактной машины. Этот способ записи алгоритма с помощью операторов близких к алгоритмическим языкам.

Графы. При использовании графов последовательность операций отображается в виде кружков (состояний) и стрелок, показывающих направление операции. Например, граф вычисления выражения z = (x + y) * n , приведен на рис.18. Дальнейшим развитием графов являются сети Петри.

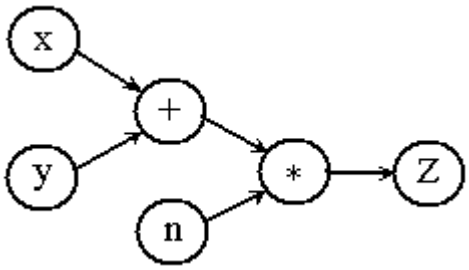


Рис.18. Пример простейшего графа

Типы алгоритмов

По структуре выполнения алгоритмы и программы делятся на три вида, рис.19:

- Линейные
- Ветвящиеся
- Циклические

Линейный алгоритм – это такой алгоритм, в котором все действия выполняются последовательно друг за другом и только один раз. Схема представляет собой последовательность блоков, которые располагаются сверху вниз в порядке их выполнения.

Алгоритмы разветвляющейся структуры. На практике часто встречаются задачи, в которых в зависимости от первоначальных условий или промежуточных результатов необходимо выполнить вычисления по одним или другим формулам. Такие задачи можно описать с помощью алгоритмов разветвляющейся структуры. В таких алгоритмах выбор направления продолжения вычисления осуществляется по итогам проверки заданного условия. Ветвящиеся процессы описываются оператором IF (условие).

Циклические вычислительные процессы. Для решения многих задач характерно многократное повторение отдельных участков вычислений. Для решения таких задач применяются циклические алгоритмы. Цикл – последовательность команд, которая повторяется до тех пор, пока не будет выполнено заданное условие. Циклическое описание многократно повторяемых процессов значительно снижает трудоемкость написания программ.

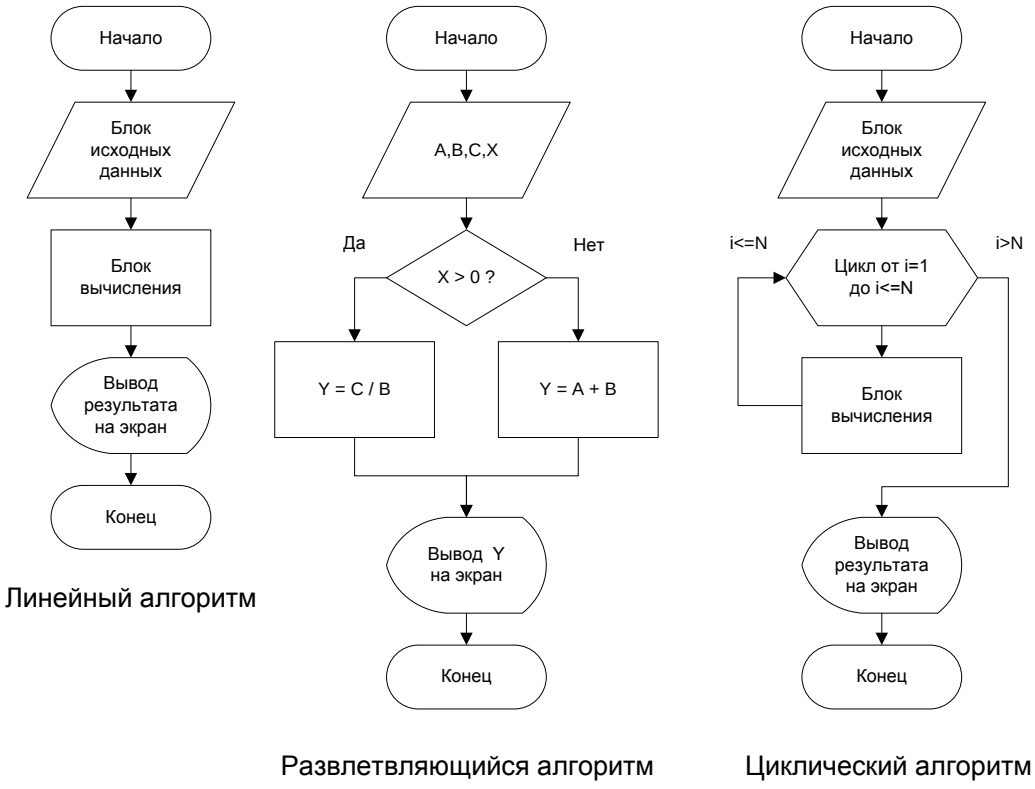


Рис.19. Виды основных алгоритмов

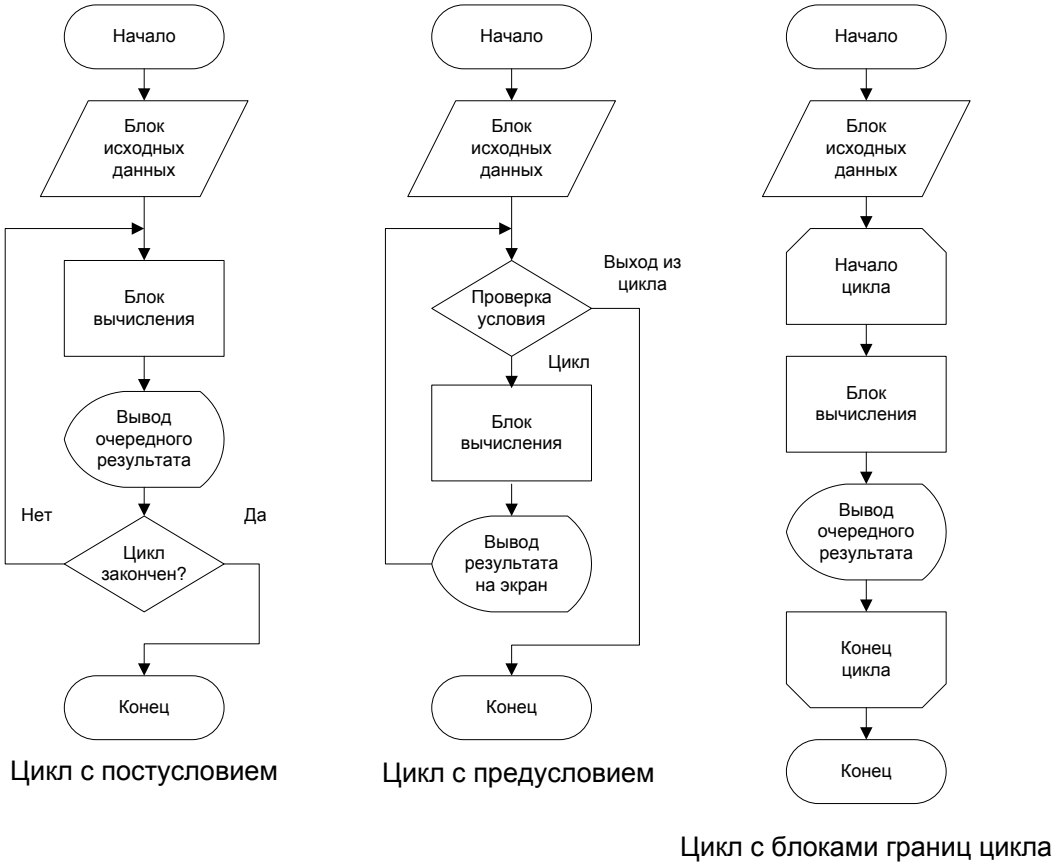


Рис.20. Разновидности циклических алгоритмов

Существует несколько схем циклических вычислительных процессов. Бывают циклы с известным числом повторений и итерационные циклы. При итерационном цикле выход из

тела цикла, как правило, происходит при достижении заданной точности вычисления. На рис.17 приведен циклический алгоритм с заданным числом повторений N, реализованный на блоке «Подготовка». Цикл также может быть организован и блоком «Проверка условия», причем может применяться как предпроверка, так и постпроверка. Особенностью первой схемы является то, что проверка условия выхода из цикла проводится до выполнения тела цикла. В том случае, если условие выхода из цикла выполняется, то тело цикла не выполняется ни разу. Особенностью второй схемы является то, что цикл выполняется хотя бы один раз, так как первая проверка условия выхода из цикла осуществляется после того, как тело цикла выполнено, рис.20.

Языки программирования

Все языки программирования созданы для решения задач с использованием математики. Наиболее близким к человеку является язык математической записи условия задачи и её решения. Имеется в виду не только строго формализованные правила математических обозначений, а изложение с применением естественных для людей приёмов оформления и комментирования.

В настоящее время в мире существует несколько сотен реально используемых языков программирования. Для каждого есть своя область применения.

Так как алгоритм - это последовательность предписаний, выполнив которые можно за конечное число шагов перейти от исходных данных к результату, то, в зависимости от степени детализации предписаний, и определяется уровень языка программирования - чем меньше детализация, тем выше уровень языка.

По этому критерию можно выделить следующие уровни языков программирования:

- машинные;
- машинно - ориентированные (ассемблеры);
- машинно-независимые (языки высокого уровня).

Машинные языки и машинно-ориентированные языки - это языки низкого уровня или языки первого поколения - 1GL, требующие указания мелких деталей процесса обработки данных. Идеальный вариант для разработки драйверов технических устройств и программирования микроконтроллеров. Как правило, это ассемблер.

С появлением ЭВМ 2-го поколения появились языки второго поколения (2GL), в которых степень интеграции действий была на порядок выше. Это такие языки, как макроассемблер и автокод.

Языки высокого уровня или языки третьего поколения - 3GL имитируют естественные языки, используя некоторые слова разговорного языка и общепринятые математические символы. Эти языки более удобны для человека.

Языки высокого уровня (3GL) делятся на:

- **процедурные** (алгоритмические) (Basic, Pascal, C и др.), которые предназначены для однозначного описания алгоритмов; для решения задачи процедурные языки требуют в той или иной форме явно записать процедуру ее решения;
- **логические** (Prolog, Lisp и др.), которые ориентированы не на разработку алгоритма решения задачи, а на систематическое и формализованное описание задачи с тем, чтобы решение следовало из составленного описания;
- **объектно-ориентированные** (Object Pascal, C++, Java и др.), в основе которых лежит понятие объекта, сочетающего в себе данные и действия над нами. Программа на объектно-ориентированном языке, решая некоторую задачу, по сути описывает часть мира, относящуюся к этой задаче. Описание действительности в форме системы взаимодействующих объектов естественнее, чем в форме взаимодействующих процедур.
- **Динамические языки** - это такие языки программирования и такие трансляторы,

которые позволяют определять типы данных и осуществлять синтаксический анализ и трансляцию "на лету", непосредственно на этапе выполнения. К динамическим языкам относятся: Basic, Perl, Tcl, Python, PHP, Ruby, Smalltalk, JavaScript.

- **Декларативные языки программирования.** К ним относятся функциональные и логические языки программирования. В этих языках не производится алгоритмического действия явно, то есть алгоритм не задается программистом, а строится самой программой. В декларативных языках задается, производится построение какой-либо структуры или системы, то есть декларируются (объявляются) какие-то свойства создаваемого объекта. Эти языки получили широкое применение в системах автоматизированного проектирования (САПР), в так называемых CAD-пакетах, в моделировании, системах искусственного интеллекта, например, MathCAD, MatLAB.

Любая программа, написанная на языке высокого уровня, для исполнения должна быть переведена в машинный код. Задачу такого преобразования выполняет транслятор. Трансляторы бывают двух видов: компиляторы и интерпретаторы. Компилятор преобразует исходный код программы целиком в независимый исполняемый, как бы самодостаточный, а интерпретатор преобразует исходный код «на лету», по мере исполнения программы. Для интерпретатора существуют различные варианты, например, исполнение программы с помощью сопутствующего RUN-TIME ядра, преобразование исходной программы в промежуточный платформонезависимый язык, а затем уже исполнение и т.д.

В последнее время во всем мире широкое развитие приняла концепция быстрой разработки программ, основанная на автогенерации некоторого готового шаблона исходного кода программы. При этом программисту предоставляются готовые решения, библиотеки, объекты, классы и т.д. Если раньше, например, типовым языком программирования являлся Паскаль, Си или Бейсик, заставлявший пользователя прописывать вручную любые действия, то с появлением концепции объектно – ориентированного программирования программисту стали предлагаться готовые библиотеки объектов и разработка программ значительно ускорилась. Поэтому в пособии рассмотрим именно эту парадигму разработки программ.

RAD - технология быстрого программирования

RAD (от англ. rapid application development — быстрая разработка приложений) — концепция создания средств разработки программных продуктов, уделяющая особое внимание быстроте и удобству программирования, созданию технологического процесса, позволяющего программисту максимально быстро создавать компьютерные программы. С конца XX века RAD получила широкое распространение и одобрение. Концепцию RAD также часто связывают с концепцией визуального программирования.

Типовым представителем этой концепции в настоящее время является пакет Embarcadero RAD Studio XE. Это полнофункциональный набор средств разработки приложений, который позволяет быстро и наглядно создавать приложения с графическим пользовательским интерфейсом для Windows, .NET, PHP и веб-решений. RAD Studio XE включает в себя Delphi, C++Builder, Delphi Prism и RadPHP; поддерживает компилируемые, управляемые и динамические языки, подключение к базам данных в разнородной среде, полнофункциональные платформы визуальной разработки и возможность использования ПО сторонних разработчиков. Все это дает возможность в 5 раз ускорить разработку приложений для разных платформ Windows, веб-решений и баз данных.

Инженеру – разработчику предлагается несколько вариантов поставки RAD Studio, например RAD Studio XE Professional, RAD Studio XE Enterprise, RAD Studio XE Architect и др. Все они отличаются набором возможностей разработки приложений. Например, вариант RAD Studio XE Professional предназначен для индивидуальных разработчиков и групп,

занимающихся созданием приложений для Windows, .NET с поддержкой интерфейсов для сенсорных экранов со встроенным и локальным хранением баз данных (или без него). В состав RAD Studio входят продукты Delphi, C++Builder, Delphi Prism и RadPHP, предоставляющие все необходимое для быстрого создания Windows-приложений, приложений .NET, PHP и веб-решений.

Редакция RAD Studio XE Enterprise предназначена для индивидуальных разработчиков и групп, занимающихся созданием клиент-серверных, многозвенных и облачных приложений, а также веб-приложений как для Windows, так и для .NET и веб-решений. RAD Studio Enterprise предоставляет удобные средства подключения к различным серверам баз данных и корпоративным источникам данных, расширенные возможности создания UML-моделей и комплексный набор средств создания высококачественных приложений.

Редакция RAD Studio XE Architect предназначена для разработчиков приложений и групп, создающих решения для корпоративных систем. RAD Studio Architect поддерживает все возможности редакции Enterprise и предоставляет дополнительные мощные средства моделирования и проектирования баз данных.

Рассмотрим стандартные методы подхода к RAD – программированию в Delphi.

Интегрированная среда разработки Delphi

Интегрированная Среда Разработки (Integrated Development Environment — IDE) — это среда, в которой есть все необходимое для проектирования, запуска и тестирования приложений и где все нацелено на облегчение процесса создания программ. IDE интегрирует в себе редактор кодов, отладчик, инструментальные панели, редактор изображений, инструментарий баз данных — все, с чем приходится работать.

Основное окно Интегрированной Среды Разработки приведено на рис.21. В верхней части окна IDE мы видим полосу главного меню. Ее состав несколько различается от версии к версии и, кроме того, зависит от варианта Delphi.

Ниже полосы главного меню расположены две инструментальные панели. Левая панель содержит два ряда быстрых кнопок, дублирующих некоторые наиболее часто используемые команды меню. Правая панель содержит палитру компонентов библиотеки визуальных компонентов (Visual Component Library — VCL). В ней содержатся и визуальные (видимые пользователю), и невизуальные компоненты (они явным образом не видны пользователю). Палитра компонентов содержит ряд страниц, закладки которых видны в ее верхней части.

В основном поле окна можно видеть слева окно Инспектора Объектов (Object Inspector), с помощью которого можно задавать свойства компонентов и обработчики событий. Правее показано окно пустой формы, готовой для переноса на нее компонентов. Под ним расположено окно Редактора Кодов. Обычно оно при первом взгляде на экран невидимо, так как его размер равен размеру формы и окно Редактора Кодов полностью перекрывается окном формы.

Рассмотрим несколько основных команд, которые используются при разработке прикладных программ в среде Delphi:

Создание нового проекта приложения начинается с команды "File | New Application". По этой команде открывается новый проект приложения с пустой формой. Сохранить на диске готовый проект или его заготовку можно командой "File | Save Project As" или "File | Save All". Удобно также для сохранения использовать быстрые кнопки — третью или четвертую слева в верхнем ряду. Открыть ранее сохраненный проект можно командой "File | Open" или "File | Open Project" (вторая слева быстрая кнопка в верхнем ряду). Если недавно проводилась работа над проектом, то удобнее воспользоваться командой "File | Reopen" или кнопочкой справа от быстрой кнопки "Open" .

Компиляции и запуск на приложения выполняется командой "Run | Run" (быстрая кнопка с зеленой стрелкой).

Подробное описание всех разделов меню можно найти во встроенной справке Delphi, которая вызывается или из меню "Help", или нажатием клавиши F1.

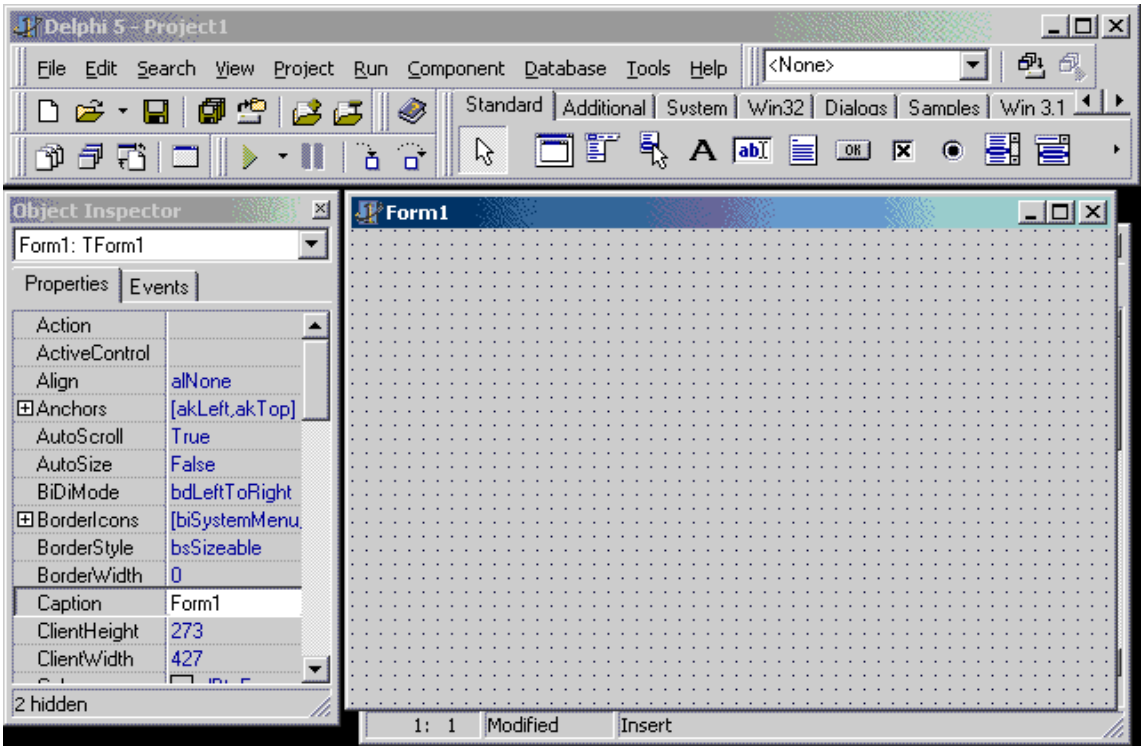


Рис.21. Основное окно среды разработки Delphi

Палитра компонентов — витрина библиотеки визуальных компонентов (Visual Component Library — VCL). Для переноса компонентов из палитры на форму, надо открыть соответствующую страницу библиотеки и указать курсором мыши необходимый компонент. Затем надо сделать щелчок мышью в нужном месте формы и компонент разместится там.

Форма, на которой размещаются компоненты, является основой почти всех приложений Delphi. Она подобна типичному окну Windows и имеет те же свойства, что присущи другим окнам Windows: управляющее меню в верхнем левом углу, полосу заголовка, занимающую верхнюю часть, кнопки развертывания, свертывания и закрытия окна в верхнем правом углу. Форма является контейнером (родителем — Parent) размещенных на ней компонентов. Компоненты также могут размещаться не непосредственно на форме, а в других компонентах — панелях. Тогда родителем по отношению к этим компонентам выступает соответствующая панель.

Одной из наиболее важных частей среды Delphi является окно Редактора Кода, рис.22. Редактор Кода является полноценным программным редактором. Его можно настраивать на различный стиль работы, который вам более привычен. В редакторе применяется выделением цветом синтаксических элементов. Жирным шрифтом выделяются ключевые слова языка Object Pascal (на рис. видно выделение таких слов, как type, class и др.). Синим курсивом выделяются комментарии (например, на рис.22 это комментарий «{ Private declarations }»).

В заголовке окна Редактора Кода отображается имя текущего файла. В верхней части окна можно видеть также закладки или ярлыки, указывающие текущую страницу. Приложения Delphi могут использовать много исходных файлов и закладки помогают переходить от одного из них к другому.

В окно Редактора Кода, как и в другие окна Delphi, встроена контекстная справка. Чтобы получить справку по какому-то слову кода (ключевому слову, написанному имени функции и т.п.) достаточно установить курсор на это слово и нажать клавишу F1. Будет

показана соответствующая тема справки.

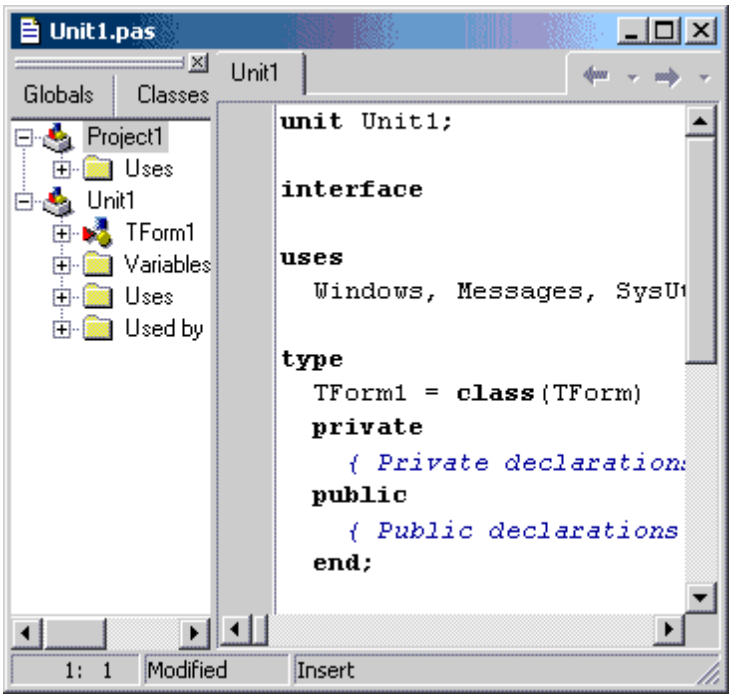


Рис.22 Окно Редактора Кода Delphi

Следующим важнейшим элементом среды разработки является Инспектор Объектов (Object Inspector) — см. левое окно на рис.23. Он обеспечивает простой и удобный интерфейс для изменения свойств объектов Delphi и управления событиями, на которые реагирует объект.

Рассмотрим принципы объектно-ориентированного программирования, на которых основан современный подход к созданию прикладных программ.

В современном представлении программа — это система объектов. Каждый объект характеризуется набором свойств. Свойство — это некоторые данные плюс процедуры их чтения и записи в объект. Эти процедуры называются методами и часто работают невидимо для пользователя. Пусть некоторый объект имеет свойство A и мы пишем в программе оператор A := A * 10 (символы «:=» используются в языке Object Pascal для присваивания переменной, указанной в левой части оператора, значения, равного выражению в правой части). В действительности в этом случае программа вызывает метод чтения значения A, умножает это значение на 10, а затем вызывает метод записи значения в A, и передает в него вычисленную величину A * 10. Эти методы невидимы для пользователя. Но помимо них каждый объект обладает еще рядом методов — функций и процедур, оперирующих со свойствами объекта. Так что более полно объект можно характеризовать как совокупность свойств и методов. Перенеся объект на форму и нажав клавишу F1, мы можем увидеть во встроенной справке Delphi все его свойства (propeties) и методы (methods).

Помимо методов и свойств любой компонент характеризуется набором событий, на которые он может реагировать. Под событиями прежде всего понимаются действия пользователя: щелчок мыши, перемещение курсора, нажатие кнопок мыши или клавиш. Но и сами объекты тоже могут генерировать различные события. В объекте могут предусматриваться обработчики тех или иных событий, воспринимаемых данным компонентом. Фактически к написанию этих обработчиков и сводится программирование приложений. В обработчиках программист описывает, как должны реагировать компоненты на соответствующие события.

Окно Инспектора Объектов (рис.23) имеет две страницы. Страница свойств ("Properties") Инспектора Объектов показывает свойства того объекта, который в данный

момент выделен вами. Щелкнув на окне пустой формы и на странице свойств Инспектора Объектов, можно увидеть свойства формы и изменять эти свойства. Например, изменив свойство Caption (надпись) формы, написав в нем «"Лабораторная работа"», увидим, что эта надпись появится в полосе заголовка нашей формы.

Если щелкнуть на некоторых свойствах, например, на свойстве Color (цвет), то справа от имени свойства откроется окно выпадающего списка. Нажав в нем на кнопку со стрелкой вниз, вы можете увидеть список возможных значений свойства (см. рис.23). Например, сменив значение свойства Color с принятого по умолчанию clBtnFace (цвет поверхности кнопок) на clWindow (цвет окна), поверхность формы изменит свой цвет.

Рядом с некоторыми свойствами мы можем видеть знак плюс (см., например, свойство BorderIcons на рис.23). Это означает, что данное свойство является объектом, который в свою очередь имеет ряд свойств. Например, свойство Font (шрифт). Рядом с ним знак плюс. Щелкнув на этом плюсе или сделав двойной щелчок на свойстве Font, видим, что откроется таблица таких свойств, как Color (цвет), Height (высота), Name (имя шрифта) и др. Среди них свойство Style (стиль), около которого тоже имеется знак плюс. Щелчок на этом плюсе или двойной щелчок на этом свойстве раскроет дополнительный список подсвойств, в котором вы можете, например, установить в true свойство fsBold (жирный). Для смены true на false и обратно в подобных булевых свойствах не обязательно выбирать значение из выпадающего списка. Достаточно сделать двойной щелчок на значения свойства, и оно изменится. После того, как мы просмотрели или изменили подсвойства, можно опять сделать двойной щелчок на головном свойстве или щелчок на знаке минус около него, и список подсвойств свернется.

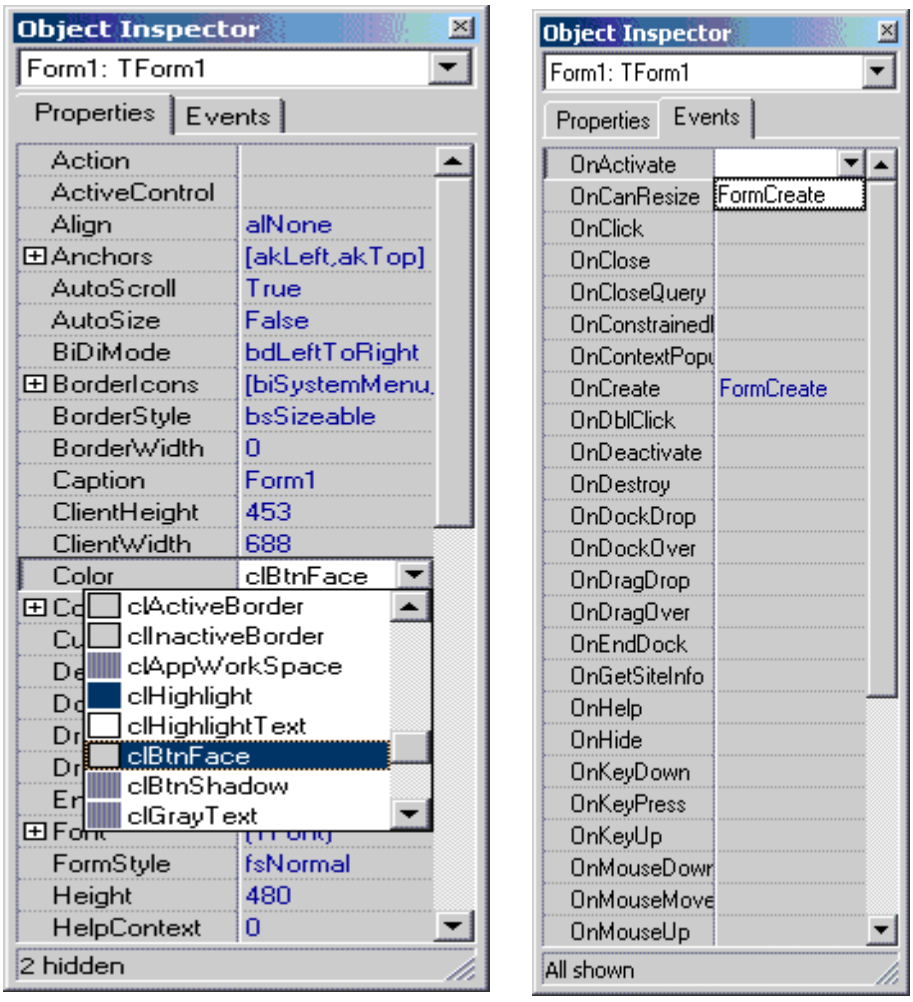


Рис.23. Окно Инспектора объектов

Страница событий ("Events") составляет вторую часть Инспектора Объектов. На ней указаны все события, на которые может реагировать выбранный объект. Например, если надо выполнить какие-то действия в момент создания формы (обычно это различные операции настройки), то мы должны выделить событие OnCreate. Рядом с именем этого события откроется окно с выпадающим списком. Если надо написать новый обработчик, то делаем двойной щелчок на пустом окне списка и попадаем в окно Редактора Кода, со следующим текстом:

```
procedure TForm1.FormCreate(Sender: TObject);
begin

end;
```

Курсор будет расположен в пустой строке между ключевыми словами begin и end. Увиденный код — это заготовка обработчика события, которую автоматически сделал Delphi. Остается только в промежутке между begin и end написать необходимые операторы.

Структура кода модуля

Рассмотрим теперь, как выглядят тексты разрабатываемых прикладных программ. Создадим простое приложение, соответствующее графу вычисления выражения, приведенному ранее на рис.16 в котором при щелчке пользователя на кнопке в окне появлялся бы результат вычисления. Выполняем для этого последовательно следующие шаги.

- Переносим на пустую форму кнопку Button со страницы "Standard" палитры компонент. Для этого выделяем пиктограмму кнопки и затем щелкаем курсором мыши в нужном нам месте формы. На форме появится кнопка, которой Delphi присвоит имя по умолчанию — Button1.
- Аналогичным образом переносим на форму с той же страницы "Standard" палитры компонент метку Label. В этой метке в процессе выполнения приложения будет появляться значение результата при нажатии пользователем кнопки. Delphi присвоит ей имя Label1.
- Далее помещаем на форму три объекта Edit для задания чисел. По-умолчанию это будет Edit1, Edit2 и Edit3.
- Для задания поясняющих надписей применяем компонент StaticText.
- Размещаем компоненты на форме примерно так, как показано на рис. 24.
- Выделяем на форме компонент Button1 — кнопку. Переходим в Инспектор Объектов и изменяем ее свойство Caption (надпись), которое по умолчанию равно Button1 (имя, которое по умолчанию присвоила этому компоненту Delphi) на «"Расчет"».
- Стираем текст в свойстве Caption метки Label1, чтобы он не высвечивался, пока пользователь не нажмет кнопку приложения и аналогично стираем текст в компонентах Edit.
- Теперь нам осталось только написать программу, которая вычисляла математическое выражение и помещала бы результат в свойство Caption метки Label1. Это действие определяется щелчком пользователя на кнопке. При щелчке в кнопке генерируется событие OnClick. Следовательно, обработчик этого события мы и должны написать.
- Выделяем кнопку Button1 на форме, переходим в Инспектор Объектов, открываем в нем страницу событий ("Events"), находим событие кнопки OnClick и делаем двойной щелчок в окне справа от имени этого события. Это стандартный способ задания обработчиков любых событий. После этого мы оказывемся в окне Редактора Кода и видим там текст:

```
procedure TForm1.Button1Click(Sender: TObject);
```

```
begin

end;
```

Заголовок этой процедуры складывается из имени класса нашей формы (TForm1), имени компонента (Button1) и имени события без префикса On (Click).

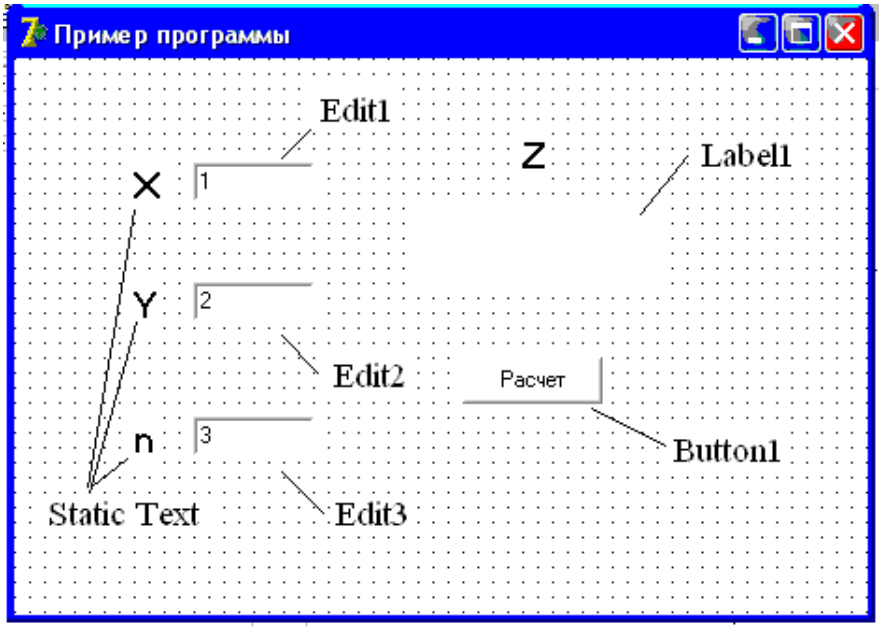


Рис.24. Размещение компонент на форме

- Пишем в обработчике следующую программу:

```
var
 x,y,n,z : integer;   // объявляем переменные целого типа
begin
x:=StrToInt(Edit1.Text);  // берем данные
y:=StrToInt(Edit2.Text);
n:=StrToInt(Edit3.Text);
z:=(x+y)*n;
Label1.Caption:=IntToStr(z); // выводим результат
end;
```

При написании программы, если ввести имя компонента, например Label1, и поставить точку после него, то всплывет подсказка, содержащая список всех свойств и методов метки. Это работает Знаток Кода — Code Insight, который стремится подсказать свойства и методы компонентов, аргументы функций и их типы, конструкции операторов. Мы можем выбрать из списка нужное ключевое слово, нажать клавишу Enter и выбранное слово (свойство, метод) окажется вписанным в текст. Отключить Знаток Кода можно выполнив команду "Tools | Editor Options" и на странице "Code Insight" выключив опцию "Code Completion.

После того, как наше приложение готово, его можно откомпилировать и выполнить командой "Run". После компиляции перед нами появится окно приложения. Введя цифры и нажав кнопку "Расчет", мы увидим результат вычисления. Можно также попробовать различные манипуляции с окном: перемещение его, изменение размеров его рамки курсором мыши, свертывание и развертывание. В заключение закрываем приложение, щелкнув на кнопке в его правом верхнем углу.

Теперь рассмотрим код программы, который создал Delphi. Этот код содержится в двух файлах — головном файле программы с расширением .dpr и в файле модуля с расширением .pas. Головной файл .dpr — один на весь проект, а файлов модулей столько, сколько форм мы ввели в свой проект. Добавление в проект новой формы можно осуществить командой "File | New Form", а удалить форму из проекта можно командой "Project | Remove from Project".

Текст головного файла программы мы, как правило, не видим и не трогаем, полностью доверяя его ведение Delphi. А написание текстов модулей мы делим с Delphi. Delphi создает заготовку модуля, включает в нее описание формы и всех компонентов, которые мы размещаем на форме, обеспечивает связь с библиотекой. А мы пишем в основном обработчики событий компонентов. При необходимости вводим также в текст различные переменные, константы, описания типов, вспомогательные функции и процедуры. Поэтому надо представлять, как выглядит текст модуля и в какие его места что можно добавлять.

Ниже приведен текст модуля описанного выше простого примера, который мы можем увидеть в окне Редактора Кода. Подробные комментарии в этом тексте поясняют, куда и что в этот код вы можете добавлять.

```
unit Unit1;

interface // Открытый интерфейс модуля

{Список подключаемых модулей}
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

{Объявление класса  формы}
type
  TForm1 = class(TForm)
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    Label1: TLabel;
    Button1: TButton;
    StaticText1: TStaticText;
    StaticText2: TStaticText;
    StaticText3: TStaticText;
    StaticText4: TStaticText;
    procedure Button1Click(Sender: TObject);

 private  // Закрытый раздел класса
    { Private declarations }
    {Сюда могут помещаться объявления переменных, функций и процедур, включаемых в
      класс формы, но не доступных для других модулей}
  public   // Открытый раздел  класса
    { Public  declarations }
    {Сюда могут помещаться объявления переменных, функций и процедур, включаемых в
         класс формы и доступных для других модулей}
end;

var
  Form1: TForm1;
```

```
  {Сюда могут помещаться объявления типов, констант, переменных, функций и процедур, к
    которым будет доступ из других модулей, но которые не включаются в класс формы}
implementation // Реализация модуля

{$R *.DFM}
  {Сюда могут помещаться предложения uses, объявления типов, констант, переменных, к
    которым не будет доступа из других модулей. Тут же должны быть реализации всех
    объявленных в разделе interface функций и процедур, а также могут быть реализации
    любых дополнительных, не объявленных ранее функций и процедур.}

procedure TForm1.Button1Click(Sender: TObject);
var
 x,y,n,z : integer;
begin
x:=StrToInt(Edit1.Text);
y:=StrToInt(Edit2.Text);
n:=StrToInt(Edit3.Text);
z:=(x+y)*n;
Label1.Caption:=IntToStr(z);
end;

end.
```

Модуль начинается с ключевого слова unit, после которого пишется имя модуля. Оно совпадает с именем файла, в которым вы сохранили свой модуль. По умолчанию для первого модуля имя равно Unit1, для второго Unit2 — и т.д.

Текст модуля состоит из двух основных разделов: interface — открытый интерфейс модуля, и implementation — реализация модуля. Все, что помещается непосредственно в раздел interface (типы, переменные, константы, функции, процедуры), может быть использовано другими модулями программы. Все, что помещается в раздел implementation — внутреннее дело модуля. Внешние модули не могут видеть типы, переменные, константы, функции и процедуры, размещенные в разделе реализации.

В разделе interface после предложения uses, содержащего список подключаемых библиотечных модулей, мы можем видеть объявление класса формы, подготовленное Delphi. Имя класса этой формы — TForm1. В класс включены те объекты, которые мы разместили на форме — метка Label1, поля ввода Edit1-3, поясняющий текст StaticText1-4 и кнопка Button1. Кроме того, в класс, включено объявление того обработчика щелчка на кнопке, который мы создали — процедуры Button1Click.

В классе предусмотрено также два раздела: private — закрытый раздел класса, и public — открытый раздел класса. То, что мы или Delphi объявим в разделе public, будет доступно для других классов и модулей. То, что объявлено в разделе private, доступно только в пределах данного модуля.

После завершения объявления класса формы мы можем видеть строки

```
var
 Form1: TForm1
```

Это объявляется переменная Form1 класса TForm1, т.е. объявляется наша форма как объекта класса TForm1.

Затем следует раздел реализации implementation, который начинается с директивы компилятора о создании формы. Далее в разделе implementation мы можем видеть реализацию объявленной в классе процедуры Button1Click с операторами, который мы ввели в программу сами. Операторы берут данные строкового типа из полей ввода Edit и,

преобразуя их в целый тип функцией StrToInt, присваивают их переменным x,y и n. Далее производится вычисление z по формуле и, с помощью обратного преобразования типа, результат выводится элементом Label1.

В раздел implementation мы можем сами включать помимо обработчиков событий любые объявления глобальных переменных, констант, типов, и описания функций и процедур. Если мы хочем из данного модуля получить доступ к другому разработанному нами модулю (другой форме), то в раздел implementation вам надо включить оператор uses, в котором указать файл модуля, доступ к которому надо хотите получить. Например, предложение

uses Unit2, Unit3;

обеспечит вам доступ ко всем элементам, объявленным в интерфейсе модулей Unit2 и Unit3.

Программный доступ к свойствам и методам объектов

Рассмотрим теперь, как получить из программы доступ к свойствам и методам объектов. Если нас интересует какое-то свойство объекта, то ссылка на него осуществляется в следующем формате:

<имя объекта>.<имя свойства> (то есть, например, строка: Edit1.Text).

После имени объекта пишется без пробела символ точки, а затем так же без пробела пишется имя свойства

Иногда свойство объекта является в свою очередь объектом. Тогда в обращении к этому свойству указывается через точки вся цепочка предшествующих объектов. Например, метки имеют свойство Font — шрифт, которое в свою очередь является объектом. У этого объекта имеется множество свойств, в частности, свойство Color — цвет шрифта. Чтобы сослаться на цвет шрифта метки Label1, надо написать Label1.Font.Color. Это означает: свойство Color объекта Font, принадлежащего объекту Label1. Например, оператор

Label1.Font.Color:=clRed;

сделает надпись метки Label1 красной.

Аналогичная нотация с точкой используется и для доступа к методам объекта. Например, для метки, как и для большинства других объектов, определен метод Free, который уничтожает метку и освобождает занимаемую ею память. Если мы в какой-то момент решили уничтожить метку Label1 в своем приложении, то пишем оператор:

Label1.Free;

Теперь посмотрим, чем различаются константы, переменные, функции и процедуры, включенные и не включенные в описание класса.

Если в приложении создается только один объект данного класса (в нашем примере — только один объект формы класса TForm1), то различие в основном чисто внешнее. Для процедур, объявленных в классе, в их описании в разделе implementation к имени процедуры должна добавляться ссылка на класс. В нашем примере имя процедуры Button1Click в описании этой процедуры в разделе implementation заменяется на TForm1.Button1Click. Для процедур, объявленных вне класса, такой замены не требуется.

Обращение к переменным и процедурам, описанным внутри и вне класса, из процедур, описанных вне класса, различается. К переменным и процедурам, описанным вне класса, обращение происходит просто по их именам, а к переменным и процедурам, описанным в классе, через имя объекта класса. Например, если надо вне класса описать какую-то процедуру, изменяющую надпись метки Label1, то мы должны обратиться к ее свойству Caption следующим образом: Form1.Label1.Caption. Только через ссылку на объект Form1 внешние по отношению к классу процедуры могут получить доступ ко всему,

объявленному в классе.

Все эти ссылки на объект не требуются в процедурах, объявленных в классе. Поэтому в процедуре TForm1.Button1Click ссылка на объект Label1 не содержит дополнительной ссылки на объект формы.

Если в приложении создается несколько объектов одного класса, например, несколько форм класса TForm1, то проявляются более принципиальные различия между переменными, описанными внутри и вне класса. Переменные вне класса так и остаются в одном экземпляре. А переменные, описанные в классе, тиражируются столько раз, сколько объектов данного класса создано. Т.е. в каждом объекте класса TForm1 будут свои переменные, описанные в классе, и их значения никак не будут связаны друг с другом. Таким образом, в переменную, описанную внутри класса, можно заносить какую-то информацию, индивидуальную для каждого объекта данного класса. А переменная, описанная в модуле вне описания класса, может хранить только одно значение,

Теперь остановимся на вопросе об областях видимости элементов программы — констант, переменных, функций и процедур, т.е. о связи места их объявления в программе и места их использования. Элементы, объявленные в разделе interface модуля, вне описания типа, видимы и доступны внутри данного модуля и из внешних модулей.

Элементы, объявленные в разделе implementation модуля, видимы и доступны внутри данного модуля, но не доступны из внешних модулей.

Элементы, объявленные в классе в разделе private, видимы и доступны только внутри данного модуля. При этом из процедур, объявленных внутри класса, к ним можно обращаться непосредственно по имени, а из других процедур — только со ссылкой на объект данного класса. Если в модуле описано несколько классов, то объекты этих классов взаимно видят элементы, описанные в их разделах private.

Элементы, объявленные в классе в разделе public, видимы и доступны для объектов любых классов и для других модулей. При этом из объектов того же класса к ним можно обращаться непосредственно по имени, а из других объектов и процедур — только со ссылкой на объект данного класса.

В классах, помимо обсуждавшихся ранее, могут быть еще разделы protected — защищенные. Элементы, объявленные в классе в разделе protected, видимы и доступны для любых объектов внутри данного модуля, а также для объектов классов — наследников данного класса в других модулях. Объекты из других модулей, классы которых не являются наследниками данного класса, защищенных элементов не видят.

Элементы, объявленные внутри другой процедуры, являются локальными, т.е. они видимы и доступны только внутри данной процедуры или внутри процедур, вложенных в данную. При этом время жизни переменных, объявленных внутри процедуры, определяется временем выполнения данной процедуры, т.е. эти переменные создаются в момент вызова процедуры и уничтожаются при завершении работы этой процедуры.

Организация библиотеки компонентов

Библиотека визуальных компонентов (Visual Component Library — VCL) Delphi содержит множество предопределенных типов компонентов, из которых пользователь может строить свою прикладную программу. Витрину библиотеки — палитру компонентов, мы видим расположенной справа в полосе инструментальных панелей интегрированной среды разработки Delphi. На этой палитре мы можем выделить курсором мыши нужный компонент и перенести его на форму. Палитра библиотеки в Delphi приведена на рис.25.

Рис. 25 Палитра компонент

Поскольку число страниц в палитре велико и не все закладки видны на экране одновременно, в правой части палитры компонентов имеются две кнопки со стрелками, направленными влево и вправо. Эти кнопки позволяют перемещать отображаемую на экране часть палитры.

Чтобы перенести компонент на форму, надо открыть соответствующую страницу библиотеки и указать курсором мыши необходимый компонент. При этом кнопка-указатель, размещенная в левой части палитры компонентов, приобретет вид не нажатой кнопки. Это значит, что мы находимся в состоянии, когда можно поместить компонент на форму. Поместить выбранный компонент на форму очень просто — надо сделать щелчок мышью в нужном месте формы.

Есть и другой способ поместить компонент на форму — достаточно сделать двойной щелчок на пиктограмме компонента, и он автоматически разместится в центре вашей формы. Если мы выбрали компонент, а затем изменили намерение размещать его, достаточно нажать кнопку указателя. Это прервет процесс размещения компонента и программа вернется в нормальный режим, в котором можно выбирать другой компонент или выполнять какую-то команду.

Имена компонентов, соответствующих той или иной пиктограмме, можно узнать из ярлыка, появляющегося, если задержать над этой пиктограммой курсор мыши. Если выбрать в палитре компонент и нажать клавишу F1, то будет показана справка по типу данного компонента.

Палитра имеет ряд страниц, на которых скомпонованы пиктограммы всех компонентов, предопределенных в Delphi. Перечислим некоторые страницы палитры Delphi:

- "Standard" - Стандартная, содержащая наиболее часто используемые компоненты;
- "Additional" - Дополнительная, являющаяся дополнением стандартной ;
- "Win32" - 32-битные компоненты в стиле Windows 95/98 и NT;
- "System" - Системная, содержащая такие компоненты, как таймеры, плееры и ряд других;
- "Data Access" - Доступ к данным через Borland Database Engine (BDE);
- "Data Controls" - Управление данными;
- “DB Express” - Связь с данными с помощью dbExpress;
- "ADO" - Связь с базами данных через Active Data Objects (ADO);
- "InterBase" - Прямая связь с InterBase, минуя BDE и ADO;
- "Midas" - Построение приложений баз данных с параллельными потоками;
- "Internet Express" - Построение приложений InternetExpress — сервера Web и клиента баз данных с параллельными потоками;
- "Internet" - Интернет, компоненты для приложений, работающих с Интернет;
- "Decision Cube" - Многомерный анализ данных;
- "Qreport" (или “Rave”) - Быстрая подготовка отчетов;
- "Dialogs" - Диалоги, системные диалоги типа «Открыть файл» и др.
- "Win 3.1" - Windows 3.x, компоненты в стиле Windows 3.x;
- "Servers" - Оболочки VCL для распространенных серверов COM.

Имеются еще страницы, содержащие примеры: "ActiveX" - примеры активных элементов ActiveX , "Samples" - образцы: различные интересные, но не до конца документированные компоненты, а также “Indy” - Компоненты клиентских приложений Internet Direct (Indy), дающих доступ к различным протоколам Интернет

Элементы языка Object Pascal

Для того, чтобы создавать эффективные программы в Delphi, основываясь на знании языка Pascal, рассмотрим некоторые элементы более продвинутого языка Object Pascal.

К элементам относятся:

- зарезервированные слова;
- идентификаторы;
- типы;
- константы;
- переменные;
- метки;
- функции и подпрограммы;
- комментарии.

Зарезервированные слова это английские слова, указывающие компилятору на необходимость выполнения определенных действий. Зарезервированные слова не могут использоваться в программе ни для каких иных целей, кроме тех, для которых они предназначены.

Зарезервированные слова представлены в таблице 2.

Таблица 2

and	except	label	resourcestring
array	exports	library	set
as	file	mod	shl
asm	finalization	nil	shr
begin	finally	not	string
case	for	object	then
class	function	of	threadvar
const	goto	or, xor	to
constructor	if	out	try
destructor	implementation	packed	type
dispinterface	in	procedure	unit
div	inherited	program	until
do	initialization	property	uses
downto	inline	raise	var
else	interface	record	while
end	is	repeat	with

Типы

Типы - это специальные конструкции языка, которые рассматриваются компилятором как образцы для создания других элементов программы, таких как переменные, константы и функции, рис.26.. К основным типам данных языка Delphi относятся:

- целые числа (integer);
- дробные числа (real);
- символы (char);
- строки (string);
- логический тип (boolean).

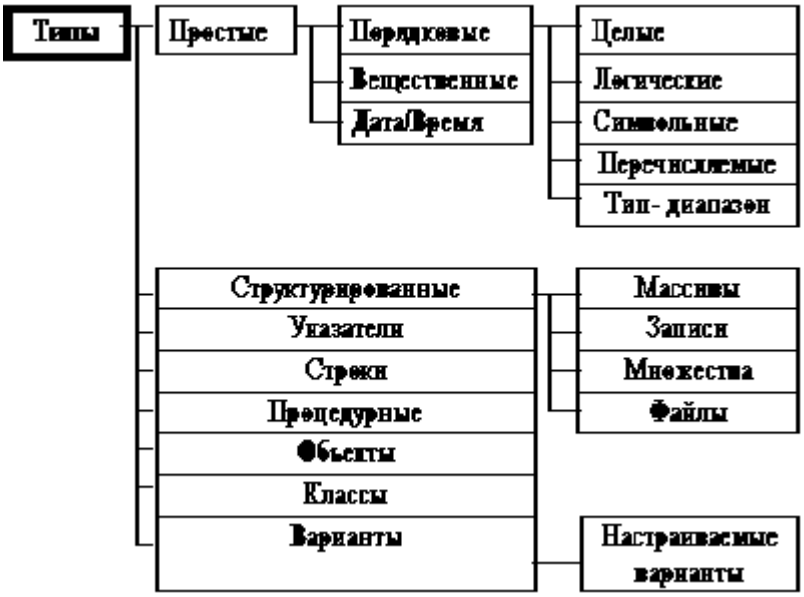


Рис.26. Основные типы данных языка Delphi

Целые числа и числа с плавающей точкой могут быть представлены в различных форматах, таблица 3.

Таблица 3

Тип	Диапазон возможных значений	Размер памяти для хранения данных
Integer	-2147483648..2147483647	4 байта (32 бита)
Cardinal	0..4294967295	4 байта (32 бита)
Shortint	-128..127	1 байт (8 бит)
Smallint	-32768..32767	2 байта (16 бит)
Longint	-2147483648..2147483647	4 байта (32 бита)
Int64	$-2^{63}..2^{63}-1$	8 байт (64 бита)
Byte	0..255	1 байт (8 бит)
Word	0..65535	2 байта (16 бит)
Longword	0..4294967295	4 байта (32 бита)

Для вещественных типов так же предусмотрены стандартные типы, таблица 4.

Таблица 4

Тип	Диапазон возможных значений	Максимальное количество цифр в числе	Размер в байтах
Real48	$2.9 \times 10^{-39} .. 1.7 \times 10^{38}$	11-12	6
Real	$5.0 \times 10^{-324} .. 1.7 \times 10^{308}$	15-16	8
Single	$1.5 \times 10^{-45} .. 1.7 \times 10^{38}$	7-8	4
Double	$5.0 \times 10^{-324} .. 1.7 \times 10^{308}$	15-16	8
Extended	$3.6 \times 10^{-4951} .. 1.1 \times 10^{4932}$	19-20	10
Comp	$-2^{63}+1..2^{63}$	19-20	8
Currency	-922337203685477.5808.. 922337203685477.5807	19-20	8

Примеры объявлений и использования переменных:

```
var
x: integer;
y: double;
...
x := 5;
```

```
y := 5.25; // обратите внимание, что дробная часть отделяется точкой
y := x + y; // так делать можно
x := x + y; // а так - нельзя, поскольку результатом должно быть целое
```

Булевы, или логические типы данных представлены в Delphi типами Boolean, ByteBool, WordBool и LongBool. Все они отличаются только размером памяти, выделяемым для хранения значения, причем значений может быть только 2: - false (ложь) и true (истина):

```
var x, y: Boolean;
...
x := true;
y := false;
```

Для символьных типов данных, то в Delphi предусмотрено 2 их типа - ANSIChar и WideChar. Первый является однобайтовым и может хранить в себе 1 символ из множества символов ANSI, которых насчитывается 256. Второй тип является 2-байтовым и предназначен для хранения 2-байтовых же символов Unicode. Как и в других случаях, Delphi имеет синоним для символьных типов - Char, который на сегодня является аналогом ANSIChar. Что касается присвоения значений, то обычные символы (буквы и цифры) присваивают переменным символьного типа как есть, заключая их в одиночные кавычки. А специальные символы, например, возврат каретки (Enter) назначают при помощи их номера в таблице ANSI, и выделяют знаком решетки:

```
var x, y: Char; // x и y получают тип ANSIChar
...
x := a; // обычные символы
y := #13; // возврат каретки в таблице ANSI имеет номер 13
```

Наконец, еще одним простым типом данных являются строки. Строковые типы данных отличаются от символьных тем, что могут хранить не единичный символ, а множество символов. В Delphi имеется 3 типа строк: ShortString, AnsiString и WideString. Первый тип строк - ShortString - достался в наследство от языка Pascal. Такие строки могут иметь не более 255 символов, и занимают от 2 до 256 байт памяти. Что касается современных строковых типов - AnsiString и WideString, то они могут иметь практически неограниченную длину (AnsiString - до 2 млрд. символов, WideString - до 1 млрд.) и занимать, соответственно, от 4 байт до 2 Гигабайт памяти. При этом по аналогии с символьными типами, тип AnsiString предназначен для хранения обычных строк, а WideString - для строк в формате Unicode. Ну и еще один тип строк - String является синонимом для типа AnsiString:

```
var str1: ShortString; // короткая строка
var str2: AnsiString; // длинная строка
var str3: String; // тоже длинная строка
...
str1 := ‘Начало’; // Строковые значения заключаются в одинарные кавычки
str2 := ‘Конец’;
str3 := str1 + str2; // получим длинную строку, содержащую Начало.Конец.
```

В целом, несмотря на кажущееся разнообразие типов данных, на практике чаще всего ограничиваются всего лишь 5-6 основными типами. Это: Integer, Double, Boolean, Char, String, и иногда - еще и Currency.

Операции и их типы

В любом языке программирования имеются знаки операций. Кроме того, некоторые ключевые слова, например такие, как div или mod также обозначают операции. Все операции в Object Pascal можно разделить на следующие типы: логические, арифметические, логические, операции присвоения и отношения, а так же специальные операции. Для их обозначения используются математические символы или ключевые слова. Участвующие в операциях значения (переменные) называются операндами. При этом та или иная операция может работать с операндами определенного типа. А результатом может быть данные как такого же типа, та и другого (например, для того же сравнения).

Арифметические операции

Арифметические операции являются наиболее распространенными, их перечень, а так же с типы исходных и результирующих данных представлены в таблице 5.

Таблица 5.

Операция	Название	Тип входной	Тип выходной
+	Сложение	integer, real	integer, real
-	Вычитание	integer, real	integer, real
*	Умножение	integer, real	integer, real
/	Деление	integer, real	integer, real
mod	Остаток от деления	integer, real	Real
div	Деление нацело	integer, real	integer, real
-	Унарный минус	integer, real	integer, real
+	Унарный плюс	integer, real	integer, real

Можно отметить, что практически для всех арифметических операций результатом будут данные того же типа, что и операнды. Единственное исключение - это операция деления, результатом которой всегда будет вещественное число.

Логические операции

Другой распространенный тип операций - логические. В Object Pascal имеются 4 типа логических операций: не, и, или, исключающее или (таблица 6).

Логическому сравнению подлежат не только булевские значения, но и любые другие выражения, которые могут быть к ним приведены. Например, выражение "3=4" может быть использовано в качестве логически сравниваемой единицы, поскольку результатом его оценки будет булево значение ложь (false).

Те же самые знаки операций, что используются в логических операциях, задействованы и в другом типе операций - побитовых. Побитовые операции выполняются над числами, представленными в двоичном виде (т.е. только нули и единицы). Однако сами операнды могут быть десятичными, шестнадцатеричными, или восьмеричными целыми числами. Например, десятичное число 5 представляется как двоичное 101, десятичное 6 - как 110, а шестнадцатеричное F3 - как двоичное 11110011.

Хотя побитовые операции выполняются над двоичными данными, возвращаемые значения являются стандартными числами. Список всех побитовых операций аналогичен, но дополнительно к имеющимся имеются операции сдвига. Результат действия побитовых операций был ранее рассмотрен в разделе двоичной арифметики.

Таблица 6

Операция	Название	Описание
not	Логическое отрицание (НЕ)	Возвращает false, если выражение может быть приведено к истине, в противном случае возвращает true. Для побитового возвращает

		инверсное число.
and	Логическое (И)	Возвращает true, когда оба выражения истинны. В противном случае возвращает false. Для побитового возвращает число побитового сравнения «И».
or	Логическое (ИЛИ)	Возвращает true, когда хотя бы одно из выражений истинно. В противном случае возвращает false. Для побитового возвращает число побитового сравнения «ИЛИ».
xor	Логическое (исключающее ИЛИ)	Возвращает true, когда только одно из выражений истинно. В противном случае возвращает false. Для побитового возвращает число побитового сравнения «Искл. ИЛИ».
shl	Побитовый сдвиг влево	Сдвигает первый операнд влево на число разрядов, заданных вторым операндом. Освобождающиеся правые биты заполняются нулями
shr	Побитовый сдвиг вправо	Сдвигает первый операнд вправо на число разрядов, заданных вторым операндом. Освобождающиеся левые биты отбрасываются

Операции сравнения

Следующий тип операций - операции сравнения. Эти операции всегда требуют наличия двух операндов: они сравнивают значения левого и правого операндов, и возвращают результат сравнения в виде логического значения, которое может принимать значение false или true (ложь или истина). Все имеющиеся в Object Pascal операции сравнения приведены в таблице 7.

Таблица 7.

Операция	Название	Описание
=	Равно	Возвращает истину (true), когда левый правый операнды равны.
<>	Не равно	Возвращает истину, когда левый и правый операнды не равны.
>	Больше	Возвращает истину, когда левый операнд больше правого.
<	Меньше	Возвращает истину, когда левый операнд больше правого.
>=	Больше или равно	Возвращает истину, когда левый операнд больше правого или равен ему.
<=	Меньше или равно	Возвращает истину, когда левый операнд меньше правого или равен ему.

Операции сравнения могут работать с различными типами данных, включая строки, при этом для строк используется стандартный подход: символы последовательно сравниваются по своим ASCII-кодам, или Unicode-кодам, если используется строковая переменная в формате WideChar.

К операциям сравнения так же можно отнести операцию членства - in. При этом в качестве левого операнда может выступать данные ординарного типа (Byte или Char), а в качестве левого - подмножество значений, совместимых с данным типом. Например, если имеется переменная x типа Char, то можно проверить, является ли ее значение частью некоего набора символов:

```
var
  x: Char;
  z: Boolean;
...
x := 'b';
z := x in [a..d];
```

В данном случае в качестве результата (z) мы получим истину, поскольку символ b является членом указанного множества [a..d], в которое входят символы a, b, c и d. Наконец, в Object Pascal имеется еще 2 операции - as и is. Они служат для приведения типа и проверки типа, соответственно. Например, если мы хотим проверить, является ли некая переменная "x" целым, то можно написать такое выражение:

```
b := x is Integer; // b получит значение true, если x - целое
```

Операция as может использоваться для приведения данных одного типа к другому, причем, преимущественно, при работе с объектами:

```
ObjA as ObjB;
```

При использовании операции as следует быть осторожным, поскольку приведение одного типа к другому возможно далеко не всегда.

Выражения и приоритет операций

Как нам уже известно, выражения - это, прежде всего, набор данных, переменных и операторов. Можно сказать, что выражения состоят из значений и операций над ними, например:

```
a := b + c;
d := e * f;
g := a - d div 2;
```

Так же, как и в обычной математике, при составлении выражений в Object Pascal, следует учитывать приоритет выполнения операций. Например, операции умножения или деления должны выполняться раньше, чем сложение и вычитание. Так, в 3-й строке из приведенных выше примеров выражений, согласно математическим правилам, сначала выполняется операция деления нацело (d div 2), затем результат этой операции вычитается из a, и итоговое значение присваивается переменной g.

Прежде всего, приоритет выполнения определяется принадлежностью конкретной операции к тому или иному типу. Так, первыми выполняются унарные операции, затем - операции умножения (включая деление и побитовые), далее обрабатываются сложение и вычитание и, наконец, в последнюю очередь - операции отношения.

В тех случаях, когда порядок следования операций, задаваемый их приоритетом, следует изменить, применяют круглые скобки. Например, если мы имеем выражение:

```
g := a - d div 2;
```

Но нам требуется сначала выполнить вычитание, то достаточно заключить участвующие в операции вычитания операнды в круглые скобки:

```
g := (a - d) div 2;
```

Массивы

Массивы в Object Pascal во многом схожи с аналогичными типами данных в других языках программирования. Отличительная особенность массивов заключается в том, что все их компоненты по сути данные одного типа. Эти компоненты можно легко упорядочить и обеспечить доступ к любому из них простым указанием его порядкового номера. Описание

типа массива (объем не более 2 Гбайт) задается следующим образом:

<имя типа> = array [<сп.инд.типов>] of <тип>;

Определить переменную как массив можно и непосредственно при описании этой переменной, без предварительного описания типа массива, например:

```
var
a,b : array [1..10] of Real;
```

Динамические массивы

В Delphi впервые введены так называемые динамические массивы. При объявлении таких массивов в программе не следует указывать границы индексов:

```
var
A: array of Integer;
B: array of array of Char;
C: array of array of array of Real;
```

В этом примере динамический массив A имеет одно измерение, массив B - два и массив C - три измерения. Распределение памяти и указание границ индексов по каждому измерению динамических массивов осуществляется в ходе выполнения программы путем инициации массива с помощью функции SetLength. В ходе выполнения такого оператора:

```
SetLength(A,3);
```

одномерный динамический массив A будет инициирован, т. е. получит память, достаточную для размещения трех целочисленных значений. Нижняя граница индексов по любому измерению динамического массива всегда равна 0, поэтому верхней границей индексов для A станет 2.

Объявление одномерного массива:

ИмяМассива: array [НижнийИндекс...ВерхнийИндекс] of ТипЭлементов;

Объявление двумерного массива:

ИмяМассива: array [НижнийИндекс1..ВерхнийИндекс1,
НижнийИндекс2..ВерхнийИкдекс2] of ТипЭлементов;

Записи

Запись - это структура данных, состоящая из фиксированного количества компонентов, называемых полями записи. В отличие от массива компоненты (поля) записи могут быть различного типа. Чтобы можно было ссылаться на тот или иной компонент записи, поля именуются.

Структура объявления типа записи:

```
<имя типа> = record
        <список полей>
    end;
```

Запись можно объявлять либо как сразу переменную:

```
Var
     Запись  = record
       Поле1:Тип1;
       Поле2: Тип2;
       ПолеJ: ТипJ;
     end;
```

Либо сначала объявить тип-запись, затем – переменную - запись:

```
type
```

```
ТипЗапись = record
    Поле1: Тип1;
    Поле 2:Тип2;
    ПолеК: ТипК;
end;

var   Запись: ТипЗапись;
```

Множества

Множества - это наборы однотипных логически связанных друг с другом объектов. Характер связей между объектами подразумевается программистом и никак не контролируется Object Pascal. Количество элементов, входящих в множество, может меняться в пределах от 0 до 255 (множество, не содержащее элементов, называется пустым). Именно непостоянством количества своих элементов множества отличаются от массивов и записей.

Два множества считаются эквивалентными тогда и только тогда, когда все их элементы одинаковы, причем порядок следования элементов в множестве безразличен. Если все элементы одного множества входят также и в другое, говорят о включении первого множества во второе. Пустое множество включается в любое другое.

Описание типа множества имеет вид:

<имя типа> = set of <базовый тип>;

Указатели

Оперативная память ОЗУ представляет собой совокупность - байтов, каждый из которых имеет собственный номер. Эти номера называются адресами, они позволяют обращаться к любому байту памяти. Object Pascal предоставляет в распоряжение программиста гибкое средство управления динамической памятью - так называемые указатели.

Указатель - это переменная, которая в качестве своего значения содержит адрес байта памяти. С помощью указателей можно размещать в динамической памяти любой из известных в Object Pascal типов данных. Лишь некоторые из них (Byte, Char, ShortInt, Boolean) занимают во внутреннем представлении один байт, остальные - несколько смежных. Поэтому на самом деле указатель адресует лишь первый байт данных.

Как правило, указатель связывается с некоторым типом данных. Такие указатели называются типизированными.

Операторы Object Pascal

Безусловный переход goto.

Безусловный переход goto, которая в Pascal используется совместно с метками, декларируемыми при помощи ключевого слова label.

Составной оператор.

В тех случаях, когда по правилам языка можно использовать всего лишь одно выражение или оператор, а нужно несколько, используют составные операторы. Составной оператор представляет собой группу из произвольного числа любых инструкций, ограниченных ключевыми словами begin и end:

```
begin
  <инструкция 1>;
  <инструкция 2>;
  ...
  <инструкция N>;
end;
```

Сколько бы ни входило инструкций в такой блок, для компилятора он воспринимается как единое целое и может располагаться в любом месте программы, где допускается наличие хотя бы одного оператора. Составные операторы могут вкладываться один в другой, при этом глубина таких вложений в Object Pascal не ограничена.

Условный оператор if

Инструкцией для управления выполнением программы является условный оператор if. Он отвечает за ветвление, т.е. выполнение (или невыполнение) того или иного варианта кода в зависимости от условий. Оператор if используется совместно с ключевым словом then, а в том случае, когда предусмотрен альтернативный вариант выполнения - еще и с else. Синтаксис инструкции следующий:

if <условие> then <код> [else <альтернативный код>]

В качестве условия может быть использовано любое выражение, которое может быть приведено к булевскому значению, т.е. к false или true. Как правило, это бывают операции сравнения, например:

```
if (a > 5) then b := 10;
if( x <> 0) then y := 1 else y :=2;
```

В тех случаях, когда для того или иного варианта кода требуется использовать более одного выражения, используют составной оператор:

```
if( a > 5) then
  begin
    b := 10;
    c := 20;
  end;
```

Между then и else допустима лишь 1 инструкция. Поэтому в таком случае точку с запятой следует разместить уже после оператора, следующего за else:

```
If( a > 5) then
  begin
    b := 10;
    c := 20;
  end
else
  begin
    b := 20;
    c := 15;
  end;
```

В тех случаях, когда требуется предусмотреть 3 или более вариантов исполнения, используют вложение операторов if друг в друга. Например, если требуется выполнить один вариант когда, когда некая переменная x меньше нуля, другой - если x равна 0, и третий - если x больше нуля, то синтаксис операторов может быть следующим:

```
if x < 0 then
 <вариант для x<0>
else
   if x = 0 then
    <вариант для x=0>
      else
         <вариант для x>0>;
```

В данном случае использован вложенный оператор if, который выполняется в случае, когда переменная x не меньше 0. Он проверяет, не является ли значением x число 0, и если нет, то, учитывая, что x явно не меньше, чем 0 (это условие к моменту выполнения вложенного оператора if уже проверено внешним, т.е. первым в данном выражении оператором if), значит значение x больше 0.

Оператор выбора case

Условный оператор удобен в тех случаях, когда необходимо проверить 1-2-3 варианта. При большем числе получается слишком громоздкая и неудобная для восприятия конструкция из множества вложенных инструкций.

В таких случаях на помощь приходит семафор - оператор множественного выбора case. Он состоит из выражения, являющегося селектором, списка вариантов, представленного константами или значениями, и необязательной части else. Таким образом, формат оператора case таков:

```
case [выражение-селектор] of
  <значение 1>: <код для значения 1>;
  <значение 2>: <код для значения 2>;
  ...
  <значение N>: <код для значения N>;
  [else <код для непредусмотренных явно значений>;]
end
```

Единственным ограничением семафора, в сравнении с условным оператором, является то, что в качестве селектора могут выступать лишь данные порядкового типа - целое число или символ. Например, код с 5 вложенными условными операторами, при помощи case можно оформить так:

```
case x of
  1: <код для значения 1>;
  2: <код для значения 2>;
  3: <код для значения 3>;
  4: <код для значения 4> ;
  5: <код для значения 5>;
end;
```

Здесь подразумевается, что типом переменной x является целое число, поскольку тип значений, коими в данном случае являются целые числа, должен соответствовать типу селектора.

В качестве констант выбора могут выступать не только единичные значения, но и их список, разделенный запятыми, или же диапазоны, определенные границами из 2 констант, разделенных двумя точками. В таком случае мы можем объединить логически связанные значения в группы, для которых следует выполнить один и тот же код.

Оператор цикла for

Для написания программы, помимо операторов условия, часто требуются операторы цикла. Прежде всего, это оператор цикла с параметром - for. Такой тип цикла обычно применяют в тех случаях, когда количество возможных повторов известно заранее. Он имеет 2 варианта написания: один - для цикла с приращением, и другой - для цикла с уменьшением:

```
for <параметр> := <выражение 1> to <выражение 2> do <тело цикла>;

for <параметр> := <выражение 1> downto <выражение 2> do <тело цикла>;
```

В первом случае (с использованием цикла for-to) при каждом проходе цикла,

называемом итерацией, значение параметра увеличивается на 1, а во втором (for-downto) - уменьшается на 1. При этом в качестве начального значения используется "выражение 1", а в качестве конечного - "выражение 2". Разумеется, если для цикла to значение первого выражения изначально будет больше значения второго, или наоборот, меньше (для цикла downto), то цикл не будет выполнен ни разу. Практическое применение циклов крайне разнообразно. Если привести наиболее общий пример из программирования, то цикл - идеальный способ заполнения массива. Например, если требуется организовать цикл для заполнения массива из 10 числовых значений последовательно возрастающими числами, то можно записать:

```
for i := 0 to 9 do MArray[i]=i;
```

В данном случае элементам массива MArray присваиваются значения от 0 до 9.

Рассмотрим цикл for с отрицательным приращением на примере вычисления математического факториала (последовательное произведение всех целых чисел от 1 до самого числа). Для этого нам понадобится следующий цикл:

```
var num, rez: integer;
...
rez := 1;
for num := num downto 1 do rez := rez * num;
```

Здесь нам потребовалось: определение переменной rez, в которой будет храниться вычисляемое значение, далее ей присвоено значение 1. В качестве числа, для которого вычисляется факториал, выступает переменная num, она же используется для самого цикла в качестве счетчика. Поскольку нам надо будет прекратить выполнение цикла, после того, как счетчик (num) достигнет значения 1, то именно это значение и указано в качестве конечного условия.

В итоге, если переменной num присвоить значение 5, то после прохождения цикла переменная rez получит значение 120. Хотя в результате работы такого цикла получится выполнение как бы наоборот (т.е. не вместо 1*2*3*4*5, на самом деле выполняется 5*4*3*2*1), это никак не помешает получить верный результат.

В качестве тела цикла, как и в случае с уже рассмотренными операторами, может использоваться составной оператор. Кроме того, циклы могут быть вложены один в другой, при этом важно лишь следить за тем, где заканчивается вложенный цикл и начинается внешний. Для этого полезно следить за правильным оформлением программы, в частности, использовать отступы, в качестве которых можно использовать либо знак табуляции, либо пробелы, в последнем случае их желательно ставить не менее 2.

```
// Вложенные циклы и форматирование кода
for x := 5 to 10 do begin
  z := x;
  for y := 10 to 20 do begin
    z := z + x * y;
    writeln(z);
  end; // конец вложенного цикла
  writeln(x);
end; // конец внешнего цикла
```

При использовании циклов с параметром важно помнить, что изменение значения параметра в теле цикла недопустимо.

Операторы циклов while и repeat

Помимо классического цикла с параметром, в Object Pascal предусмотрено еще 2 вида

циклов - с предусловием и с постусловием. В качестве цикла с предусловием выступает оператор while. Он имеет следующий синтаксис:

while <Условие> do <тело цикла>

Этот цикл будет выполняться до тех пор, пока верно выражение, указанное в качестве условия. Соответственно, если значение выражение будет изначально ложным, то цикл не будет выполнен ни разу, например:

```
while false do ;
```

В то же время, при помощи оператора while удобно делать бесконечный цикл. В бесконечных циклах весь контроль возлагается на операторы, помещаемые в тело цикла:

```
while true do begin
  <инструкции>
  end;
```

Очевидно, что цикл с таким условием будет выполняться вечно. Таким образом, в теле цикла следует предусмотреть возможность его прерывания иным способом.

В отличие от while, цикл с постусловием, задаваемый при помощи оператора repeat, всегда выполняется хотя бы 1 раз, поскольку проверку на соответствие условию он проходит после того, как будет выполнено его тело:

repeat <тело цикла> until <условие>

Важно так же отметить, что в цикле с постусловием ключевые слова repeat и until образуют как бы составной оператор. Иначе говоря, если в цикле while при необходимости использовать более одной инструкции следует использовать составной оператор, то для цикла repeat этого не требуется:

```
repeat
  x := x + 1;
  y := x * 2;
until y < 1000;
```

При использовании циклов while и repeat важно помнить, что хотя бы одна из инструкций, выполняемых в теле цикла, должна влиять на значение его условия, в противном случае мы рискуем получить зацикливание.

Операторы break и continue

Чтобы устранить зацикливание программы, а так же сделать сами циклы более гибкими, используют специальные операторы - break и continue. Оператор break позволяет завершить цикл досрочно, а оператор continue - выполнить только часть операторов в теле цикла, перейдя к его следующей итерации.

Таким образом, досрочный выход из цикла, определяемый при помощи оператора break, происходит в месте вызова этого оператора, и управление будет передано первому оператору, находящемуся после цикла. Например, в уже рассмотренном нами бесконечном цикле, можно задать условие его прерывания так:

```
while true do begin
  a := b * c;
  if( a > 1000) then break;
  b := b + 1; // в случае если a > 1000, эта строка выполнена не будет
end;
```

Еще одним полезным применением досрочного выхода является его использования в качестве дополнительного параметра. Например, если нам нужен цикл, который должен прерваться по 1 из 2 условий, то для второй проверки мы можем использовать условный

оператор if совместно с break:

```
repeat
 i := i + 1;
 if( i > 100) then break;
 y := y - 1;
until y < 50;
```

Здесь мы определили цикл, который будет завершен либо после того, как значение переменной y достигнет 50 (что задано в самом условии цикла), либо если значение переменно x превысит указанное в условии if значение 100 - здесь как раз будет задействован оператор break.

Иногда бывает необходимо перейти к следующему шагу цикла досрочно, пропустив часть операторов. Для этих целей используют оператор continue. В отличие от break, этот оператор не завершает цикл, а заставляет программу досрочно перейти к новой проверке условия цикла. Рассмотрим эту ситуацию на примере. Допустим, что нам надо получить список чисел, на которые число 100 делится без остатка:

```
for i := 1 to 100 do begin
 if (100 mod i <> 0) then continue;
 writeln(i);
end;
```

Для этого мы определили цикл, в котором при каждой итерации делитель, в качестве которого выступает счетчик цикла, увеличивается на единицу. В самом теле цикла использован условный оператор, в котором условием выступает выражение, в котором производится операция "остаток от деления" и ее результат сравнивается с 0. Таким образом, если дано условие истинно (т.е. если остаток равен нулю), то интерпретатор переходит к следующему оператору в теле цикла, который выводит нужное нам число, в противном случае выполняется оператор continue и начинается новый шаг цикла.

Использование процедур и функций

При разработке больших программ приходится разбивать программу на несколько частей и, если в программе наблюдается множество повторяющихся действий, то их желательно оформить либо в виде функции, либо подпрограммы. Отличие между ними в оформлении и передаче параметров. Например, оформление функции:

```
function ИмяФункции(var Параметр1: Тип 1; var ПараметрJ: ТипJ ) : Тип;
const
{ описание констант }
var
{ описание переменных }
begin
{ инструкции функции }
Result:=Значение;
end;
```

Слово var ставится перед именем параметра в том случае, если параметр используется для возврата значения из функции в вызвавшую ее программу.

Объявление процедуры:

```
procedure ИмяПроцедуры(var Параметр1: Тип1; var ПараметрJ: ТипJ ) ;
const
```

```
{ описание констант }
var
{ описание переменных }
begin
{ инструкции процедуры }
end;
```

Аналогично: слово var ставится перед именем параметра в том случае, если параметр используется для возврата значения из процедуры в вызвавшую ее программу.

Работа с файлами

В Delphi под файлом понимается информация структурированного типа, размещенная либо на внешнем/внутреннем носителе, либо это логическое устройство.

В языке поддерживается два способа работы с файлами – через файловую переменную, и – напрямую через имя – дескриптор файла.

Файловый тип можно объявить в следующих вариантах:

<имя>= File of <тип> - для типизированных файлов;
<имя>= TextFile - для текстовых файлов;
<имя>= File - для нетипизированных файлов.

Файлы и логические устройства должны быть открыты, для чего производится связывание специальной файловой переменной с реальным файлом с помощью оператора:

AssignFile(f,'имя файла.тип'), где f - файловая переменная.

- Для чтения файл инициируется командой Reset(f[,размер]).
- Новый файл инициируется командой Rewrite(f[,размер]).
- Добавление информации требует использования команды Append(f) для открытия файла.
- После работы с файлом его необходимо закрыть командой CloseFile(f).
- Стирание файла производится командой Erase(f).
- Проверку существования файла можно осуществлять функцией FileExist(имя файла), дающей логический результат.
- Проверка конца файла производится функцией Eof(f), а проверка правильности последней операции ввода-вывода функцией IOResult.

Текстовый файл в Delphi представляет собой совокупность строк переменной длины. Доступ к каждой строке последовательный, а информация из файла читается последовательно. В конце каждой строки имеется признак конца строки Eoln(f). Читать и записывать информацию в текстовом файле можно командами (процедурами) Read(f,<список>), Readln(f,<список>), Write(f,<список>), Writeln(f,<список>).

При каждой операции указатель положения смещается и можно делать проверки положения указателя функциями SeekEof(f) и SeekEoln(f).

Процедура Flush(f) предназначена для сброса буфера ОЗУ на жесткий диск.

Для типизированных файлов считается, что длина каждого компонента файла постоянна и равна длине используемого типа. Например, файл, определенный как:

```
T: File of integer;
```

имеет длину компонента, равную 2 байтам.

Номер первого компонента – 0. Чтение и запись информации аналогична, однако можно

сразу перейти на нужный компонент командой Seek(f, <номер компоненты>).

Также можно получить номер текущей компоненты командой FilePoz(f) и общий размер файла командой FileSize(f).

Можно также удалить часть файла, начиная с текущей позиции, командой Truncate(f).

Для нетипизированных файлов считается, что тип компонент может быть любым. Исходя из этого, программист сам должен определить размер читаемой/записываемой компоненты, а само действие производится командами BlockRead(f,buf,N,[Result]) и BlockWrite(f,buf,N, [Result]). Здесь buf – переменная - буфер, N – число компонент, Result – возврат фактического числа компонент операции.

Получить номер текущего элемента (текущей позиции указателя) можно функцией FilePoz(f), а узнать размер файла – через FileSize(f).

Остальные команды и функции аналогичны работе с типизированными файлами.

Кроме этого, работу с файлами можно организовать и средствами библиотеки VCL. Например, компонент DriveComboBox предназначен для выбора дисковода компьютера, а компонент DirectoryListBox служит для отображения дерева каталогов и перемещения по нему. Компонент FileListBox применяется для просмотра списка файлов, содержащихся в выбранном каталоге, и для выбора файла. Его свойство FileName содержит полное имя выбранного файла. Компонент FilterComboBox предназначен для выбора маски списка файлов. У него есть свойство Mask и, если он связан с компонентом FileListBox, то список выбора файлов последнего работает в соответствии с заданной маской.

Практика решения вычислительных задач на языке Delphi

1. Расчет математических выражений на языке программирования Delphi

Рассмотрим пример создания расчетной программы. Пусть необходимо получить цифровые данные для построения графика безопасной нагрузки для некоторой строительной колонны как функции отношения высоты колонны h к ее диаметру d. Находим в справочнике эмпирические формулы, позволяющие определить эту безопасную нагрузку:

$$S = \begin{cases} 1195 - 0{,}034R^2 & \text{при } R < 120 \\ \dfrac{1265}{1 + \dfrac{R^2}{1265}} & \text{при } R \geqslant 120 \end{cases}$$

, где S – безопасная нагрузка в кГ/см^2, R – отношение высоты колонны к диаметру.
Безопасную нагрузку требуется вычислить для значений R от 20 до 200 с шагом 5.

Имея математическое описание, используя пакет Visio, составляем алгоритм программы, рис.28 .

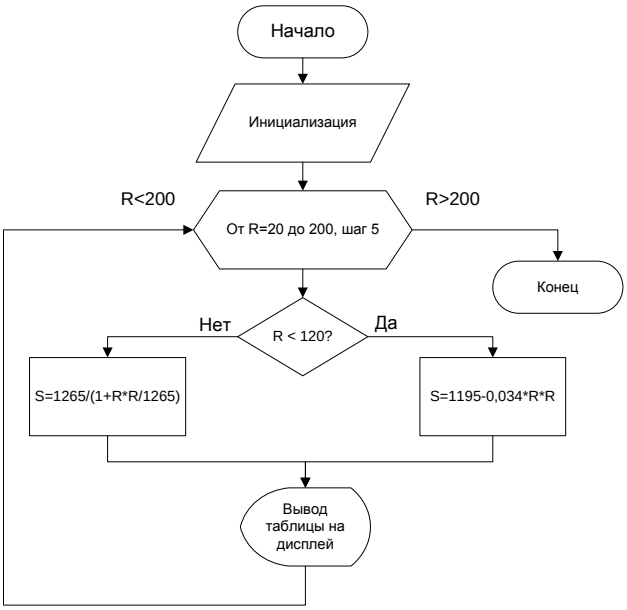


Рис.28. Алгоритм расчета таблицы нагрузок

Далее, запустив пакет Delphi, проектируем дизайн формы программы, рис.29 .

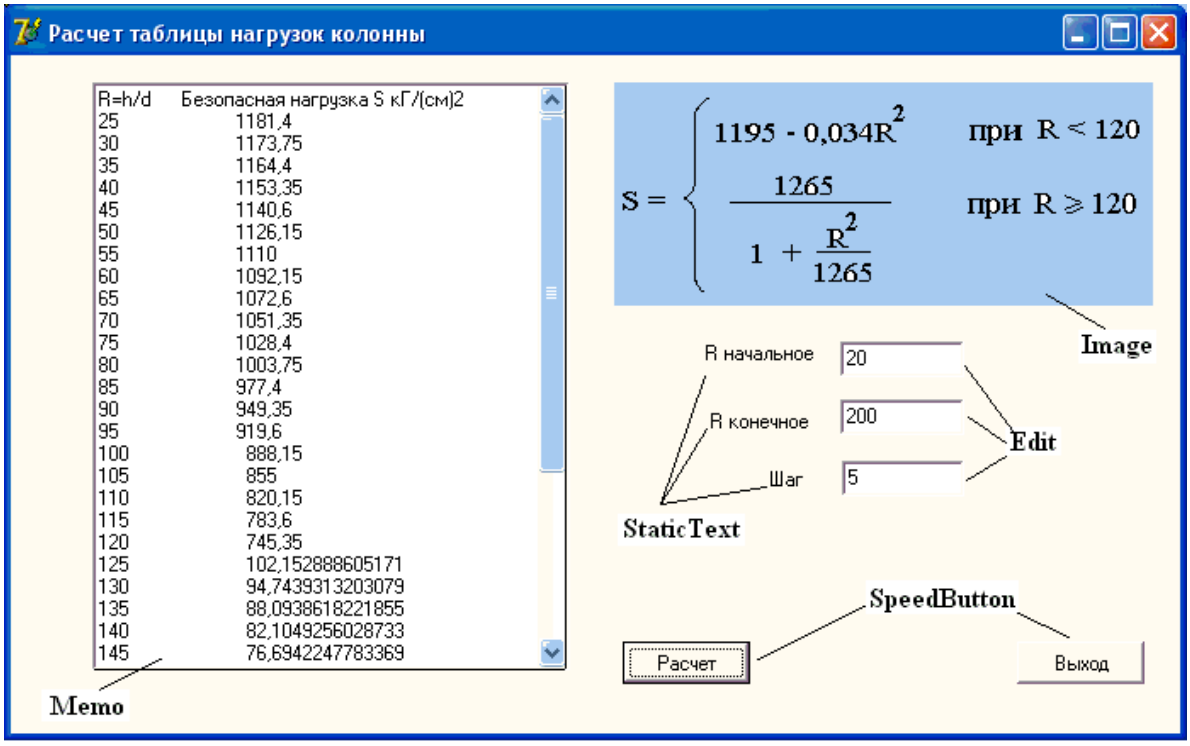


Рис.29 Примерный вид экранной формы программы расчета нагрузок

Фрагмент программного кода:

```
procedure TForm1.BitBtn1Click(Sender: TObject);
var
 s,R,rn,rk,stp: real;                // Объявляем переменные
 i,n: integer;
begin
 rn:=StrToFloat(Edit1.Text); //Считываем данные с экрана
```

```
 rk:=StrToFloat(Edit2.Text);
 stp:=StrToFloat(Edit3.Text);
 n:= Trunc((rk-rn)/stp);          //определяем, сколько раз подсчитывать
 R:=rn;                           //начальное значение
  for i:=1 to n do                //цикл
  begin
  if(R<120) then s:=1195-0.034*R*R
  else
  s:=1265/(1+R*R/1265);
  R:=R+stp;
  Memo1.Lines.Add(FloatToStr(R)+'                  '+FloatToStr(S)); //на экран
  end;
end;
procedure TForm1.BitBtn2Click(Sender: TObject); //Выход
begin
close;
end;
end.
```

Практическая работа.

Задание. Спроектировать экранную форму, содержащую рисунок с параметрами задачи, расчетные переменные и результат вычисления.

Цель работы. Изучить элементарные навыки и приемы программирования вычислительных задач.

План работы:

2. Найти свой вариант задачи и разработать алгоритм вычисления в пакете Visio.
3. Используя графический редактор Paint, создать рисунок с заданными выражениями в соответствии с вариантом задачи.
4. Запустив Delphi, создать экранную форму, содержащую рисунок с параметрами задачи, расчетные переменные и результат расчета аналогично форме, приведенной на рис.25.
5. Написать программу обработки нажатия кнопок, используя процедуру и функцию для расчета промежуточных переменных в некотором диапазоне.
6. Используя любой текстовый редактор, написать отчет по лабораторной работе.
7. Результаты работы в виде программы и отчета предъявить преподавателю.

Для дизайна формы используем компоненты: **Image** – для загрузки картинки с заданием, картинку загружаем через свойство Picture, свойством Stretch подгоняем ее под размер; для ввода параметра Y используем компонент **Edit**, комментарии на форму помещаем с помощью компонента **Label**; кнопки выбираем типа **Speed Button**; результаты расчетов выводим на форму через компонент **Memo**.

Вариант 1

$$y = \operatorname{ctg} x - \sqrt{1+x^2}, \text{ где}$$

$$x = \frac{z_1^3 - 1}{z_2^2 + 1}; \; z_1 = \frac{90 - b^2}{\sqrt{3b^2 + 1}};$$

$$z_2 = \begin{cases} \operatorname{tg} b, & \text{если } b = 10 \\ 1, & \text{если } b < 10, \end{cases}$$

значение «b» задается.

Вариант2

$y = e^x\sqrt{1+e^{2x}} + \operatorname{arctg} e^x$, где

$x = z_2 \cos^2 z_1 + \sin a^2;\ z_1 = \dfrac{3a}{1-2a};$

$z_2 = \begin{cases} a^2, & \text{если } a>0 \\ 0, & \text{если } a \leq 0 \end{cases}$, значение «$a$» задается.

Вариант 3

$y = x^2 + b - \sqrt{b^2}\sin b$, где

$x = \dfrac{\operatorname{tg} z_1}{z_2};\ z_1 = \dfrac{\sin^2 \dfrac{a}{b}}{a^2};$

$z^2 = \begin{cases} ab, & \text{если } a<0, b>0 \\ 1, & \text{если } a<0, b<0, \end{cases}$ значения «a» и «b» задаются

Вариант 4

$y = \operatorname{arctg} x + \ln\sqrt{\dfrac{1+x}{1-x}} + e^x$, где

$x = \dfrac{1-\sin m}{a_1^2};\ a_1 = \dfrac{\cos \dfrac{b}{a_2}}{b}$

$a_2 = \begin{cases} |b|, & \text{если } b<0 \\ 2, & \text{если } b>0, \end{cases}$ значения «b» и «m» задаются.

Вариант 5

$$y = x\,\mathrm{arctg}\,\frac{x}{a} - \frac{a}{2}\ln(x^2 + a^2),\ \text{где}$$

$$a = \frac{0.01\cdot(z_1^3+1)}{tg(z_2^2+1)};\ z_2 = \frac{\sqrt{1-\cos^2 2c}}{(\cos 2c)};$$

$$z_1 = \begin{cases}\sqrt{c}, & \text{если, } c>0\\ -c, & \text{если, } c\leqslant 0,\end{cases}$$ значения «*c*» и «*x*» задаются.

Вариант 6

$$y = b\ln(\sqrt{x+t}+\sqrt{x}) - \sqrt{x^2+tx},\ \text{где}$$

$$x = \frac{b\sqrt{z+1}}{tz+1};\ z = \sqrt{|\sin|b||};$$

$$t = \begin{cases}b^2, & \text{если, } b>1\\ e^b, & \text{если, } b\leqslant 1,\end{cases}$$ значение «*b*» задается.

Вариант 7

$$y = x^{tk} + \lg(\sin^2 x + \sqrt{1+\sin^3 x}),\ \text{где}$$

$$x = \frac{3z+\mathrm{tg}(tz)}{(t+1)};\ t = \frac{z}{1+e^k};$$

$$z = \begin{cases}|\sin k|, & \text{если, } 0<k\leqslant 10\\ 10, & \text{если, } k=0,\end{cases}$$ значение «*k*» задается.

Вариант 8

$$y = \frac{a}{2}(e^{\frac{x}{2}} + e^{-\frac{x}{2}}),\ z = \frac{f_1^3 + f_2}{\lg|2f_2|},$$

$$f_1 = 2^{\frac{1}{k-2}}\ f_2 = \sqrt{|k|}\sin x,$$

$$x = \begin{cases}k+1, & \text{если } k>0,\\ k-1, & \text{если } k\leqslant 0,\end{cases}$$ значения «*k*» и «*a*» задаются.

Вариант 9

$$y = \sqrt{z_1}\frac{\cos x}{\sin^2 x} + \ln\left|\operatorname{tg}\frac{x}{2}\right|;$$

$$x = \frac{1{,}5z-5}{3z+5};\ z = \frac{e^{-iz_1}-1}{z_2};$$

$$z_1 = \begin{cases} |z_2|, & \text{если } z_2 < 0, \\ \dfrac{z_2}{8}, & \text{если } z_2 > 0, \end{cases}$$ значение «z_2» задается.

Вариант 10

$$f = (z+1)\operatorname{arctg} 2z,\ z = \frac{x_1^2 - x_1 - 2}{x_2^3 + 1},$$

$$x_1 = \frac{e^{2x_2}}{\sin(2y)},\ x_2 = \begin{cases} \sin y, & \text{если } 100 \leqslant y \leqslant 200 \\ \cos y, & \text{если } y < 100, \end{cases}$$ значение «y» задается

Вариант 11

$$f(x) = \begin{cases} -x-1, & x < -1 \\ x-1, & -1 \le x < 0 \\ -x+1, & 0 \le x < 1 \\ x+1, & x \ge 1 \end{cases}$$ **значение «x» задается в пределах от -2 до +2**

Вариант 12

F(x) = $2x^2$-1, если 0 < x <= 2
e^x, если x > 2
|x|, если x <= 0

2. Построение графиков функций средствами языка Delphi

Рассмотрим пример создания программы отображения графика функции какой-то либо переменной. Пусть требуется отобразить график функции $y=e^{(x*x/2)} * \cos(1{,}5*x)$ в пределах от 0 до 1 с шагом 0,1 , причем пределы и шаг расчета можно изменять. Сначала разработаем алгоритм программы, но так как он аналогичен рассмотренным ранее циклическим алгоритмам, то здесь его не приводим. Далее проектируем экранную форму, рис.30.

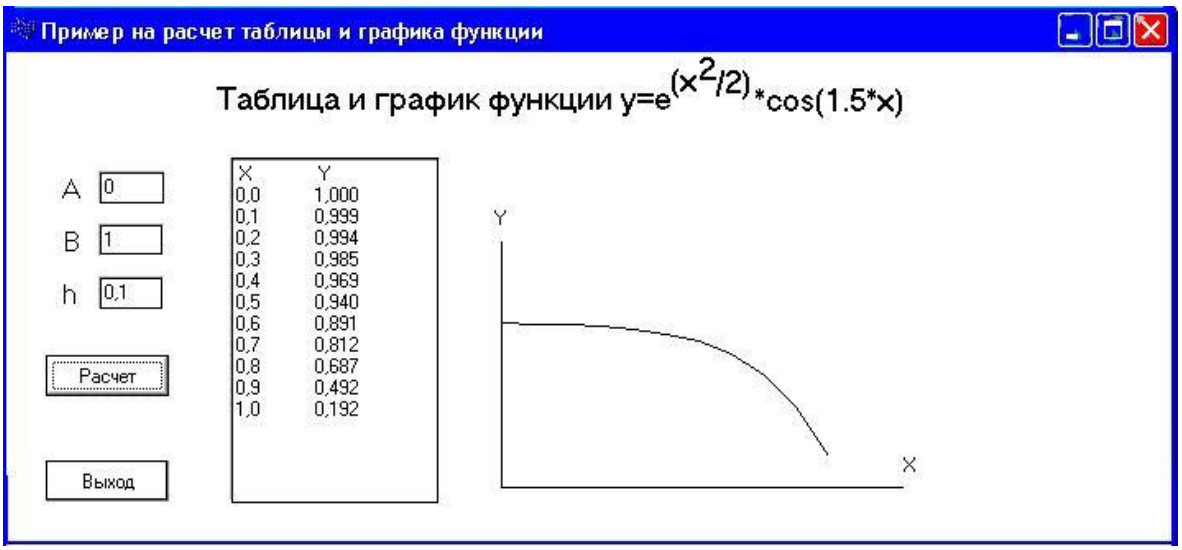


Рис.30. Примерный вид формы вывода графика функции

Фрагмент программного кода рисования графика:

```
…….
VAR
x,y,a,b,h : double;
begin
a=StrToFloat(Edit1.Text);   // берем границы и шаг
b=StrToFloat(Edit2.Text);
h=StrToFloat(Edit3.Text);
Memo1.Clear();
Memo1.Lines.Add(" X\t Y");
  PaintBox1.Canvas.Brush.Color=clLtGray;
  // рисуем оси графика
  PaintBox1.Canvas.MoveTo(5,200); //начало
  PaintBox1.Canvas.LineTo(250,200); //ось x
  PaintBox1.Canvas.MoveTo(5,200);
  PaintBox1.Canvas.LineTo(5,50); // ось y
  PaintBox1.Canvas.TextOutA(1,30,'Y');
  PaintBox1.Canvas.TextOutA(250,180,'X');
  PaintBox1.Canvas.MoveTo(5,200);
 for x:=a to b step h do begin              // организуем цикл
   y=exp(pow(x,2))*cos(1.5*x);
  Memo1.Lines.Add(FloatToStrF(x,ffFixed,3,1)+"\t"+FloatToStrF(y,ffFixed,5,3));
 PaintBox1.Canvas.LineTo(floor(200*x)+5,200-floor(100*y));
```

Практическая работа

Задание. Спроектировать экранную форму, содержащую график заданной функции с параметрами задачи и результат расчета значений функции на заданном интервале.

Цель работы. Научить студентов элементарным навыкам и приемам программирования вычислительных задач построения графиков функций.

План работы:

1. Найти свой вариант задачи и разработать алгоритм вычисления в пакете Visio.
2. Запустив Delphi, создать экранную форму, содержащую график функции с параметрами задачи, расчетные значения максимума и минимума аналогично форме, приведенной на рис.26.

3. Написать программу обработки нажатия кнопок.
4. Используя текстовый редактор, написать отчет по лабораторной работе. Результаты работы в виде действующей программы и отчета предъявить преподавателю.

Таблица

№	Заданная функция	A и B	Метод поиска
1	Y = X*X*sin(1/X)	-0.5, 3	Равномерный поиск
2	Y = sin(2*X)/(3-sin(X))	0, π	дихотомии
3	$Y = X^3 + X^2 + 3 - 3*X$	-1, 3	Равномерный поиск
4	Y = sqrt(X*X+2) - 0.1*X	0, 3	Равномерный поиск
5	Y = sin(-1/X)*cos(X)	1, 2	дихотомии
6	Y = X*sin(4*X)	0.3, π /2	Равномерный поиск
7	$Y = e^{(sqrt(X))}*sin(2*x)$	0, π	дихотомии
8	$Y = X^{cos(X)} - 4*X^2 + 1$	0, π/2	Равномерный поиск
9	$Y = 9*e^{(sin(X))} + 0.3$	0, π	дихотомии
10	Y = ((2 - exp(X))*sin(X/2))/(1+X)	0, 2	Равномерный поиск
11	Y = sqrt(X)*abs(sin(X/3)	0, π	дихотомии
12	$Y = 0.2*X^6 - 1.3*X^2 + 1.7$	-2, 2	дихотомии

3. Создание программы поиска экстремума функции

Рассмотрим пример создания программы поиска экстремума функции. Поиск экстремума является часто встречаемой задачей. Наиболее простой задачей является поиск экстремума одной переменной, но целевая функция может иметь и несколько аргументов, то есть быть многомерной, в общем случае $F(x_1, x_2, \ldots x_n)$. Кроме этого, функция может быть мультимодальной, то есть иметь несколько экстремумов, например функция Растригина.

Рассмотрим поиск экстремума унимодальных функций, то есть, имеющих один экстремум на заданном интервале, подобно функции, представленной на рис.1а). Самым простым является алгоритм равномерного поиска. Он основан на последовательном просмотре движения вверх по возрастающему склону функции и сравнения очередного значения функции с предыдущим значением. Как только очередной шаг приведет к тому, что мы обнаружим спуск вместо подъема, то поиск прекращается. Составим алгоритм программы, рис. 31.

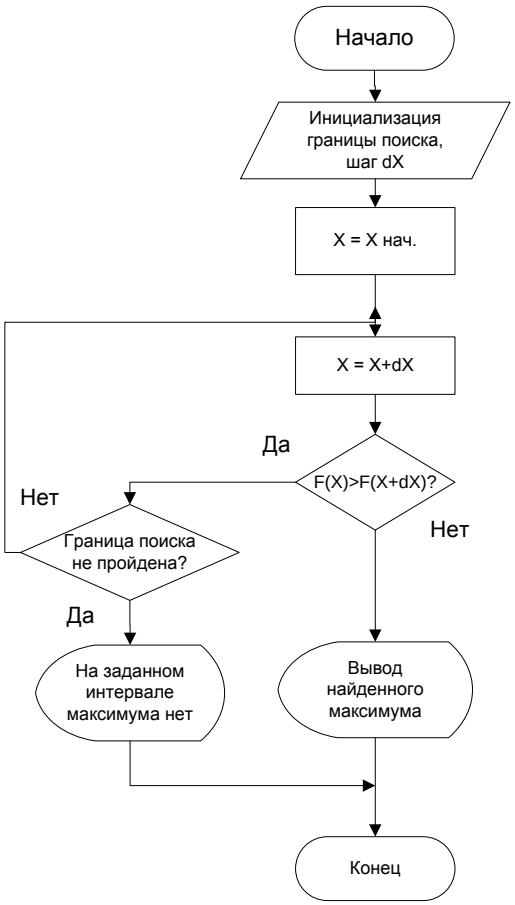


Рис.31. Алгоритм поиска максимума методом равномерного поиска

Рассмотрим в качестве альтернативы этому методу более быстродействующий метод дихотомии. Если интервал поиска обозначить как a и b, а точность e, то словесный алгоритм поиска будет следующим:

1. Проверяем условие |b - a| < e? Если это условие выполняется, то переходим к п.6, а если нет, то к п.2.
2. Делим интервал поиска пополам x = (a + b)/2 и вычисляем две точки относительно этой середины: $x_1 = (a + b – e)/2$ и $x_2 = (a + b + e)/2$.
3. Для этих значений вычисляем $F1(x_1)$ и $F2(x_2)$.
4. Проверяем условие: $F1(x_1) > F2(x_2)$? Если оно выполняется, то смещаем границу поиска, принимая $b = x_2$ и переходим к п.1, а если нет, то к п.5.
5. Принимаем $a = x_1$ и переходим к п.1.
6. Выводим на экран найденное значение максимума $F(x_m)$, где $x_m = (a + b)/2$.

Пример подпрограммы расчета максимума/минимума функции:

```
// Поиск экстремума функции на интервале методом дихотомии
// на входе границы  a и b, на выходе значения  x и y
     procedure extr(a,b:real;var x,y:real);
    var x,x1,y,y1,y2,n,m,e :real;
    begin
     n:=a; m:=b; e:=0.001; // границы и точность расчета
     repeat
        x1:=(n+m-e)/2;
        x:=x1;
        calc(x);  // заданная функция в виде подпрограммы
        y1:=y;
```

```
    x2:=(n+m+e)/2;
    x:=x2;
    calc(x);
    y2:=y;
    if y1<y2 then m:=x2
             else n:=x1;
  until abs(m-n)<2*e;
   x:=(n+m)/2;
   calc(x);
   end;
```

Практическая работа

Задание. Модифицировать экранную форму, содержащую график заданной функции с параметрами задачи для расчета минимального и максимального значения функции на заданном интервале.

Цель работы. Научить студентов элементарным навыкам и приемам программирования вычислительных задач построения графиков функций.

План работы:

1. Найти соседний вариант задачи построения графика функции, приведенный в таблицеXX и разработать алгоритм вычисления в пакете Visio согласно заданному методу поиска.
2. Запустив Delphi, создать экранную форму, содержащую график функции с параметрами задачи, расчетные значения максимума и минимума аналогично форме, приведенной на рис.26.
3. Написать программу обработки нажатия кнопок.
4. Используя текстовый редактор, написать отчет по лабораторной работе. Результаты работы в виде действующей программы и отчета предъявить преподавателю.

4. Создание программы вычисления бесконечных рядов на языке Delphi

В качестве примера расчетных заданий рассмотрим задачу вычисления бесконечного ряда. В инженерной практике ряды играют роль эффективного средства математического исследования и моделирования. Бесконечный ряд может быть как сходящимся, так и расходящимся. Если ряд расходящийся, то имеет смысл вычисления только некоторого определенного числа его членов, если ряд сходящийся, то $\lim\limits_{n\to\infty}(R_n)=0$ и можно подсчитать его сумму или произведение. Таким образом, сначала необходимо выяснить тип ряда. Например, исследуем ряд $\sum\limits_{n=1}^{\infty}(\frac{2n-1}{n+2})^n$ на сходимость. Для этого оценим $\lim\limits_{n\to\infty}\sqrt[n]{(\frac{2n-1}{n+2})^n}=\lim\limits_{n\to\infty}(\frac{2n/n-1/n}{n/n+2/n})$ =2, так как 2 > 1, то по признаку Коши ряд расходится.

Если ряд сходится, то необходимо выяснить, задан ли n –ный член ряда в общем виде, и, если не задан, то необходимо его определить методом математической индукции как разницу между соседними членами ряда.

Если в выражение входит степенная функция, то используется функция pow(x,y), соответствующая x^y. Если в выражение входит функция n!=1*2*3*4…n, то можно использовать рекурсивную функцию:

```
factorial(n:integer):integer;
var
```

```
a:integer;
begin
if(n=1) then a:= 1
else
a:= factorial( n-1)*n;
factorial:= a;
end;
```

Практическая работа

Задание. Разработать алгоритм и спроектировать экранную форму, содержащую рисунок заданной функции вычисления ряда и результат вычисления его значения.

Цель работы. Научить студентов элементарным навыкам и приемам программирования вычисления бесконечных рядов.

План работы:

1. Найти свой вариант задачи и разработать алгоритм вычисления в пакете Visio.
2. Запустив Delphi, создать экранную форму, содержащую необходимые графические элементы, параметры задачи, и расчетные значения, аналогично форме, приведенной на рис.27.
3. Написать программу обработки нажатия кнопок.
4. Используя текстовый редактор, написать отчет по лабораторной работе. Результаты работы в виде действующей программы и отчета предъявить преподавателю.

Вариант 1.

Вычислить выражение с заданной точностью:

$y = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots$ для $|x| < 1$ с точностью $e=2{,}5*10^{-5}$.

Вариант 2.

Вычислить выражение с заданной точностью:

$z = 1 + 2\sum_{k=1}^{\infty}(-1)^k e^{-2kx}$ для $0 < x < 1$ с точностью $e=10^{-7}$.

Вариант 3.

Вычислить выражение с заданной точностью:

$y = x\prod_{k=1}^{\infty}(1 - \frac{x^2}{k^2\pi^2})$ для $|x| < 1$ с точностью $e=10^{-6}$.

Вариант 4.

Вычислить выражение с заданной точностью:

$y = \sum_{k=1}^{\infty}\ln(1 - \frac{x^2}{k^2\pi^2})$ для $0 < |x| < 1$ с точностью $e=10^{-7}$.

Вариант 5.

Вычислить выражение с заданной точностью:

$y = \sum_{k=1}^{\infty}\frac{x}{(2k-1)^4}$ для $0 < |x| < 1$ с точностью $e=0{,}3*10^{-5}$.

Вариант 6.

Вычислить выражение с заданной точностью:

$y = \sum_{n=0}^{\infty} 2^n \sin(\frac{x}{3^n})$ для $0 < |x| < 1$ с точностью $e=0{,}3*10^{-7}$.

Вариант 7.
Вычислить выражение с заданной точностью:

$y = \sum_{n=1}^{\infty} \frac{(-1)^n}{x^n n!}$ для $0 < x < 1$ с точностью $e=0{,}3*10^{-4}$.

Вариант 8.
Вычислить выражение с заданной точностью:

$y = \sum_{n=0}^{\infty} \frac{x^{2n+1}}{2n+1}$ для $0 < x < 1$ с точностью $e=10^{-7}$.

Вариант 9.
Вычислить выражение с заданной точностью:

$y = \sum_{k=1}^{\infty} \frac{2x}{(2k-1)2k}$ для $0 < x < 1$ с точностью $e=10^{-5}$.

Вариант 10.
Вычислить выражение с заданной точностью:

$y = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + ... + \frac{x^n}{n!} + ...$ для $|x| < 1$ с точностью $e=10^{-4}$.

Вариант 11.
Вычислить выражение с заданной точностью:

$y = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + ...$ для $|x| < 1$ с точностью $e=0{,}5*10^{-4}$.

Вариант 12
Вычислить выражение с заданной точностью:

$y = -\frac{(2x)^2}{2} + \frac{(2x)^4}{24} + ... + (-1)^m \frac{(2x)^{2m}}{(2m)!}$ для $|x| < 1$ с точностью $e=2{,}5*10^{-5}$.

5. Создание программы сортировки информации на языке Delphi

Рассмотрим пример создания программы сортировки информации. Как отмечалось ранее, задача сортировки данных может решаться разными способами. Рассмотрим для примера метод простого выбора. Алгоритм основан на поиске максимального элемента и обмену его местами с последним элементом списка. Далее процедура повторяется с оставшимися элементами. Составим алгоритм программы, рис. 32.

Сначала программы формируем одномерный массив чисел тем или иным способом, например, используя случайную функцию. Затем организуем цикл перебора массива с его конца до начала, исключая самый первый элемент. Затем берем последний элемент неотсортированного массива и его индекс как отправную точку, заметим, что после каждого цикла размер неотсортированного массива будет уменьшаться на единицу, и проверяем, является ли очередной элемент больше этого последнего элемента. Если больше, считаем его максимальным и отправляем в конец. Таким образом, будем добавлять очередной максимальный элемент оставшейся части исходного массива в конец до тех пор, пока массив

не кончится. После этого выводим отсортированный массив на экран (на алгоритме цикл вывода не показан).

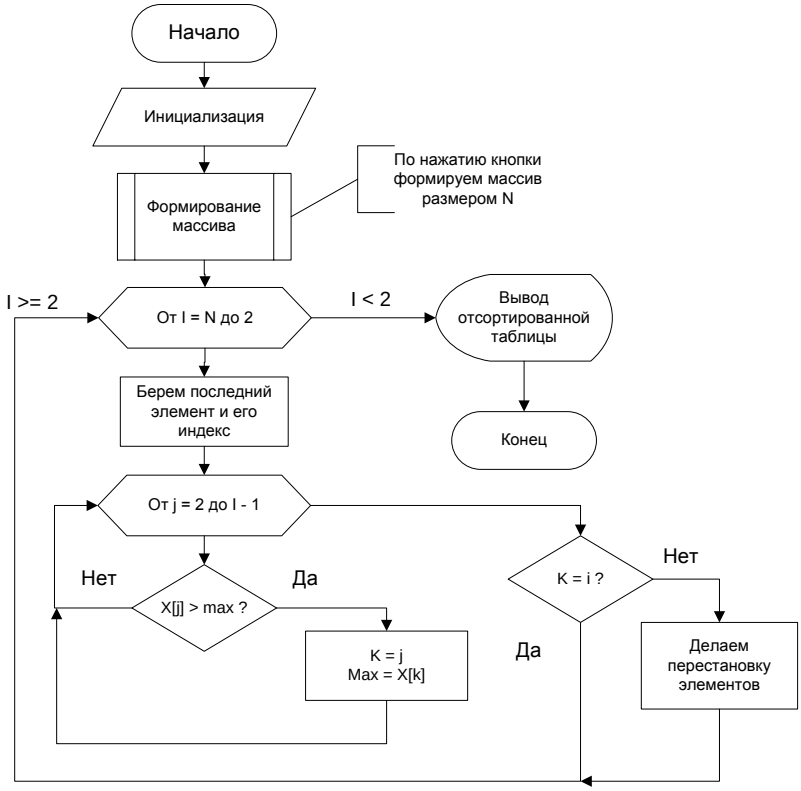


Рис. 32.Алгоритм программы сортировки массива

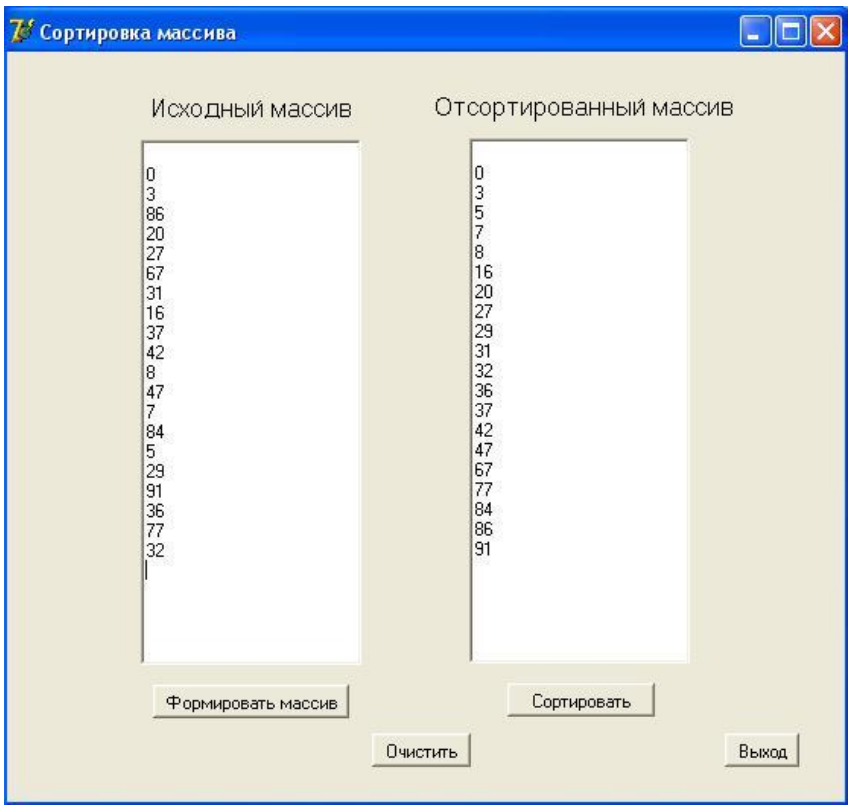


Рис.33. Экранная форма программы сортировки массива

Исходя из разработанного алгоритма, можно перейти к разработке интерфейса программы, рис.33, и программного кода, приведенного ниже.

Фрагмент программного кода сортировки массива методом простого выбора

```
// Формирование массива
procedure TForm1.SpeedButton1Click(Sender: TObject);
var
  i:integer;
begin
 for i:=1 to 20 do
 begin
 x[i]:=random(100);                    // Функция случайного числа
 Memo1.Lines.Add(IntToStr(x[i]));
 end;
end;
// Сортировка
procedure TForm1.SpeedButton2Click(Sender: TObject);
var
  i,j,k,max:integer;
  begin
 // Поиск максимального числа и его номера k
  max:=x[1];
 for i:=2 to 20 do
 begin
  if(x[i]>max) then begin max:=x[i]; k:=i; end;
  end;
 for i:= 20 downto 2 do
 begin
 k:=i; max:= x[i];
  for j:=2 to i-1 do
     if x[j] > max then begin k:=j; max:=x[k]; end;
     if k<>i then begin x[k]:=x[i]; x[i]:=max; end;
 end;
 // Вывод отсортированного массива на экран
  for i:=1 to 20 do
 begin
 Memo2.Lines.Add(IntToStr(x[i]));
 end;
end;
// Очистка
procedure TForm1.SpeedButton3Click(Sender: TObject);
begin
 Memo1.Lines.Clear;
 Memo2.Lines.Clear;
end;
// Выход
procedure TForm1.SpeedButton4Click(Sender: TObject);
begin
close;
end;
```

Другим, не менее известным способом сортировки, является метод простого обмена или метод «пузырька». Для этого метода характерно постепенное перемещение «всплытие» элементов по заданному свойству в конец (или начало) списка.

Словесный алгоритм метода следующий:

1. Берем первую пару элементов массива X(1) и X(2) и, если первый элемент больше второго, то элементы переставляем, иначе оставляем без изменений.
2. Затем берем вторую пару элементов X(2) и X(3) и, если X(1) > X(2), также переставляем.
3. Повторяем действия от I=1 до N-1. В результате максимальный элемент переместится в самый конец массива.
4. Уменьшаем размер массива до N-1, так как максимальный элемент массива уже находится на своем месте, повторяем действия. результате

Практическая работа

Задание. Разработать алгоритм и спроектировать экранную форму, содержащую неотсортированный массив и результат его сортировки.

Цель работы. Научить студентов элементарным навыкам и приемам программирования сортировки данных.

План работы:

1. Найти свой вариант задачи и разработать алгоритм вычисления в пакете Visio.
2. Запустив Delphi, создать экранную форму, содержащую необходимые элементы аналогично форме, приведенной на рис.33.
3. Написать программу обработки нажатия кнопок.
4. Используя текстовый редактор, написать отчет по лабораторной работе. Результаты работы в виде действующей программы и отчета предъявить преподавателю.

Таблица

№	Метод сортировки	Порядок	Размер массива
1	Простого выбора	по возрастанию	10
2	Пузырьковый	по возрастанию	10
3	Простого выбора	по убыванию	20
4	Пузырьковый	по убыванию	20
5	Простого выбора	по возрастанию	24
6	Пузырьковый	по возрастанию	24
7	Простого выбора	по убыванию	12
8	Пузырьковый	по убыванию	12
9	Простого выбора	по возрастанию	16
10	Пузырьковый	по возрастанию	16
11	Простого выбора	по убыванию	18
12	Пузырьковый	по убыванию	18

6. Работа с файлами и обработка текстовой информации на языке Delphi

Рассмотрим задачу создания программы чтения текстовой информации из файла и ее обработки. Обработка текстовой информации, как правило, сводится к решению следующих задач: поиск слова в файле; замена одного слова на другое; подсчет числа встречающихся слов; поиск подстроки в строке текста; вставка слов в нужное место; сортировка по алфавиту; шифрование и дешифрование и т.д. Рассмотрим пример разработки программы, реализующей некоторые функции обработки. Пусть исходная текстовая информация содержится в файле «Dann.txt», который можно создать с помощью стандартной программы

«Блокнот». Текстовая информация состоит из набора данных: порядковый номер, наименование, число штук и цена за одну единицу. Эта информация позволит хранить информацию, например, о товарах, хранимых на складе какой-нибудь организации. Чтобы работать со структурированной информацией, используем тип «запись» с полями «Номер», «Наименование», «Количество», «Цена».

Таблица XX

1 двигатель 15 2000
2 транзистор 100 15
3 робот 1 50000
4 колесо 22 200
5 батарея 55 100
6 компьютер 12 15000

Разработаем дизайн формы программы, рис. . Используем новый элемент для отображения данных: StringGrid. – таблица строк. Он достаточно удобен для работы с табличной текстовой информацией, в отличие от ранее использовавшегося объекта Memo.

Этот элемент обладает некоторыми свойствами, например:

- StringGrid1.Cells[номер столбца, номер строки] := ' Выводимая информация ' – вывод данных в указанную ячейку;
- StringGrid1.Cols[номер столбца].Clear – очистка указанного столбца;
- StringGrid1.Rows[номер строки].Clear – очистка указанной строки;
- StringGrid1.ColWidths[номер столбца] := число – задание ширины столбца;
- StringGrid1.RowHeights[номер строки] := число – задание высоты строки;
- ScrollBars – определяет вид полос прокрутки, если данные не помещаются полностью в таблицу. Если установить свойство в ssVertical, то используется вертикальная полоса прокрутки, если в ssHorizontal, то горизонтальная полоса прокрутки, а если ssBoth – то обе полосы прокрутки. По умолчанию свойство стоит на значении ssNone, то есть, полос прокрутки нет.

Остальные свойства этого объекта мы использовать не будем и их можно изучить самостоятельно.

В программе используем кнопку «Чтение файла» - где будет читаться информация из файла и выводиться на экран, кнопку «Сортировка» - где проведем сортировку текстовой информации методом «пузырька», кнопку «Нахождение слова и замена» - с помощью

которой напишем программу поиска заданного слова и замену его на другое. Необходимо отметить, что в приведенном фрагменте программы сортировка сделана только для поля «Наименование» и только для вывода на экран. Фрагмент программы замены слова также действует только на вывод на экран, то есть содержимое файла не изменяется в обоих случаях.

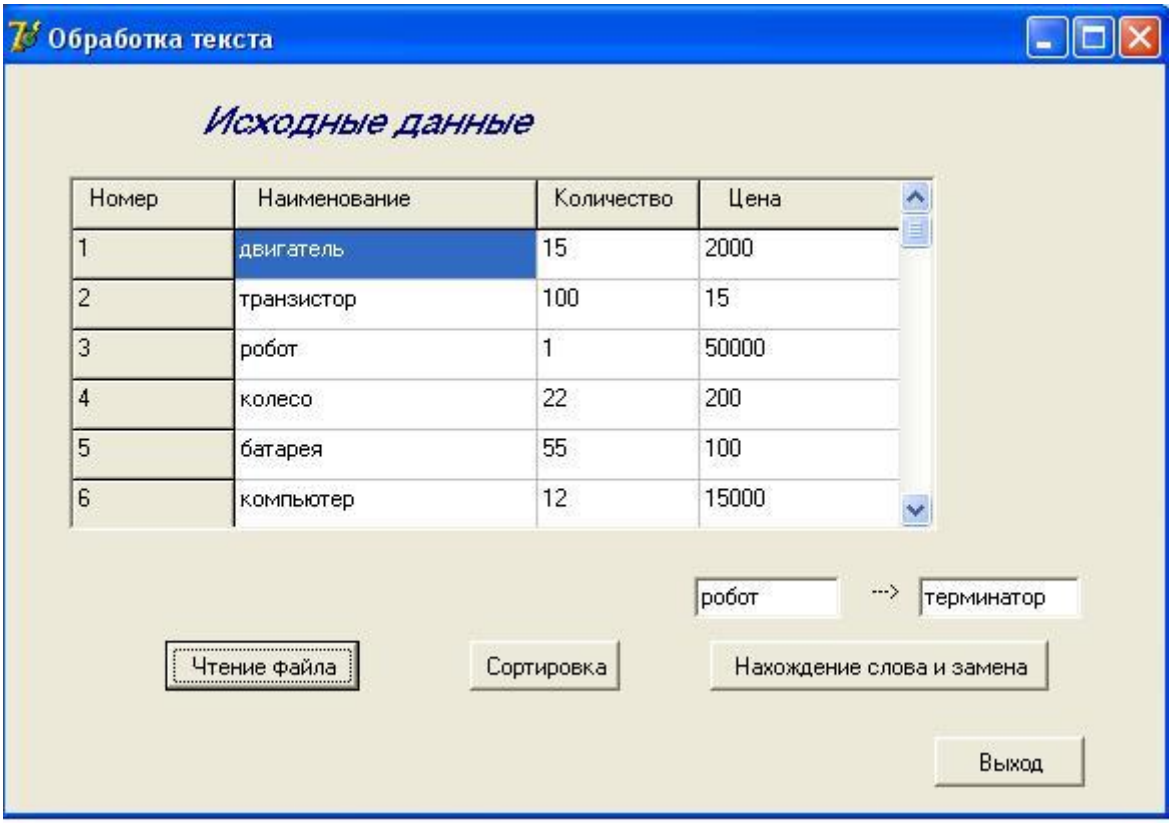


Рис.XX. Основная форма программы работы с текстовым файлом

Фрагмент программного кода работы с текстовым файлом:

```
  public
    k:integer;
    fin: textfile;            // текстовый файл
    slova: array[1..50] of string[15]; // вспомогательный массив для обработки до 50 слов
    { Public declarations }
  end;
 type
 tabl = record              // объявим тип «запись»
 nomer:  string[5];         // поле номер
 name:   string[15];        // поле наименование
 kolich: string[10];        // поле количество
 cena:   string[8];         // поле цена
 end;

var
  Form1: TForm1;

implementation

{$R *.dfm}
```

```
//чтоние данных из файла
procedure TForm1.BitBtn1Click(Sender: TObject);
var
 dann: tabl;
 begin
  AssignFile(fin,'dann.txt');     // связываем файловую переменную с файлом
  Reset(fin);
  while not SeekEof(fin) do
  begin
   with dann do begin  // читаем из файла
  readln(fin,nomer);
  readln(fin,name);
  readln(fin,kolich);
  readln(fin,cena);
// вывод на экран
  StringGrid1.Cells[0,k+1]:=nomer;
  StringGrid1.Cells[1,k+1]:=name;
  StringGrid1.Cells[2,k+1]:=kolich;
  StringGrid1.Cells[3,k+1]:=cena;
  k:=k+1;
              end;
  end;
  label1.Caption:='Исходные данные';
end;
 // сортировка методом "пузырька"
procedure TForm1.BitBtn2Click(Sender: TObject);
var
  dann: tabl;
  i:integer;
 stop:boolean;
 slov: string[15];
 begin
    StringGrid1.Cols[1].Clear;
    StringGrid1.Cells[1,0]:='   Наименование';
  Reset(fin);
  k:=1;
  while not SeekEof(fin) do
  begin
   with dann do begin
  readln(fin,nomer);
  readln(fin,name);
  readln(fin,kolich);
  readln(fin,cena);
  slova[k]:=name;
  k:=k+1;
              end;
  end;
// собственно сортировка
  repeat
  stop:=true;
  for i:=1 to k-1 do
    if slova[i] > slova[i+1] then
```

```
    begin
    slov:=slova[i];
    slova[i]:=slova[i+1];
    slova[i+1]:=slov;
    stop:=false;
    end;
  until stop;
// вывод на экран
  for i:=1 to k do begin
  StringGrid1.Cells[1,i]:=slova[i+1];
  end;
 label1.Caption:='Отсортированные данные';
 end;
// поиск и замена
procedure TForm1.BitBtn3Click(Sender: TObject);
var
 slov,zamena:string;
 i:integer;
 begin
  slov:=Edit1.text;
 zamena:=Edit2.Text;
  for i:=1 to k do
  begin
    if(slova[i]=slov) then slova[i]:=zamena;
  end;
// вывод на экран
  for i:=1 to k do begin
  StringGrid1.Cells[1,i]:=slova[i+1];
  end;
 label1.Caption:='С поиском и заменой';
end;
// Выход
procedure TForm1.BitBtn4Click(Sender: TObject);
begin
CloseFile(fin);
Close;
end;
// в самом начале создания формы выводим «шапку» таблицы
procedure TForm1.FormCreate(Sender: TObject);
begin
  StringGrid1.ColWidths[0]:=80;
  StringGrid1.Cells[0,0]:='  Номер';
  StringGrid1.ColWidths[1]:=150;
  StringGrid1.Cells[1,0]:='   Наименование';
  StringGrid1.ColWidths[2]:=80;
  StringGrid1.Cells[2,0]:='  Количество';
  StringGrid1.ColWidths[3]:=100;
  StringGrid1.Cells[3,0]:='    Цена';
  end;
```

Практическая работа

Задание. Разработать алгоритм и спроектировать экранную форму, содержащую прочитанный из файла массив структурированных данных, отсортировать его и провести поиск и замену найденного слова.

Цель работы. Научить студентов элементарным навыкам и приемам программирования работы с текстовым файлом, сортировки текстовых данных, поиска и замены.

План работы:

1. Найти свой вариант задачи и, запустив Delphi, создать экранную форму, содержащую необходимые элементы аналогично форме, приведенной на рис.XX.
2. Написать программу обработки нажатия кнопок.
3. Используя текстовый редактор, написать отчет по лабораторной работе. Результаты работы в виде действующей программы и отчета предъявить преподавателю.

Таблица XX

№	Тип данных	Сортировка	Число данных
1	Список студентов	по возрастанию	10
2	Учет материалов	по возрастанию	10
3	Учет книг	по убыванию	12
4	Учет компьютеров	по убыванию	10
5	Учет роботов	по возрастанию	14
6	Учет двигателей	по возрастанию	14
7	Учет автомобилей	по убыванию	12
8	Меню	по убыванию	12
9	Учет инструментов	по возрастанию	16
10	Учет принтеров	по возрастанию	16
11	Учет сотовых телефонов	по убыванию	18
12	Расписание	по убыванию	18

Контрольные вопросы

1. Какие формы представления информации вы знаете?
2. Какие единицы измерения информации и производительности компьютерной техники применяются в настоящее время?
3. Как производится переход от одной системы счисления к другой?
4. Какие виды компьютеров вам известны?
5. Нарисуйте и объясните структуру компьютера.
6. Как работает двоичная арифметика?
7. Какие элементы образуют АЛУ?
8. Что такое вредоносная программа и как с ней бороться?
9. Приведите состав типового офисного пакета программ.
10. Что такое программа?
11. Что понимают под исполнителем?
12. Что представляет собой машинный код?
13. Что такое транслятор? Перечислите типы трансляторов.
14. Как работает интерпретатор? В чем его достоинства?
15. В чем заключается достоинство компиляторов?
16. Какие компоненты необходимы для создания программ? Каково назначение каждого из этих компонентов?
17. Что называется интегрированной системой программирования?
18. Чем характеризуются системы визуального программирования?
19. Какие подходы по способу разработки программ можно выделить? Охарактеризуйте каждый подход.

20. Каковы основные системы программирования?
21. Перечислите основные этапы развития языков программирования.
22. Что понимают под алгоритмом?
23. Каковы способы записи алгоритмов?
24. В чем заключаются основные свойства алгоритма?
25. Перечислите основные алгоритмические структуры и опишите их.
26. Каковы основные принципы разработки алгоритмов?
27. Назовите основные этапы составления алгоритмов.
28. Приведите пример, реализующий этапы алгоритмизации.
29. Каковы основные этапы решения задач с помощью ЭВМ? Дайте характеристику каждому этапу.
30. Какие компоненты можно использовать для ввода информации и для вывода информации на форме?
31. Каким образом можно организовать процедуру и функцию в Delphi?
32. Какие методы сортировки информации вы знаете?

Литература.

1. Основы современных компьютерных технологий: Учебное пособие/ Под ред.проф. Хомоненко – СПб.: Корона принт, 2005.
2. Глушаков С.В. и др. Программирование на Delphi 5.0 – Харьков. Фолио, 2002.
3. Архангельский А.Я. Программирование в Delphi 7 – М: ООО БИНОМ-Пресс, 2003.
4. Delphi. Программирование на языке высокого уровня: Учебник для вузов / В.В.Фаронов. – СПб.: Питер, 2003.
5. Лабораторный практикум по информатике: Учеб. Пособие для вузов/В.С. Микшина, Г.А. Еремеева и др.; под ред. В.А. Острейковского. – 2-е изд., стер.- М.: Высш. Шк., 2006.