МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

ПРОБЛЕМНО ОРИЕНТИРОВАННАЯ ИНФОРМАТИКА ХИМИКО-ТЕХНОЛОГИЧЕСКИХ ПРОЦЕССОВ

Рекомендовано Сибирским региональным учебно-методическим центром высшего профессионального образования для межвузовского использования в качестве учебного пособия для студентов, обучающихся по специальности 240802.65 «Основные процессы химических производств и химическая кибернетика»

3-е издание

Издательство
Томского политехнического университета
2014

УДК 66.02:004(075.8)
ББК 35.11:32.97я73
П78

Авторы
А.В. Кравцов, Н.В. Чеканцев, Е.С. Шарова,
М.С. Гынгазова, Ю.А. Смышляева, Э.Д. Иванчина

П78 **Проблемно ориентированная информатика химико-технологических процессов**: учебное пособие / А.В. Кравцов, Н.В. Чеканцев, Е.С. Шарова, М.С. Гынгазова, Ю.А. Смышляева, Э.Д. Иванчина; Томский политехнический университет. – 3-е изд. – Томск: Изд-во Томского политехнического университета, 2014. – 160 с.

ISBN 978-5-4387-0397-6

В пособии изложены основные характеристики языков программирования, представлена технология программирования, показаны численные методы решения задач и функциональных выражений химической технологии. Описаны основы работы с Windows, со средой Delphi. Подробно рассмотрены особенности использования пакета Microsoft Office, а также такие важные темы, как: применение информационных технологий в химии; базы данных и базы знаний; сеть Интернет; средства защиты информации.

Предназначено для студентов, обучающихся по специальности «Химическая технология и биотехнология», также будет полезно для начинающих пользователей компьютера.

УДК 66.02:004(075.8)
ББК 35.11:32.97я73

Рецензенты

Кандидат химических наук
заведующая ЦЗЛ «Киришинефтеоргсинтез»
Г.В. Костина

Кандидат технических наук
научный сотрудник Института
сильноточной электроники СО РАН
Н.Ф. Ковшаров

ISBN 978-5-4387-0397-6

ВВЕДЕНИЕ

Информатика, как естественнонаучная дисциплина, отличается от химии, физики, высшей математики и других предметов постоянно меняющимся содержанием в связи с совершенствованием электронных носителей информации. В то же время деятельность инженера-химика в современных условиях сопряжена с постоянным использованием средств и методов информатики.

Широко внедряемые в последнее время новые информационные технологии позволяют осуществить компьютеризацию производства, что обеспечивает инженерно-технологический персонал заводов надежным инструментом для технологического и экономического прогнозирования химических процессов.

С другой стороны, для потенциально-опасных производств, на которых возможно возникновение аварийных ситуаций, приоритетной задачей является предотвращение таких ситуаций, вероятность возникновения которых в связи с интенсификацией процессов возрастает. Такой прогноз требует привлечения методов математического моделирования и искусственного интеллекта для оперативной диагностики действующего производства и выявления причин отклонения его показателей от регламента.

Кроме того, использование компьютерных интеллектуальных систем для обучения технологического персонала промышленных предприятий с наработкой у них навыков вывода химических установок из эксплуатационных и аварийных нештатных ситуаций позволяет существенно повысить технический уровень операторов и инженерно-технического персонала.

Построение таких компьютерных систем – это сложный многоэтапный процесс сбора, анализа и обработки разносторонней информации о химико-технологической сущности производства, методологической основой которого является математическое моделирование, а стратегией изучения – системный анализ. Сущность системного анализа заключается в том, что вся информация об изучаемом явлении постепенно накапливается и обобщается с целью разработки полной модели химико-технологического процесса.

Построение интеллектуальных компьютерных систем может быть выполнено и на основе корреляционных и аппроксимирующих зависимостей, однако точность прогнозов и технико-экономических оценок в этом случае невысока и справедлива для данной установки в конкретных условиях.

Основное назначение физико-химических моделей в интеллектуальных системах (ИС) и тренажерах для обучения по сравнению с информационными моделями и моделями представления знаний – это обеспечение возможности активного исследования и обучения – тренажа.

Основу физико-химических моделей объектов оставляет механизм и кинетика превращения реагентов на катализаторе, которые имеют ярко выраженный нестационарный характер вследствие изменения активности катали-

затора и износа оборудования. Практика показала, что без учета этих факторов ИС не будет отражать реальной ситуации.

Широкое внедрение в химию методов компьютерного моделирования и информационных систем не только не освобождают исследователя от необходимости знания математических методов и применения их в решении химических задач, но, напротив, делают это изучение одним из важнейших этапов подготовки специалиста – химика. Так, приближенно решение нелинейных уравнений позволяет быстро и с достаточной точностью определить выходы продуктов в химических процессах. Для исследования кинетических закономерностей химических явлений необходимо знание приближенных методов решения дифференциальных уравнений, а также методов вычисления интегралов. Владение элементами теории вероятностей и статистическими методами обработки результатов обязательно для анализа экспериментально полученных данных.

Наряду с этим, для эффективного поиска, переработки преобразования, распространения использования информации необходимо знание алгоритмических языков. «... без алгоритмов и алгоритмических языков предмета «Информатики» не существует», – писал академик Глушков.

Многолетняя практика подготовки технологов на химико-технологическом факультете показала, что только на основе принципа непрерывности и энциклопедичности образования можно научить студента диалектически мыслить, при этом нельзя абстрагироваться в понятиях «Информатика», так же, как и в понятии «Технология» от конкретной сущности информатики и технологии в определенной области естествознания. Поэтому анализ свойств химико-технологической системы при одновременном освоении законов преобразования информации целесообразно использовать при изучении курса «Информатики». Аналогично базы данных, которые содержат физико-химические и теплофизические свойства веществ, более доступны для восприятия будущих химиков-технологов, чем их общее понятие.

К сожалению, отсутствие учебников по информатике для химиков сдерживает освоение студентами необходимого материала и внедрение компьютерных методов в химию. В данном пособии изложены результаты практических навыков применения численных методов и языка программирования Турбо-Паскаль для решения задач химической технологии.

Этот курс читается на кафедре химической технологии топлива и химической кибернетики в течение 20 лет с постоянным совершенствованием на основе результатов научно-прикладной и учебно-методической работы. При этом накапливаются базы данных и базы знаний для основных процессов химической, нефтехимической и нефтеперерабатывающей промышленности.

Это пособие является завершением цикла методических разработок курса «Информатика», читаемых для студентов химико-технологического факультета Томского политехнического университета.

1. ОБЩАЯ ХАРАКТЕРИСТИКА ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

Разработка системного и прикладного программного обеспечения на персональных компьютерах осуществляется с помощью инструментальных средств, к которым в первую очередь относятся:

- трансляторы с языков высокого уровня;
- средства редактирования, компоновки и загрузки программ;
- макроассемблеры (машинно-ориентированные языки);
- отладчики машинных программ.

Основные инструментальные языки высокого уровня, используемые на персональных компьютерах – *Basic*, *Pascal*, *C* и др. Не вдаваясь в детальные описания указанных языков, рассмотрим кратко их основные свойства.

Basic

Исторически одним из самых популярных языков высокого уровня стал *Basic*. В чем причина этой популярности? Прежде всего *Basic* очень прост в освоении и использовании.

Режим интерпретации способствует сокращению характерного цикла в работе программиста: составлению программы – пробное использование – исправление ошибок – повторное исполнение. Это очень удобно при разработке небольших программ. Интерпретация, однако, имеет неизбежный недостаток - программа работает существенно медленнее, чем в случае использования трансляторов компилирующего типа, как, например, для языков *Pascal* и *C*. Объясняется это тем, что в режиме интерпретации каждый оператор языка сначала читается системой, анализируется в контексте уже работающей программы и лишь после этого исполняется. В трансляторах компилирующего типа, в отличие от этого, все стадии чтения и анализа осуществляется заранее – на этапе компиляции, а при исполнении работает готовая программа. Чтобы сохранить преимущества языка *Basic* и в то же время дать возможность построения эффективных, быстро работающих программ, созданы Basic-компиляторы. При этом на этапе составления и отладки программы используются преимущества интерпретационного режима, а после завершения отладки программа компилируется. Появления компилятора поставило *Basic* в один ряд с другими языками высокого уровня и придало ему дополнительную популярность.

Как правило, начинающие программисты пользуются этим языком для составления свои первых программ. *Basic* отводится ведущая роль в школьном образовании, как языку обучения основам программирования.

Языки Pascal и C

Языки *Pascal* и *C* чаще всего используются профессиональными системными программистами для разработки системных и прикладных программ. Оба эти языка позволяют работать с данными сложной структуры; оба имеют развитые средства для выделения отдельных частей программ в процедуры. Трансляторы этих языков работают в режиме компиляции, что позволяется создавать эффективные программы. Важным средством для построения больших программных систем является модульность, т. е. возможность независимой разработки отдельных частей программ и последующего их связывания в единую систему. Все эти особенности способствовали тому, что именно на *Pascal* и *C* разрабатывается большинство крупных программных систем.

Следует отметить, что между указанными языками, несмотря на общее сходство, имеются существенные различия. *Pascal* является классическим языком программирования, который приобрел популярность как отличный инструмент для решения серьезных задач. Программирование на *Pascal* обеспечивает высокую степень надежности программ.

Pascal, наряду с *Basic*, считается также учебным языком; он принят во многих учебных заведениях как базовый язык для изучения программирования.

Язык *C* в отличии от *Pascal* с момента появления был ориентирован на разработку системных программ. Он, в частности, послужил главным инструментом для создания операционных систем ЮНИКС и *MS-DOS*. В этом языке имеются более гибкие средства для эффективного использования особенностей аппаратуры, чем в *Pascal*. С другой стороны, синтаксис языка *C* менее прозрачен, чем у *Pascal*; возможностей для внесения ошибок больше; чтение текстовых программ требует определенного навыка. В связи с этим язык *C* применяется главным образом для создания системных и прикладных программ, в которых скорость работы и объем памяти являются критическими параметрами.

2. ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ TURBO PASCAL

2.1. Оператор присваивания

Общий вид оператора присваивания:

<имя переменной>:=<выражение>

В зависимости от типа переменной бывают математический, логический, символьный, строковый и др. операторы присваивания. В данной главе подробно будет рассмотрен математический оператор присваивания. Выражение в правой части оператора может представлять собой константу (a:=4E-3), имя переменной (d:=x) или математическое выражение, записываемое по следующим правилам:

1. Математическое выражение может состоять из констант, имен переменных и стандартных математических функций, соединенных знаками арифметических операций: "+", "-", "*", "/" (табл. 2.1.1, пример 1). Как и в математике низший приоритет имеют (т. е. выполняются в последнюю очередь) операции "+" и "–", более высокий "*" и "/" и наивысший – вызов функции. Для повышения приоритета операции используются скобки (в Паскале при записи математического выражения используются только круглые скобки (табл. 2.1.1, пример 2).

2. Если выражение представляет собой дробь, то оно «вытягивается в строчку», т. е. сначала записывается числитель, затем знак "/" затем знаменатель (табл. 2.1.1, пример 3). Если в числителе (или знаменателе) дроби стоит сумма (или разность), то т. к. операции "+" и "–" имеют более низкий приоритет, чем "/", то числитель (или знаменатель) заключаются в скобки (табл. 2.1.1, пример 4). Т. к. расчет выражения осуществляется слева направо, нет необходимости заключать в скобки произведение (частное), находящееся в числителе, но если оно стоит в знаменателе, то скобки ставить необходимо (табл. 2.1.1, пример 5).

3. Для вычисления функций в Паскале имеется набор стандартных математических функций (см. табл. 2.1.2). Для вызова функции пишется ее имя и затем аргумент в круглых скобках (табл. 2.1.1, пример 6). Если функция не является стандартной, необходимо выразить ее через стандартные математические функции (см. табл. 2.1.3), (табл. 2.1.1, пример 7).

Пример 2.1.1 Записать оператор присваивания.

а) $g = \dfrac{\sqrt{x^2 + \sin\dfrac{1}{4x}}}{tg^3 x} + 4 \cdot 10^5 \cdot \sqrt[5]{|1 - \arcsin ax|}$

g:=sqrt(sqr(x)+sin(1/(4∗x)))/exp(3∗ln(sin(x)/cos(x)))+
4e5∗exp(1/5∗ln(abs(1-sqrt((a*x)/(1-sqr(a*x))))));

б) $x = \dfrac{1+2x^{\sin x}}{\lg a/x} - e^{-x^3}$

x:=(1+2∗exp(sin(x)∗ln(x)))/(ln(a/x)/ln(10))-exp(-x∗sqr(x)).

Таблица 2.1.1

Запись математических выражений на Паскале

Математическая запись	Запись оператора на языке Паскаль
Арифметические выражения	
1. $x = a + 5.2 + 3\cdot 10^5 d$	x:=a+5.2+3E5∗d;
2. $x = (0.1+b)\left[c + 4d(a+d)\right]$	x:=(0.1+b)∗(c+4∗d∗(a+d));
3. $x = \dfrac{3.5}{a}$	x:=3.5/a;
4. $x = \dfrac{2+a-c}{4-d}$	x:=(2+a-c)/(4–d);
5.а. $x = \dfrac{2b(a+c)}{4.8\cdot 10^{-2} a(b-c)}$	x :=2∗b∗(a+c)/(4.8E-2∗a∗(b-c)); или x:=2∗b∗(a+c)/4.8E-2/a/(b-c);
5.б. $x = \dfrac{p}{q+1} \Big/ \dfrac{a-b+1}{pq}$	x:=p/(q+1)/((a–b+1)/(p∗q))или x:=p∗(p∗q)/(q+1)/(a–b+1)
Использование стандартных математических функций	
6.а. $x = \dfrac{\sin x}{e^x \cdot \ln x}$	x := sin(x)/(exp(x)∗ln(x));
6.б. $x = \ln\left\lvert x^3 + \sin^2 x\right\rvert$	x:=ln(abs(x∗sqr(x)+sqr(sin(x))))
7. $x = tg^3\left(\sqrt{x} - 3.3\right)$	x:=exp(3∗ln(sin(sqrt(x)-3.3)/cos(sqrt(x)-3.3))) или a:=sqrt(x)-3.3;x:=exp(3∗ln(sin(a)/cos(a)))

Таблица 2.1.2

Стандартные математические функции

x^2	sqr(x)	Вторая степень x
e^x	exp(x)	Экспонента x
Sin *x*	sin(x)	Синус x
Arctg *x*	arctan(x)	Арктангенс x
$\sqrt{x}$	sqrt(x)	Квадратный корень x
ln *x*	ln(x)	Натуральный логарифм x
cos *x*	cos(x)	Косинус x
[x]	int(x) или trunc(x)	Целая часть числа (для x>0).

Окончание табл. 2.1.2

{x}	frac(x)	Дробная часть числа
	round(x)	Округление до целого
$\lvert x \rvert$	abs(x)	Модуль числа

Таблица 2.1.3

Расчет некоторых нестандартных математических функций

1. Тригонометрические функции

$$tg(x)=\frac{\sin(x)}{\cos(x)},\ ctg(x)=\frac{\cos(x)}{\sin(x)},\ \sec(x)=\frac{1}{\cos(x)},\ \cos ec(x)=\frac{1}{\sin(x)}.$$

2. Обратные тригонометрические функции

$$\arcsin(x)=\arctan\left(\sqrt{\frac{x}{1-x^2}}\right),\ \arccos(x)=\frac{\pi}{2}-\arcsin(x).$$

3. Возведение в степень

$$a^x=e^{x\cdot\ln(a)},\ \sqrt[n]{x}=x^{1/n}=e^{\frac{\ln(x)}{n}}.$$

4. Расчет логарифмов

$$\log_a x=\frac{\log_b x}{\log_b a}=\frac{\ln(x)}{\ln(a)},\ \lg x=\frac{\ln(x)}{\ln(10)}.$$

5. Гиперболические функции

а) гиперболический синус $sh(x)=\dfrac{e^x-e^{-x}}{2}$,

б) гиперболический косинус $ch(x)=\dfrac{e^x+e^{-x}}{2}$,

в) гиперболический тангенс $th(x)=\dfrac{e^x-e^{-x}}{e^x+e^{-x}}$.

6. Обратные гиперболические функции

а) ареасинус $arsh(x)=\ln\left(x+\sqrt{x^2+1}\right)$,

б) ареакосинус $arch(x)=\ln\left(x+\sqrt{x^2-1}\right)$,

в) ареатангенс $arth(x)=\dfrac{1}{2}\cdot\ln\left|\dfrac{x+1}{x-1}\right|$.

2.2. Программирование линейных алгоритмов

В данном разделе представлены примеры программ, в основе которых лежит алгоритм линейной структуры, т. е. алгоритм, в котором действия выполняются последовательно друг за другом без разветвлений. Обычно в задачах такого типа первое действие – ввод исходных данных,

затем следуют необходимые расчеты, и в заключении - вывод результатов. Примеры оформления ввода и вывода данных приведены в табл. 2.2.1.

Пример 2.2.1. Составить программу определения площади и объема сферы.

$$S_{СФЕРЫ} = 4 \cdot \pi \cdot R^2 \quad V_{СФЕРЫ} = \frac{4}{3} \cdot \pi \cdot R^3 .$$

```
VAR R, S, V:real;
BEGIN
  {Ввод исходных данных}
  write('Введите радиус сферы R='); readln(R);
  {Расчет площади и объема сферы}
  S:=4*Pi*sqr(R);
  V:=4/3*Pi*exp(3*ln(R));
  {Вывод результатов}
  writeln('Площадь сферы, S=',S:12);
  writeln('Объем сферы, V=',V:12);
END.
```

Пример 2.2.2. Рассчитать давление углекислого газа, предположив, что его поведение подчиняется уравнению Редлиха–Квонга:

$$P = \frac{R \cdot T}{V - b} - \frac{a}{\sqrt{T} \cdot V \cdot (V + b)},$$

если известны температура газа T (K) и занимаемый им объем V ($м^3$). Параметры a и b определяются по формулам:

$$a = \Omega_a \cdot \frac{R^2 \cdot T_{кр}^{2.5}}{P_{кр}}; \qquad b = \Omega_b \cdot \frac{R \cdot T_{кр}}{P_{кр}},$$

где $T_{кр}$(К) и $P_{кр}$(Па) – критические температура и давление.

Ω_a=0.4274802327, Ω_b=0.08664035.

```
CONST
  {Описание констант}
  R=8.314;
  Sa=0.4274802327;
  Sb=0.08664035;
  Tk=31.0 + 273.15;
  Pk=7.39 * 101325;
VAR T, V, P, a, b:real;
BEGIN
  writeln('Введите значения температуры (T) и объема (V)');
```

```
  readln (T,V);
  a:=Sa∗sqr(R)∗exp(2.5∗ln(Tk))/Pk;
  b:=Sb∗R∗Tk/Pk;
  P:=R∗T/(V-b)-a/(sqrt(T)∗T∗(V+b));
  writeln('Давление газа равно P=', P:7:0);
END.
```

Таблица 2.2.1

Примеры оформления ввода данных с клавиатуры
и вывода результатов на экран.
(Переменные a, b, c, x – типа real, а i – integer)

Операторы	Вид на экране
Ввод данных	
1. Ввод одной или нескольких переменных без пояснения	
Readln(a); Readln(a,c,x);	2.3_ 2.3 4.6E-3 4_
2. Ввод одной переменной с пояснением в строчку	
Write('Imin='); Readln(Imin);	Imin=27_
3. Ввод нескольких переменных в строчку	
Write('Введите a,b,c: '); Readln(a,b,c);	Введите a,b,c: 8 3.123 4.44_
4. Ввод нескольких переменных в столбик	
Writeln(' a b c'); Readln(a,b,c);	a b c 8 3.765 2E-4_
Вывод данных	
1. Вывод одной переменной	
Writeln('a=',a:6:3)	a=–34.284
2. Вывод нескольких переменных в строчку	
Writeln('a=',a:6:3,' b=',b:10,' i=',i:3);	a=–34.284 b= 1.063E+02 i=127
3. Вывод нескольких переменных в столбик	
Writeln('a=',a:6:3); Writeln('b=',b:10); Writeln('i=',i:3);	a=–34.284 b= 1.063E+02 i=127
4. Вывод нескольких переменных в виде таблицы с заголовком	
Writeln(' a b i'); Writeln((a:6:3,' ',b:10,' ',i:3);	a b i –34.284 1.063E+02 127

2.3. Программирование разветвляющихся алгоритмов. Условный оператор

Алгоритм, в котором выполнение того или иного действия зависит от выполнения некоторого условия, называется разветвляющимся. Для программирования разветвляющихся алгоритмов в Паскале используется условный оператор.

Условный оператор используется в тех случаях, когда выполнение следующего действия в программе зависит от результатов предыдущих вычислений. Для записи условного оператора используются следующие служебные слова: **IF**, **THEN**, **ELSE**.

Общий вид условного оператора **IF** A **THEN** B; где A – логическое отношение, B – оператор, простой или составной. Для записи условного оператора перехода используются следующие логические отношения: = равно; < > не равно; < меньше; > больше; >= больше или равно; <= меньше или равно. Рассмотрим следующий условный оператор:

IF A < > 0 **THEN** B:=X/A;

В результате выполнения данного условного оператора B примет значение X/A только в том случае, если A не равно 0. Если A=0, то оператор присваивания B:=X/A; выполнен не будет.

Условный оператор может быть записан в следующем виде: **IF** A **THEN** ST1 **ELSE** ST2; где A – логическое отношение, а ST1 и ST2 – некоторые операторы. Если логическое отношение A – истина, то выполняется оператор ST1. Если логическое отношение A – ложь, то выполняется оператор ST2.

Пример 2.3.1

```
PROGRAM VVOD;
VAR n, C, d, b, g:real;
BEGIN write('введи n= ');read(n); c:=n+LN(N)+1;b:=c+1;
IF c < > 0 THEN IF b< > 10 THEN d:=1 ELSE d:=12.;g:=3;
writeln(C, d, g);
END.
```

Составной оператор. Если при некотором условии нужно выполнить последовательность операторов, то их объединяют в один составной оператор. Составной оператор начинается ключевым словом **BEGIN** и заканчивается ключевым словом **END**. Между этими словами помещаются составляющие операторы, которые выполняются в порядке их следования.

Рассмотрим пример составного условного оператора.

Пример 2.3.2

y=Cos 2x/5 при 5<=x<=10, r=sin 2x/10,
f= y+r; f=2.8 e^X при x<5 или x>10.

```
PROGRAM FUNK;
VAR x, y, r, f:real;
BEGIN write('введи x= ');read(x);
IF (x<=10) and (x>=5) THEN BEGIN
y:=Cos(2.0*x)/5.0; r:=Sin(2*x)/10.0;
f:=y+r; END ELSE
f:=2.8*exp(x);
write('значение f',f);
END.
```

Виды разветвляющихся алгоритмов и способы программирования приведены в табл. 2.3.1 и 2.3.2.

Пример 2.3.3. Составить программу для решения квадратного уравнения.

```
VAR A, B, C, X, X1, X2, D:real;
BEGIN
  {Ввод коэффициентов a,b,c.}
  writeln('Данная программа решает квадратные уравнения типа :');
  writeln(' A*X*X + B*X + C = 0 ')
  write('Введите коэффициент A = '); readln(A);
  write('Введите коэффициент B = '); readln(B);
  write('Введите коэффициент C = '); readln(C);
  {Расчет дискриминанта}
  D:=sqr(B)-4*A*C;
  {Расчет корней.}
  IF D < 0 THEN
  BEGIN
  writeln('Данное уравнение вещественных корней');
  writeln('НЕ ИМЕЕТ');
  END;
  IF D = 0 THEN
BEGIN
X:=-B/(2*A);
writeln('Данное уравнение имеет ОДИН вещественный корень');
writeln(' X = ',X:10:6);
END;
IF D > 0 THEN
```

```
BEGIN
X1 := (-B+sqrt(D))/(2*A);
X2 := (-B-sqrt(D))/(2*A);
writeln('Данное уравнение имеет ДВА вещественных корня');
writeln(' X1 = ',X1:10:6,' X2 = ',X2:10:6);
END;
END.
```

Пример 2.3.4. Составить программу вычисления теплового эффекта химической реакции, если теплоемкость зависит от температуры.

$$\Delta H_T = \left(\Delta H_{298} + \Delta C_P \cdot \left(T - 298\right)\right) \cdot n,$$

где n=7.3 моль – количество вещества; ΔH_{298}=63 200 Дж/моль – мольный эффект реакции при стандартных условиях.

$$\Delta C_P = \begin{cases} -13.7 Дж / моль \cdot К, T < 600К, \\ -4.9 Дж / моль \cdot К, 600 \le T < 800К, \\ 5.3 Дж / моль \cdot К, T \ge 800К. \end{cases}$$

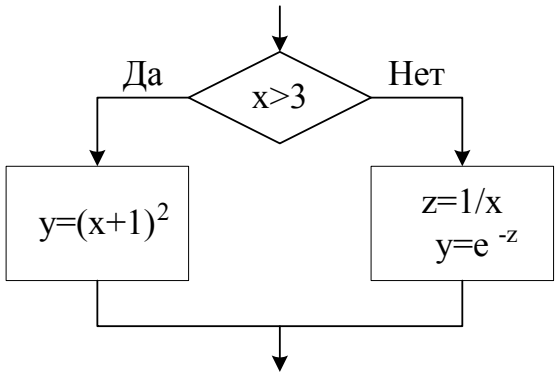
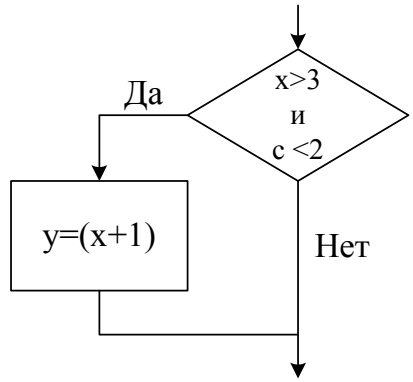
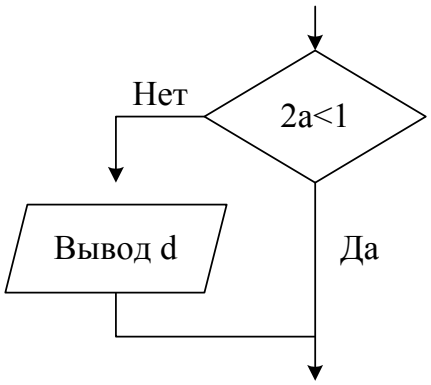
```
CONST
n=7.3; H298=63200; T1=600;T2=800;
Cp1= -13.7; Cp2= -4.9; Cp3=5.3;
VAR T, Cp, H:real;
BEGIN
write('Введите температуру, К : T = '); readln(T);
IF T < T1 THEN Cp:= Cp1 ELSE
IF T < T2 THEN Cp:= Cp2 ELSE Cp:= Cp3;
H:=(H298+Cp*(T-298))*n;
writeln('Тепловой эффект : H=', H:6:3);
END.
```

Пример 2.3.5. Вводятся значения длин трех отрезков. Составить программу определения возможности образования этими отрезками треугольника.

```
VAR a, b, c: real;
BEGIN
write('Введите длину первого отрезка a='); readln(a);
write('Введите длину второго отрезка b='); readln(b);
write('Введите длину третьего отрезка c='); readln(c);
IF (a<=0) OR (b<=0) OR (c<=0) THEN writeln('Вы неправильно ввели значения длин !') ELSE
IF (a+b>c) AND (a+c>b) AND (b+c>a) THEN writeln('Из этих отрезков МОЖЕТ быть образован треугольник !') ELSE writeln('Из этих отрезков НЕ МОЖЕТ быть образован треугольник !');
END.
```

Таблица 2.3.1

Виды разветвляющихся алгоритмов и способы их программирования

Фрагмент блок – схема	Фрагмент программы
а) Разветвление.  	**IF** x>3 **THEN** y:=sqr(x+1) **ELSE** **BEGIN** z:=1/x; y:=exp(-z) **END**;
б) Обход. (1-й случай).  	**IF** (x>3) **AND** (c<2) **THEN** d:=x+1;
в) Обход. (2-й случай).  	Варианты программирования. 1) **IF NOT** (2∗a>1) **THEN** writeln(’d=’,d:10); 2) **IF** 2∗a<=1 **THEN** writeln(’d=’,d:10); 3) **IF** 2∗a>1 **THEN ELSE** writeln(’d=’,d:10); 4) **IF** 2∗a>1 **THEN GOTO** 1; writeln(’d=’,d:10); 1:……

Два способа расчета функции с условиями (табл. 2.3.2):

$$f(x)=\begin{cases} x^2-a, x<0, \\ 4x+\sqrt{x}, 0\le x<a, \\ \ln\left|\dfrac{x}{a}\right|, a\le x. \end{cases}$$

Таблица 2.3.2

Способ 1	Способ 2
. . . **IF** x<0 **THEN** f:=sqr(x)-a **ELSE** **IF** x<a **THEN** f:=4∗x+sqrt(x) **ELSE** f:=ln(abs(x/a)); . . .	. . . **IF** x<0 **THEN** f:=sqr(x)-a; **IF** (0<=x) **AND** (x<a) **THEN** f:=4∗x+sqrt(x); **IF** a<=x **THEN** f:=ln(abs(x/a)); . . .

2.4. Оператор варианта

Оператор варианта (**CASE** ... **OF**) предназначен для программирования алгоритмов с множественным выбором.

Пример 2.4.1. Составить программу для вывода названий месяцев по их номеру.

```
VAR N:integer;
BEGIN
write('Введите номер месяца : '); readln(N);
CASE N OF
  1: writeln(' ЯНВАРЬ');
  2: writeln(' ФЕВРАЛЬ');
  3: writeln(' МАРТ');
  4: writeln(' АПРЕЛЬ');
  5: writeln(' МАЙ');
  6: writeln(' ИЮНЬ');
  7: writeln(' ИЮЛЬ');
  8: writeln(' АВГУСТ');
  9: writeln(' СЕНТЯБРЬ');
  10: writeln(' ОКТЯБРЬ');
  11: writeln(' НОЯБРЬ');
  12: writeln(' ДЕКАБРЬ');
  ELSE writeln('Вы неправильно ввели номер месяца ! ');
END.
```

Пример 2.4.2. Составить программу вычисления сумм арифметической и геометрической прогрессий, а также определения значения заданного члена прогрессии.

Для арифметической прогрессии:

$$A_N = A_1 + d \cdot (N-1)$$

$$S_N = \frac{A_1 + A_N}{2} \cdot N$$

Для геометрической прогрессии:

$$B_N = B_1 \cdot q^{N-1}$$

$$S_N = \frac{B_N \cdot q - B_1}{q - 1}$$

```
VAR V, N:integer;
A1, B1, AN, BN, SA, SB, d, q:real;
BEGIN
writeln('Выберите вид прогрессии:');
writeln(' 1- арифметическая');
writeln(' 2- геометрическая');
readln(V);
CASE V OF
1: BEGIN
writeln('Вы выбрали арифметическую прогрессию.');
write('Введите первый член прогрессии A1 ='); readln(A1);
write('Введите разность прогрессии d ='); readln(d);
write('Введите номер N ='); readln(N);
AN:=A1+d*(N-1);
SA:=(A1+AN)*N/2;
writeln(N, '-й член прогрессии равен =',AN:6:3);
writeln('Сумма первых ',N,' членов прогрессии равна =',SA:6:3);
END;
2: BEGIN
writeln('Вы выбрали геометрическую прогрессию.');
write('Введите первый член прогрессии B1 ='); readln(B1);
write('Введите знаменатель прогрессии q ='); readln(q);
write('Введите номер N ='); readln(N);
BN:=B1*exp((N-1)*ln(q));
SB:=(BN*q-B1)/(q-1);
writeln(N, '-й член прогрессии равен =',BN:6:3);
writeln('Сумма первых ',N,' членов прогрессии равна =',SB:6:3);
END;
END.
```

2.5. Программирование циклических алгоритмов

Алгоритмы, в которых действия повторяются многократно, называются циклическими. Виды циклических алгоритмов и способы их программирования приведены в табл. 2.5.1. Часто используются циклы для расчета сумм, произведений, количества. Виды сумм и способы их

расчета приведены в табл. 2.5.2. Примеры программ с циклическими алгоритмами приведены также в следующих разделах: 6, 7, 8.

В языке Паскаль существуют три оператора цикла: **FOR … TO … DO, WHILE … DO, REPEAT … UNTIL.**

Общий вид оператора **WHILE** A **DO** ST, где A – логическое выражение, ST – оператор, простой или составной.

WHILE X< >0 **DO BEGIN** G:= C+1/X; x:=x-1 **END**; В данном примере вычисляется логическое выражение типа X< >0.

Если оно «истина», то будут выполняться операторы G:= 1/X; X:=X-1; и управление опять будет передаваться вновь на проверку выражения X< >0. Как только условие X<>0, будет «ложь», будет выполняться оператор, следующий за **END**. То есть цикл повторяется пока X< >0.

Оператор цикла **FOR**:

FOR i:=n1 **TO** n2 **DO** ST;

где i – переменная цикла; n1 - начальное значение переменной; n2 – конечное значение переменой; ST – оператор, простой или составной.

FOR i:= 1 **TO** 20 **DO** a:=a+1; При такой записи будет выполняться оператор A:=A+1; пока i поочередно принимает значения i = 1, 2, …, 20.

В операторе **FOR** шаг изменения параметра цикла равен 1, однако имеется разновидность цикла **For** или цикл по убывающим значениям параметра цикла, в этом случае параметр цикла изменяет свое значение с шагом -1 от N2 до N1 : **FOR** I:= n2 **DOWNTO** n1 **DO** ST; , **FOR** i:= 20 **DOWNTO** n2 **DO** A:=A+1;

Оператор с постпроверкой условия имеет следующий вид: **REPEAT** ... **UNTIL**, где **REPEAT** < операторы > **UNTIL** < условие >. Операторы выполняются хотя бы один раз, после чего вычисляется условие. Если значение условия есть FALSE, операторы повторяются, в противном случае оператор завершает свою работу.

Однако цикл может быть организован с помощью условного оператора и оператора безусловного перехода. Метки. Для указания последовательности выполнения программ используют метки. Метка может состоять из букв и цифр, но первой должна стоять буква. В качестве исключения метка может состоять только из цифр, в диапазоне от 0 до 9999. Все метки, используемые в программе, должны быть обозначены с помощью служебного слова **LABEL**. (Например, label c1, c2;).

Оператор безусловного перехода имеет следующий общий вид:

GOTO N,

где N – имя метки; **GOTO** – служебное слово.

Метка N описывается с помощью служебного слова **LABEL** и ставится перед следующим оператором, который должен быть выполнен.

Пример 2.5.1

```
PROGRAM Met;
LABEL d1;
VAR p, y, x:real;
BEGIN
read(p);
GOTO d1;
Y:=ln(p)+1;
d1:X:= sqr(p);
write(x);
END.
```

В результате выполнения данной программы будет вычислена величина X. Значение У останется незаданным, так как управление в программе передается на оператор с меткой d1.

Пример 2.5.2. Составить программу для определения гидравлического сопротивления слоя насадки при изменении скорости газа от 10 до 15 м/с с шагом h=0.5 м/с.

$$\Delta P = \lambda \cdot \frac{H}{d_э} \cdot \frac{w_г^2 \cdot \rho}{2}$$

где H – высота слоя, м,

$d_э$ – эквивалентный диаметр, м,

ρ – плотность газа, кг/м^3,

$w_г$ – скорость газа, м/с.

Коэффициент гидравлического сопротивления рассчитывается по формуле:

$$\lambda = \begin{cases} 140/\mathrm{Re}, \mathrm{Re} < 40 \\ 16/\mathrm{Re}^{0.2}, \mathrm{Re} \geq 40 \end{cases}$$

где Re – критерий Рейнольдса – равен:

$$\mathrm{Re} = \frac{w_г \cdot d_э \cdot \rho}{\mu}.$$

μ – динамическая вязкость газа, Па·с.

Рассмотрим пример решения задачи с использованием оператора цикла **WHILE** ... **DO**.

```
VAR dP, L, H, d, w, p, Re, m, wn, wk, hw:real;
BEGIN
H:=1; d:=0.005; p:=1.35; m:=2.45E-6;
writeln(' wn wk hw');
readln(wn,wk,hw);
writeln(' w Re L dP');
w:=wn;
WHILE w<=wk DO
BEGIN
Re:=w*d*p/m;
IF Re<40 THEN L:=140/Re ELSE L:=16/exp(0.2*ln(Re));
dP:=L*H/d*sqr(w)*p/2;
writeln(w:5:2,' ',Re:10,' ',L:10,' ',dP:10);
w:=w+wh;
END;
END.
```

Рассмотрим пример решения задачи с использованием оператора цикла **REPEAT** ... **UNTIL**.

```
VAR dP, L, H, d, w, p, Re, m, wn, wk, hw:real;
BEGIN
H:=1;
d:=0.005;
p:=1.35;
m:=2.45E-6;
writeln(' wn wk hw');
readln(wn,wk,hw);
writeln(' w Re L dP');
w:=wn;
REPEAT
Re:=w*d*p/m;
IF Re<40 THEN L:=140/Re ELSE L:=16/exp(0.2*ln(Re));
dP:=L*H/d*sqr(w)*p/2;
writeln(w:5:2,' ',Re:10,' ',L:10,' ',dP:10);
w:=w+wh
UNTIL w>wk
END.
```

Пример 2.5.3. Составить программу для расчета удельной теплоемкости метана по эмпирической зависимости:

$$\Delta C_P = a + b \cdot T + c \cdot T^2 + \frac{d}{T^2} + e \cdot T^3,$$

где a, b, c, d, e – эмпирические коэффициенты: $a = 4.171$; $b = 14.45\times10^{-3}$; $c = 0.267\times10^{-6}$; $d = 0.0$; $e = -1.722\times10^{-9}$.

Произвести расчеты для интервала температур от 0 до 500 К, с шагом 50 К.

```
LABLE 1;
CONST a=4.171;
b=14.45E-03;
c=0.267E-06;
d=0.0;
e=-1.722E-9;
VAR T, T1, T2, dT, Cp:real;
BEGIN
T1:=0;
T2:=500;
dT:=50;
T:=T1;
writeln('T,K Cp, Дж/мольK');
1: Cp:=a+b*T+c*sqr(T)+d/sqr(T)+e*exp(3*ln(T));
writeln(T:3:0,' ',Cp :9:4);
T:=T+dT;
IF T<T2 THEN GOTO 1;
writeln('Расчет окончен ! ');
END.
```

Пример 2.5.4. Составить программу для расчета выражения:

$$F = \sqrt[3]{\sum_{i=1}^{11} x^i} + \prod_{k=1}^{20} \sqrt[k]{x}.$$

```
VAR F, S, P, x:real; i, k:integer;
BEGIN
write('x='); readln(x);
S:=0;
FOR i:=1 TO 11 DO
```

```
S:=S+exp(i*ln(x));
P:=1;
FOR k:=1 TO 20 DO
P:=P*exp(1/k*ln(x));
F:=exp(1/3*ln(S))+P;
writeln('S=',S:10,' P=',P:10,' F=',F:10)
END.
```

Пример 2.5.5. Составить программу вычисления бесконечного ряда:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots .$$

Условие окончания расчета ряда: $\frac{x^{n+1}}{(n+1)!} < \varepsilon$, ($\varepsilon = 0.0001$).

```
CONST e=0.0001;
VAR i:integer; X, S, Y:real;
BEGIN
write('Введите значение X ='); readln(X);
i:=0;                {Номер слагаемого.}
y:=1;                {Значение i-го слагаемого.}
S:=0;                {Сумма ряда.}
REPEAT
S:=S+y;          {Расчет суммы.}
i:=i+1;
y:=y*x/i         {Расчет следующего слагаемого.}
UNTIL y<e;
writeln('Сумма бесконечного ряда =', S:10:6);
END.
```

Пример 2.5.6

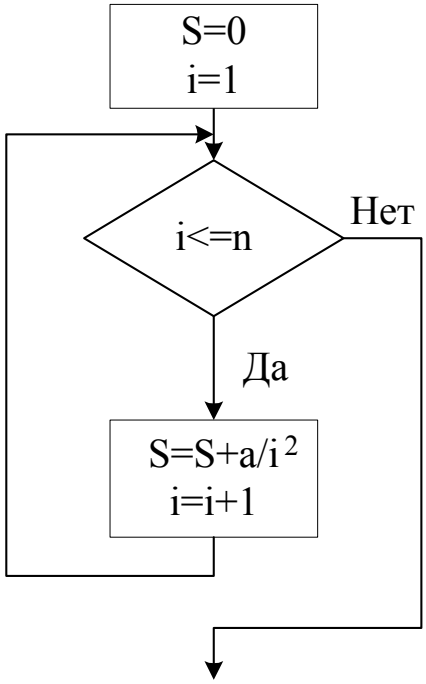
```
PROGRAM sum;
VAR i,n:integer; s:real;
BEGIN
write('укажите число n ');
readln(n); s:=0; FOR i:=n DOWNTO 1 DO s:=s+i;
writeln('значение суммы ',s);
END.
```

Таблица 2.5.1

Способы программирования циклов различной структуры

(на примере расчета суммы $S = \sum_{i=1}^{n} \frac{a}{i^2}$ *).*

Фрагмент блок-схемы	Операторы

1. Цикл с предварительным условием
 
 	
	а) С использованием
	цикла **WHILE** ... **DO**.
	S:=0; i:=1;
	WHILE i<=n **DO**
	BEGIN
	S:=S+a/sqr(i);
	i:=i+1
	END;
	
	б) С использованием
	операторов **IF** ... **THEN** и **GOTO**.
	S:=0; i:=1;
	1: **IF** i<=n **THEN**
	BEGIN
	S:=S+a/sqr(i);
	i:=i+1;
	GOTO 1
	END;

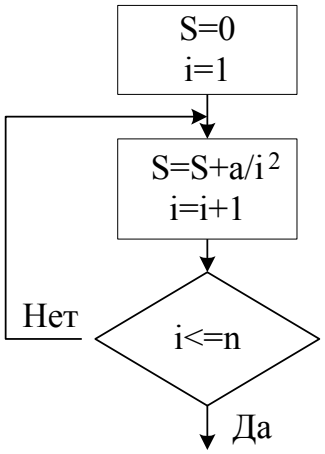
2. Цикл с последующим условием  	а) С использованием цикла **REPEAT** ... **UNTIL** S:=0; i:=1; **REPEAT** S:=S+a/sqr(i); i:=i+1 **UNTIL** i>n; б) С использованием операторов **IF** ... **THEN** и **GOTO**. S:=0; i:=1; 1: S:=S+a/sqr(i); i:=i+1 **IF** i<=n **THEN GOTO** 1;

Таблица 2.5.2

Расчет сумм (произведений, количества)

Вид суммы	Фрагмент программы
1. «Простая» сумма $S = \sum_{i=1}^{n} \frac{1}{i^2}$	S:=0; **FOR** i:=1 **TO** N **DO** S:=S+1/sqr(i);
2. Сумма «с условием» $S = \sum_{\substack{k=3 \\ 2i>10}}^{15} \left(-4+2\cdot i\right)^2$	S:=0; **FOR** k:=3 **TO** 15 **DO** **IF** 2∗i>10 **THEN** S:=S+sqr(-4+2∗i);

3. Расчет нескольких сумм
$S_1 = \sum_{i=1}^{n} i^2 \quad S_2 = \sum_{i=1}^{n} \sqrt{i}$
	S1:=0; S2:=0;
	FOR i:=1 **TO** n **DO**
	BEGIN
	S1:=S1+sqr(i); S2:=S2+sqrt(i)
	END;

4. «Сумма в сумме»
а)
$S = \sum_{i=1}^{5} \sum_{j=1}^{6} e^{j-i}$
	а) S:=0;
	FOR i:=1 **TO** 5 **DO**
	FOR j:=1 **TO** 6 **DO**
	S:=S+exp(j-i);

б)
$S = \sum_{i=1}^{n} \left(\ln i \cdot \sum_{j=1}^{m} ij \right)$
	б) S:=0;
	FOR i:=1 **TO** n **DO**
	BEGIN
	S2:=0;
	FOR j:=1 **TO** m **DO** S2:=S2+i∗j;
	S:=S+ln(i)∗S2;
	END;

5. Сумма с реккурентной формулой
$S = \sum_{i=1}^{10} a_i^2$,
где $a_i=2*a_{i-1}$, $a_1=1$.
	a:=1; S:=0;
	FOR i:=1 **TO** 10 **DO**
	BEGIN
	S:=S+sqr(a); a:=2∗a;
	END;

Расчет произведения аналогичен расчету суммы, только до цикла пишется оператор P:=1, а в цикле – выражение вида P:=P∗<выражение, стоящее под знаком произведения>. Расчет количества – это расчет суммы вида $K = \sum 1$, поэтому до цикла пишется оператор K:=0, а в цикле - K:=K+1.	

2.6. Одномерные массивы

Массив – это упорядоченный набор фиксированного количества данных одного типа. В Паскале над переменной типа «массив» определено лишь две операции: присваивания и сравнения, поэтому при работе с массивом требуемую операцию следует осуществить над каждым элементом массива.

Основные действия над массивами

1. Формирование массива:

а. Вводом с клавиатуры:

```
FOR i:=1 TO N DO BEGIN write('C[',i,']='); readln(C[i]) END;
```

б. Расчет массива по формуле (например, $C_i = 2^{-i}$):

```
FOR i:=1 TO N DO C[i]:=exp(-i*ln(2));
```

в. Задание массива с помощью функции **RANDOM**:

```
randomize; {пишется перед первым употреблением RANDOM}
 ...
FOR i:=1 TO N DO C[i]:=RANDOM;
```

2. Вывод массива на экран.

```
FOR i:=1 TO N DO writeln('C[',i,']=',C[i]:12);
```

3. Расчет последовательностей ($C_i = 4C_{i-1} + 17.3$):

```
C[1]:=3.6; FOR i:=2 TO N DO C[i]:=4*C[i-1]+17.3;
```

4. Нахождение минимального (или максимального) элемента.

```
Cmin:=C[1]; FOR i:=2 TO N DO IF C[i]<Cmin THEN Cmin:=C[i];
```

5. Нахождение номера минимального (или максимального) элемента:

```
Nmin:=1; FOR i:=2 TO N DO IF C[i]<C[Nmin] THEN Nmin:=i;
```

6. Нахождение первого (последнего) элемента массива с заданным условием (например, $x_i<0$):

```
k:=0; REPEAT k:=k+1 UNTIL X[k]<0;       (поиск 1-го элемента)
k:=N+1; REPEAT k:=k-1 UNTIL X[k]<0;   (поиск последнего элемента)
```

7. Перестановка местами элементов массива с номерами i и j:

```
r:=C[i]; C[i]:=C[j]; C[j]:=r;
```

8. Нахождение количества элементов массива, удовлетворяющих условию (например, $2<x_i\le5$):

```
M:=0; FOR i:=1 TO N DO IF (2<X[i]) AND (X[i]<=5) THEN M:=M+1;
```

9. Способ задания массива в разделе **CONST**:

```
  CONST d:ARRAY[1..4] OF real=(1.2, 3.6, 2e-1, 12.34);
```

Пример 2.6.1. Составить программу определения минимального элемента массива A(10), суммы элементов, стоящих до него и произведения положительных элементов после него.

```
VAR A:ARRAY[1..10] OF real;
i, MinN:integer;
Min, S, P:real;
BEGIN
{Ввод массива A(10)}
FOR i:=1 TO 10 DO
BEGIN
write('Введите ',i,' элемент массива:'); readln(A[i]);
END;
{Поиск номера минимального элемента в массиве A(10)}
Min:=A[1]; MinN:=1;
FOR i:=2 TO 10 DO
IF Min>A[i] THEN
BEGIN Min:=A[i]; MinN:=i; END;
writeln('Минимальный элемент данного массива =', Min);
{Расчет суммы и произведения}
S:=0; P:=1;
FOR i:=1 TO MinN-1 DO S:=S+A[i];
FOR i:=MinN+1 TO 10 DO
IF A[i]>0 THEN P:=P*A[i];
writeln('Сумма =', S:10:6);
writeln('Произведение =', P:10:6);
END.
```

Пример 2.6.2. Найти критерий линейной корреляции по формуле

$$k = \frac{\sum_{i=1}^{n} (y_i - \bar{y})(x_i - \bar{x})}{\sum_{i=1}^{n} (y_i - \bar{y})^2 \sum_{i=1}^{n} (x_i - \bar{x})^2}$$

где $\bar{x} = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i$, $\bar{y} = \frac{1}{n} \cdot \sum_{i=1}^{n} y_i$.

```
VAR x, y:ARRAY[1..100] OF real;
Xs, Ys, Sxy, Sx2, Sy2, k:real;
i, n:integer;
BEGIN
  {Ввод массивов x и y с клавиатуры.}
write('Введите количество элементов в массиве. n=');
```

```
readln(n);
FOR i:=1 TO n DO
BEGIN
write('x',i, '='); readln(x[i]);
write('y',i, '='); readln(y[i])
END;
{Расчет средних значений массивов x и y}
Xs:=0; Ys:=0;
FOR i:=1 TO n DO
BEGIN
Xs:=Xs+x[i]/n;
Ys:=Ys+y[i]/n
END;
{Расчет коэффициента линейной корреляции}
Sxy:=0; Sx2:=0; Sy2:=0;
FOR i:=1 TO n DO
BEGIN
Sxy:=Sxy+(y[i]-Ys)*(x[i]-Xs);
Sx2:=Sx2+sqr(x[i]-Xs);
Sy2:=Sy2+sqr(y[i]-Ys);
END;
k:=Sxy/(Sy2*Sx2);
{Вывод результата}
writeln('Коэффициент линейной корреляции. k=',k:8:6)
END.
```

Пример 2.6.3. Составить программу случайного задания массива F(20), определения факториала от числа положительных элементов данного массива, а все отрицательные элементы массива заменить их квадратами.

```
VAR F:array [1..20] of real;
i, N:integer;
P:real;
BEGIN
  {Задание массива F(20) случайными числами из интервала [-10,10].}
randomize; FOR i:=1 TO 20 DO F[i]:=(1-2*RANDOM)*10;
{Определение количества положительных элементов.}
N:=0;
FOR i:=1 TO 20 DO
IF F[i]>0 THEN N:=N+1;
  {Расчет факториала. N!}
```

```
P:=1; FOR i:=1 TO N DO P:=P*i;
{Замена отрицательных элементов массива их квадратами.}
FOR i:=1 TO 20 DO
IF F[i]<0 THEN F[i]:= F[i]*F[i];
  {Вывод результатов}
  writeln('N!=',P:10);
  FOR i:=1 TO 20 DO writeln('F',i,'=',F[i]:7:4);
END.
```

Пример 2.6.4. Составить программу для расчета функции по формуле Лагранжа

$$y = \sum_{i=1}^{n} \left(y_i \cdot \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j} \right)$$

```
CONST
{Задание значений массивов в программе в виде типизированных констант.}
n=5;
x:ARRAY[1..n] OF real=(1,3,5,7,9);
y:ARRAY[1..n] OF real=(1.33,2.57,3.44,4.74,5.11);
VAR x0, S, P:real;
i, j:integer;
BEGIN
{Расчет суммы.}
S:=0;
FOR i:=1 TO n DO
BEGIN
{Расчет произведения.}
P:=1;
FOR j:=1 TO n DO
IF i< >j THEN P:=P*(x0-x[j])/(x[i]-x[j]);
S:=S+y[i]*P
END;
writeln('f(',x0:5:2,')=',S:12)
END.
```

Пример 2.6.5. По древней легенде мудрец, придумавший игру шахматы, попросил в награду у султана столько зерен пшеницы, сколько уместиться на шахматной доске, если заполнять ее следующим образом: на первую клетку доски – одно зерно, на вторую – два, на 3-ю – 4, на 4-ю – 8 и т. д. Составить программу для расчета количества зерен.

```
PROGRAM Chess;
VAR i:integer; k, S:real;
BEGIN
k:=1; S:=0;
FOR i:=1 TO 64 DO
BEGIN
S:=S+k;
k:=2*k;
END;
writeln('S=',S:12)
END.
```

Пример 2.6.6

```
PROGRAM Xmimax;
VAR x:ARRAY [1..10] of real;
i, n:integer;
xmax, xmin:real;
BEGIN
write('укажите число элементов '); readln(n);
i:=1; xmax:=10.e-37;xmin:=10.e37;
WHILE i<=n DO
BEGIN
write('введи ',i, 'значение'); readln(x[i]);
IF x[i] <= xmin THEN xmin:=x[i];
IF x[i] >= xmax THEN xmax:=x[i];i:=i+1;
END;
writeln('значение xmin',xmin,'xmax',xmax) ;
END.
```

Пример 2.6.7

```
PROGRAM Xsum;
VAR x:ARRAY [1..100] of real;
i, n:integer;
s:real;
BEGIN
write('укажите число элементов '); readln(n);
i:=1;s:=0;
WHILE i<=n DO
BEGIN
```

```
write('введи ',i, 'значение'); readln(x[i]);
s:=s+x[i]; i:=i+1;
END;
writeln('значение суммы ',s) ;
END.
```

Пример 2.6.8

```
PROGRAM sum;
VAR i, n:integer;
s:real;
BEGIN
write('укажите число n '); readln(n);
i:=1;s:=0;
WHILE i<=n DO
BEGIN
s:=s+1/i; i:=i+1;
END;
writeln('значение суммы ',s) ;
END.
```

2.7. Матрицы

Пример 2.7.1. Составить программу определения максимального элемента в случайно заданном массиве A[10,10] и посчитать сумму элементов данного массива, значения которых по абсолютной величине не превышает значение 0,5.

```
VAR i, j:integer;
A: ARRAY [1..10, 1..10] OF real;
MAX, S:real;
BEGIN
{Ввод матрицы с клавиатуры}
randomize;
FOR i:=1 TO 10 DO
FOR j:=1 TO 10 DO
BEGIN
write('A[',i, ',' ,j,']=');
readln(A[i,j])
END;
{Нахождение минимального элемента в матрице A(10,10)}
MAX:=A[1,1];
```

```
FOR i:=1 TO 10 DO
FOR j:=1 TO 10 DO
IF MAX<A[i, j] THEN MAX:=A[i, j];
{Нахождение суммы элементов |a(i,j)|<0.5}
S:=0;
FOR i:=1 TO 10 DO
FOR j:=1 TO 10 DO
IF abs(A[i, j])<=0.5 THEN S:=S+A[i, j];
{Вывод результатов}
writeln('Максимальный элемент данного массива =', MAX:10:6);
writeln('Сумма =', S:10:6);
END.
```

Пример 2.7.2. Составить программу определения суммы элементов главной диагонали в массиве A[3,3].

```
CONST
A:ARRAY [1..3,1..3] OF real=( (2,4,6.7), (1.3, 0.11,2E-3), (6,2,7) );
VAR i, j:integer;
A: ARRAY [1..10, 1..10] OF real;
S:real;
BEGIN
S:=0;
FOR i:=1 TO 10 DO S:=S+A[i, i];
writeln('Сумма элементов главной диагонали матрицы =', S:10:6);
END.
```

Пример 2.7.3. Составить программу определения номера строки в случайно заданном массиве A[7,10], сумма элементов которой максимальна.

```
VAR i, j, Nmax:integer;
A: ARRAY [1..7, 1..10] OF real;
S: ARRAY [1..7] OF real;
BEGIN
randomize;
FOR i:=1 TO 7 DO
FOR j:=1 TO 10 DO
A[i, j]:=(1-2*random)*10;
{Нахождение суммы элементов каждой строки}
FOR i:=1 TO 7 DO
BEGIN
S[i]:=0;
```

```
FOR j:=1 TO 10 DO
S[i]:=S[i]+A[i, j];
END;
  {Нахождение номера максимального элемента в массиве S(7)}
Nmax:=1;
FOR i:=1 TO 7 DO
IF S[Nmax]<S[i] THEN Nmax:=i;
{Вывод результатов}
writeln('Максимальная сумма строки =', MAX:10:6);
END.
```

2.8. Файлы

Пример 2.8.1. Рассчитать массив P(6) по формуле:

$$P_i = a \cdot C_i + b + \frac{1}{e} \cdot \sum_{j=1}^{3} d_j^2 .$$

Исходные данные ввести из файла 'data.txt', который представляет собой следующее (буквы рядом с числами указывают, какие числа каким переменным соответствуют). Результаты записать в файл 'data.res'.

2.73E-4^{A} 0.83^{B}
1 5 -6 4 5 -1$^{C(6)}$
7.43
3.51 $^{D(3)}$
-0.14
1.23^{E}

```
VAR p:ARRAY [1..6] OF real;
c:ARRAY [1..6] OF integer;
d:ARRAY [1..3] OF real;
i, j:integer;
a, b, e, s:real;
f:text;
BEGIN
  Assign(F,'data.txt');
  Reset(F);
  readln(F,a,b);
  FOR j:=1 TO 6 DO read(F,c[j]);
  readln(F);
  FOR i:=1 TO 3 DO readln(F,d[i]);
  readln(F,e);
  Close(F);
```

```
  S:=0;
  FOR j:=1 TO 3 DO S:=S+sqr(d[j]);
  Assign(F,'data.res');
  Rewrite(F);
  FOR i:=1 TO 6 DO
  BEGIN
    P[i]:=a*C[i]+b+S/e;
    writeln(F,'P[',i,']=',P[i]:10)
  END;
  Close(F)
END.
```

Считывание массива из файла

1. Считывание массива из файла. Количество элементов известно.

```
Assign (F , 'data.txt');
Reset (F);
FOR i:=1 TO N DO readln(F, x[i]);
Close (F);
```

Примечание. Данные в файле data.txt записаны в столбик. Например,

```
1.00
2.47
...
4.89
```

2. Считывание массива из файла. Количество элементов неизвестно.

```
Assign (F, 'data.txt');
Reset (F);
N:=0;
WHILE NOT (seekeof (F)) do
BEGIN
N:=N+1;
readln (F, x[N] , y[N]);
END;
Close (F);
```

3. Запись массива в файл.

```
Assign (F , 'data.res');
Rewrite (F);
FOR i:=1 TO N DO
writeln(F, 'X[' , i , ']=' , x[i]:5:2);
Close (F);
```

4. Считывание матрицы из файла.

```
Assign (F , ’data.txt’);
Reset (F);
FOR i:=1 TO N DO
BEGIN
FOR j:=1 TO M DO read(F,a[i,j]);
readln(F)
END;
Close (F);
```

Пример 2.8.2. Рассчитать массовые доли компонентов газовой смеси:

$$\overline{x}_i = \frac{x_i \cdot M_i}{\sum_{i=1}^{n} x_i \cdot M_i},$$

если известны мольные доли x(n) и молярные массы m(n) компонентов.

```
VAR x, xm, m:ARRAY[1..20] OF real;
S:real;
i:integer;
C:text;
BEGIN
  write(’n=’); readln(n);
  {Ввод массивов x(n) и m(n) из файла.}
  Assign(C,’data.txt’);
  Reset(C);
  FOR i:=1 TO n DO readln(C,x[i],m[i]);
  Close(C);
  {Расчет суммы.}
  S:=0;
  FOR i:=1 TO n DO
  S:=S+x[i]*m[i];
  {Расчет массива xm(n) с одновременным выводом результатов.}
  FOR i:=1 TO n DO
  BEGIN
    xm[i]:=x[i]*m[i]/S;
    writeln(’x[’,i,’]=’,x[i]:6:3,’ xm[’,i,’]=’,xm[i]:6:3)
  END;
END.
```

Пример 2.8.3. Найти среднюю молярную массу фракции нефти по формуле

$$\overline{m} = \frac{\sum\limits_{T_n<T_i<T_k} x_i \cdot m_i}{\sum\limits_{T_n<T_i<T_k} x_i},$$

(т. е. найти среднюю молярную массу тех компонентов нефти, температура кипения которых попадает в температурный интервал выкипания фракции нефти [Tn,Tk]). Данная задача сводится к нахождению сумм «с условием».

```
VAR x, m, T:ARRAY[1..100] OF real;
Tn, Tk, S, Sm, Ms:real; F:text; i, N:integer;
BEGIN
{Ввод с клавиатуры значений Tn и Tk.}
write('Температура начала кипения фракции. Tn=');
readln(Tn);
write('Температура окончания кипения фракции.Tk=');
readln(Tk);
{Ввод из файла значений массивов x,m,T. Количество элементов в массиве заранее неизвестно.}
Assign(F, 'Fuel.dat');
Reset(F); N:=0;
WHILE not(seekeof(F)) DO
BEGIN
N:=N+1;
readln(F,x[N],m[N],T[N]);
END;
Close(F);
{Расчет средней молярной массы для веществ с температурами кипения Tn<Ti<Tk.}
S:=0; Sm:=0;
FOR i:=1 TO N DO
IF (Tn<T[i]) AND (T[i]<Tk) THEN
BEGIN
S:=S+x[i]; Sm:=Sm+x[i]*m[i];
END;
Ms:=Sm/S;
{Вывод результата.}
writeln('Средняя молярная масса. Ms=',Ms:6:2,' г/моль')
END.
```

3. ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ И МОДУЛИ

3.1. Подпрограммы

Различают глобальные и локальные параметры. Глобальные параметры являются общими на две или более программных единиц. Локальные параметры изменяют свои значения только внутри данной программной единицы. Как видно из примера, приведенного ниже, писание параметров в блоке **VAR** означает, что данная группа параметров является глобальными для входных и выходных значений (x_1, x_2). При отсутствии в заголовке служебного слова **VAR** значения параметров передаются внутрь процедуры, но не передаются из нее в главную программу (a,b,c).

В тех случаях, когда параметрами процедур и функций являются массивы, необходимо размерность и размер этих массивов задавать через служебное слово **TYPE**. Ниже приведен пример составления процедуры maxmin для определения максимального и минимального значений двумерного массива.

Пример 3.1.1. Составить программу для расчета выражения

$$k = \lg\frac{ax^2\sqrt{x}}{3+x} + 2\cdot 10^3\left(1+2x\right)^{1-\sqrt{a}},$$

оформив определение значений нестандартных функций в виде подпрограмм.

```
VAR k, a, x:real;
FUNCTION lg(x:real):real;
BEGIN
lg:=ln(x)/ln(10)
END;
FUNCTION pow(a,x:real):real;
BEGIN
pow:=exp(a*ln(x))
END;
BEGIN
  write('Введите a,x:');
  readln(a,x);
  k:=lg(a*sqr(x)*sqrt(x)/(3+x))+2E3*pow(1-sqrt(a),1+2*x);
  writeln('k=',k:10)
END.
```

Пример 3.1.2. Составить программу расчета массива Z(10)

$$Z_i = \frac{f^2(x)}{\sin c} + \frac{\sqrt{x_i}}{S^2 \cdot q}.$$

Расчет функции *f(x)* оформить в виде подпрограммы – функции

$$f(x) = \begin{cases} 1/x, x > 3, \\ 1 + 2 \cdot x^3, x \le 3. \end{cases}$$

Расчет суммы – в виде подпрограммы – процедуры

$$S = \sum_{i=1}^{15} \sqrt{x_i}\ .$$

```
TYPE mas=ARRAY [1..10] OF real;
VAR x, z:mas; c, q, S:real;
FUNCTION F(x:real):real;
BEGIN
  IF x>3 THEN f:=1/x ELSE f:=1+2*x*sqr(x);
END;
PROCEDURE Sum(x:mas;VAR S:real);
  VAR i:integer;
BEGIN
  S:=0;
  FOR i:=1 TO 10 DO S:=S+sqrt(x[i]);
END;
BEGIN
  writeln(' c q');
  readln(c,q);
  FOR i:=1 TO 10 DO
  BEGIN
    write('X[',i,']=');
    readln(x[i])
  END;
  Sum(x,S);
  FOR i:=1 TO 10 DO
  BEGIN
  P[i]:=sqr(f(x[i]))/sin(c)+sqrt(x[i])/(sqr(S)*q);
    writeln('P[',i,']=',P[i]:10);
  END;
END.
```

Пример 3.1.3. Составить программу для расчета выражения

$$F = \sum_{i=3}^{8} x_i^2 + \sum_{k=2}^{10} y_k^3\ .$$

```
TYPE mas=ARRAY [1..20] OF real;
VAR x, y:mas;  f:real; i,k:integer;
FUNCTION Sum(a:mas; n1,n2,p:integer):real;
VAR S:real; i:integer;
BEGIN S:=0;
  FOR i:=n1 TO n2 DO S:=S+exp(p*ln(a[i]));
  Sum:=S
END;
BEGIN
  {Ввод массивов x и y с клавиатуры.}
FOR i:=1 TO 8 DO
BEGIN
write('x',i, '='); readln(x[i]);
END;
FOR k:=1 TO 10 DO
BEGIN
write('y', k, '='); readln(y[k]);
END;
  F:=Sum(x,3,8,2)+Sum(y,2,10,3);
  writeln('F=',F:10)
END.
```

Пример 3.1.4

```
PROGRAM Stepen;
VAR x,y:real;
f1,f2:text;
FUNCTION POWER(a,b:real):real;
BEGIN
IF a > 0 THEN power:=exp(b*ln(a))
ELSE
IF a < 0 THEN power:=exp(b*ln(abs(a)))
ELSE
IF b=0 then power:= 1
ELSE
power:=0;
END;
BEGIN
assign(f2,'agg.rez'); reset(f2);
read(f2,x,y);
assign(f1,'q'); rewrite(f1);
write(power(x,y):4:1);
```

```
close(f1);
END.
```

Для обращения к функции Power мы просто указали ее в качестве параметра при выводе данных. Параметры X и Y в момент обращения к функции – это фактические параметры. Они автоматически заменяют формальные параметры A и B по порядку следования. Количество и тип формальных и фактических параметров строго соответствуют друг другу. Рассмотрим пример на составление процедуры вычисления $y=x+1$.

Пример 3.1.5

```
PROGRAM Proc;
VAR a, d, x, y:real;
f1, f2:text;
PROCEDURE ff( VAR x,y:real);
BEGIN
y:=x+1;
END;
BEGIN
assign(f2,'agg.rez'); reset(f2);
read(f2,a,d);
assign(f1,'q'); rewrite(f1);
write(f1,a:4:1,d:4:1);
ff(a,d);
write(a,d);
close(f1);
END.
```

Пример 3.1.6

```
PROGRAM Proc1;
PROCEDURE sq(a, b, c:real;VAR x1, x2 : real);
VAR d:real;
BEGIN
d:=B*B - 4*A*C;
x1:=(-b+SQRT(D))/(2*a);
x2:=(-b-SQRT(D))/(2*a);
END;
VAR y1,y2:real;
BEGIN
sq (5.7,-1.2,-8.3,y1,y2);
write(y1,y2);
END.
```

Пример 3.1.7

```
PROGRAM maxmin;
TYPE b1=ARRAY[1..4,1..3] OF real;
VAR i, j:integer;
ymax, ymin:real;
y:b1;
r1, r2:text;
PROCEDURE mm(VAR x: b1; VAR xmax, xmin:real);
VAR n, k:integer;
BEGIN
xmax:=x[1,1]; xmin:=x[1,1];
FOR n:=1 TO 4 DO
FOR k:=1 TO 3 DO
BEGIN
if x[n,k] <= xmin then xmin:=x[n,k];
if x[n,k] >= xmax then xmax:=x[n,k]
END;
END;
BEGIN
assign(r1,'a.dat'); reset(r1);
FOR i:=1 TO 4 DO
FOR j:=1 TO 3 DO
read(r1,y[i,j]);
assign(r2,'b.res'); rewrite(r2);
mm(y,ymax,ymin);
write(r2,ymax,ymin);
close(r2);
END.
```

3.2. Использование стандартных модулей

3.2.1. Модуль Crt. Работа с экраном в текстовом режиме

Практически любая программа использует дисплей для отображения вводимой и выводимой информации. Всю выводимую на экран дисплея информацию подразделяют на текстовую и графическую. Соответственно выделяют текстовый и графический режимы.

Для инициализации (установки) текстового режима используется процедура TextMode(Mode:word). Выполнение этой процедуры приводит к очистке экрана и активации указанного режима.

TextColor(Color:byte) – устанавливает цвет выводимых на экран символов.

TextBackGround(Color:byte) – устанавливает цвет фона, т. е. цвет области, которая окружает отображаемый на экране символ.

ClrScr – очищает активное окно и устанавливает курсор в верхний левый угол.

GotoXY(x,y:byte) – перемещает курсор в позицию с координатами X,Y в рамках активного окна.

WhereX – возвращает X-координату текущей позиции курсора.

WhereY – возвращает Y-координату текущей позиции курсора.

Для работы с клавиатурой в Crt предусмотрены следующие функции:

KeyPressed:boolean – возвращает значение True, если на клавиатуре была нажата какая-либо клавиша. В противном случае эта функция возвращает значение False.

ReadKey:char – считывает символ с клавиатуры. Считываемый символ на экране не отображается.

Практически любая программа, разработанная для ПЭВМ, использует дисплей для отображения вводимой и выводимой информации. В зависимости от типа используемого в ПЭВМ адаптера (ПЭВМ комплектуется одноцветным и цветным графическим адаптером) всю выводимую на экран дисплея информацию подразделяют на текстовую и графическую. Соответственно выделяют текстовый и графический режимы.

3.2.2. Модуль Graph. Работа с экраном в графическом режиме

Модуль Graph поддерживает графический режим работы дисплея. В этом режиме любое изображение на экране дисплея синтезируется из множества мельчайших элементов, называемых пикселями. Каждый пиксель представляет собой светящуюся точку таких размеров, при которых промежутки между отдельными пикселями отсутствуют. Если группа смежных пикселей светится, то они воспринимаются не как совокупность отдельных точек, а как сплошной участок. Таким образом, на экране дисплея может быть синтезировано любое графическое изображение.

В графическом режиме экран дисплея разделяется прямоугольной сеткой, каждый элемент которой имеет свои координаты. Левый верхний угол экрана имеет координаты (0,0). Значение левой координаты (X) увеличивается в горизонтальном направлении слева направо. Значение правой координаты (Y) увеличивается в вертикальном направлении сверху вниз. Количество точек по горизонтали и вертикали называется разрешающей способностью.

Координаты правой нижней границы экрана можно определить, используя функции GetMaxX и GetMaxY.

Реализация графического режима в ПЭВМ обеспечивается благодаря наличию специальной схемы, называемой графическим адаптером.

ПЭВМ может комплектоваться следующими типами графических адаптеров: CGA, VCGA, EGA, VGA, Hercules, AT&T, PC-3270, IBM-8514. Работу графического адаптера поддерживает специальная программа, называемая драйвером. Загрузочный модуль драйвера хранится в специальном файле с расширением bgi. Используемый адаптер может функционировать в различных режимах.

Рассмотрим пример инициализации (установки) графического режима:

```
USES Graph;
VAR Gd,Gm:integer;
BEGIN
Gd:=Detect;
InitGraph(Gd,Gm,'');
IF GraphResult< >grOk THEN Halt(1);
...CloseGraph
END.
```

Графический режим инициализируется с помощью стандартной процедуры InitGraph. При этом переменным Gd и Gm необходимо указать номер адаптера и номер графического режима. Если переменной Gd предварительно присвоить значение константы Detect, описанной в модуле Graph (ее значение 0), то при загрузке драйвера программа выполнит автоматическое распознавание типа адаптера. При этом если есть выбор графических режимов, то устанавливается тот из них, который обеспечивает более высокое качество изображения. Третий параметр процедуры InitGraph – путь до файла с загрузочным модулем драйвера. Если путь отсутствует, то поиск этого файла будет осуществляться в текущем каталоге. Ошибки, которые могут возникать при инициализации графического режима, анализируют с помощью функции GraphResult. Для выхода из графического режима используется стандартная процедура CloseGraph. Эта процедура восстанавливает режим, существовавший до инициализации графики.

Для создания графических изображений модуль Graph предоставляет широкий набор процедур и функций. Рассмотрим точки и линии.

1. PutPixel(X,Y:integer;Color:word) – ставит на экране точку с координатами (X,Y) цвета Color.

2. Line(X1,Y1,X2,Y2:integer) – выводит на экран линию, соединяющую точки с координатами (X1,Y1) и (X2,Y2).

3. Rectangle(X1,Y1,X2,Y2:integer) – выводит на экран изображение прямоугольника с координатами диагонали (X1,Y1) и (X2,Y2).

4. Circle(X,Y:integer;Radius:word) – выводит на экран изображение окружности с координатами центра (X,Y) и радиусом (Radius).

5. Ellipse(X,Y :integer; StAngle, EndAngle, XRadius, YRadius: word) – выводит на экран изображение эллиптической дуги с центром в точке (X,Y) от начального угла StAngle до конечного угла EndAngle с горизонтальной полуосью XRadius и вертикальной YRadius. Отсчет углов осуществляется относительно горизонтальной оси в направлении против часовой стрелки (3 часа – 0, 12 часов – 90 и т. д.) Если StAngle=0, а EndAngle=360, то будет выведено изображение полного эллипса.

Установка цвета линий осуществляется процедурой SetColor(Color:integer). Цвет фона SetBkColor(Color:word).

Рассмотрим пример:

```
USES Graph;
VAR Gd,Gm:integer;
BEGIN
Gd:=Detect;
InitGraph(Gd,Gm,'d:\bp\bgi');
rectangle(100,50,200,100);
line(100,50,200,100);
line(100,100,200,50);
ellipse(150,75,0,360,50,25);
readln;
CloseGraph
END.
```

3.2.3. Закрашенные области

1. Bar(X1,Y1,X2,Y2:integer) – выводит на экран закрашенный прямоугольник с координатами диагонали (X1,Y1) и (X2,Y2).

2. Bar3D(X1,Y1,X2,Y2:integer;Depth:word;Top:boolean) – выводит на экран изображение закрашенного прямоугольного параллелепипеда, который рисуется в изометрическом изображении с глубиной Depth. Параметр Top определяет, рисовать ли верхнюю грань параллелепипеда. Значение Top выбирается из соответствующего списка констант модуля Graph. Если рисовать, Top =TopOn, если нет,Top = TopOff.

3. FloodFill(X,Y:integer;Border:word) – заполняет (закрашивает) ограниченную область текущем цветом. Граница закрашиваемой области высвечивается цветом, заданным в Border.

4. PieSlice(X,Y:integer;StAngle,EndAngle,Radius:word) – выводит на экран изображение закрашенного сектора круга, используя в качестве центра круга точку (X,Y), начального угла StAngle, конечного угла EndAngle и радиуса Radius. Контур сектора высвечивается текущим цветом. Если StAngle=0, а EndAngle=360 градусам, то PieSlice выводит на экран закрашенную окружность.

5. Sector(X,Y:integer;StAngle,EndAngle,XRadius,YRadius:word) – выводит на экран изображение эллиптического сектора, используя в качестве центра круга точку (X,Y), начального угла StAngle, конечного угла EndAngle, а в качестве горизонтальной и вертикальной полуосей – XRadius и YRadius. Контур сектора высвечивается текущим цветом. Если StAngle=0, а EndAngle=360, то на экране будет выведено изображение закрашенного эллипса.

3.2.4. Вывод текстовой информации

В графическом режиме вывод текстовой информации осуществляется с помощью штриховых и побитовых шрифтов. Каждый символ в штриховом шрифте определен серией отрезков, что позволяет использовать любой коэффициент увеличения символов без ухудшения качества изображения. Побитовый шрифт определен матрицей 8*8 пикселей для каждого символа. Для увеличения побитового шрифта используется коэффициент масштабирования, однако, большое увеличение побитового шрифта делает изображение грубым. Каждый штриховой шрифт хранится в соответствующем файле с расширением CHR.

1. OutTextXY(X,Y:integer;Text:string) – выводит строку, начиная с точки, имеющей координаты (X,Y).

2. SetTextJustify(Horiz,Vert:word) – устанавливает значения выравнивания текста. Для установки значения выравнивания в модуле Graph определены следующие константы:

а) горизонтальное выравнивание;
LeftText = 0 – выравнивание слева,
CenterText = 1 – выравнивание по центру,
RightText = 2 – выравнивание справа,

б) вертикальное выравнивание;
Bottom Text = 0 – выравнивание снизу,
CenterText = 1 – выравнивание по центру,
TopText = 2 – выравнивание сверху.

3. SetTextStyle(Font,Direction,CharSize:word) – устанавливает текущий шрифт, тип и коэффициент увеличения символов. Параметр Font – тип шрифта. Для установки типа шрифта в модуле Graph описаны следующие константы:

DefaultFont = 0 Побитовый шрифт,
TriplexFont = 1 Тройной шрифт,
SmallFont = 2 Малый шрифт,
SansSerifFont = 3 Гротесковый шрифт,
GothicFont = 4 Готический шрифт
и другие.

Direction задает направление вывода. (0 – горизонтальное, слева направо, 1 – вертикальное, снизу вверх). CharSize – коэффициент увеличения символов.

Пример 3.2.1. Массив X(N) напечатать на экране в виде столбцов по M элементов в каждом. Выделить элементы, превышающие по значению величину K.

```
PROGRAM P1;
USES Crt;
VAR x:array[1..20] of real;
i, N, M:integer;
K:real;
BEGIN
writeln(' N M K');readln(N,M,K);
FOR i:=1 TO N DO
BEGIN
write('X[',i,']=');readln(x[i])
END;
ClrScr;
FOR i:=1 TO N DO
IF x[i]>K THEN
BEGIN
TextColor(14); TextBackGround(4);
END
ELSE
BEGIN
TextColor(15); TextBackGround(1);
END;
GotoXY(10*((i-1) div M)+1,(i-1) mod M+1);
write(i:2,' ',x[i]:6:4)
END.
```

Пример 3.2.2. Составить программу введения пароля.

```
USES Crt;
CONST text:ARRAY [1..5] OF string=
(’ ’,
 ’ ВВЕДИТЕ ПАРОЛЬ ’,
 ’ ’,
 ’ ’
   ’,
 ’ ’);
VAR a:string[6]; f1:char; i:integer;
```

```
BEGIN
REPEAT
ClrScr;                    {Очистка экрана}
TextColor(15);             {Установка цвета символов. Белый.}
TextBackGround(3);         {Установка цвета фона. Темно-голубой.}
{Зарисовка рамки}
FOR i:=1 TO 5 DO
BEGIN
GotoXY(25,10+i);
writeln(text[i]);
END;
a:='';
{Ввод пароля}
FOR i:=1 TO 6 DO
BEGIN
GotoXY(32+(i-1),13);
f1:=readkey; {Ввод символа с клавиатуры}
TextColor(5);
GotoXY(32+(i-1),13);
write('*');
a:=a+f1;
END;
IF a='paroll' THEN
BEGIN
TextColor(12);
GotoXY(33,14);
writeln('OK !');
{Звуковой сигнал "Верно"}
sound(2500);
delay(2500);
nosound;
delay(20000);
END
ELSE
BEGIN
TextColor(6);
TextBackGround(0);
{Звуковой сигнал "Неверно"}
sound(100);
delay(2500);
nosound;
delay(20000);
```

```
END;
UNTIL a=’paroll’
END.
```

Пример 3.2.3. Составить программу построения графика функции *y=sin(x)*.

```
USES Graph;
{Описание функции y=sin(x)}
FUNCTION f(x:real):real;
BEGIN f:=sin(x);
END;
VAR gd, gm, Nx, Nx1, Nx2, Ny, Ny1, Ny2:integer;
x, x1, x2, h, y, y1, y2:real;
BEGIN
{Переход в графический режим}
gd:=Detect;
InitGraph(gd,gm,'c:\bp\bgi');
x1:=0;
x2:=4*Pi;
h:=0.01;
y1:=-1;
y2:=1;
{Размеры рамки}
Nx1:=10;
Nx2:=GetMaxX-Nx1;
Ny1:=10;
Ny2:=GetMaxY-Ny1;
{Зарисовка рамки}
rectangle(Nx1,Ny1,Nx2,Ny2);
{Построение графика по точкам}
x:=x1;
REPEAT
y:=f(x);
{Вычисление координаты точки экрана по заданным x и y}
Nx:=round(Nx1+(Nx2-Nx1)*(x-x1)/(x2-x1));
Ny:=round(Ny2-(Ny2-Ny1)*(y-y1)/(y2-y1));
PutPixel(Nx,Ny,12);
x:=x+h;
UNTIL x>x2;
readln; {Пауза}
closegraph {Выход из графического режима}
END.
```

4. РЕШЕНИЕ ФУНКЦИОНАЛЬНЫХ И ВЫЧИСЛИТЕЛЬНЫХ ЗАДАЧ ХИМИЧЕСКОЙ ТЕХНОЛОГИИ

4.1. Обработка экспериментальных данных

4.1.1. Интерполяционный многочлен Лагранжа

Интерполяционная формула Лагранжа используется для произвольно заданных узлов интерполирования.

Пусть в точках x_0, x_1, …, x_n таких, что $a \le x_0 < \ldots < x_n \le b$, известны значения функции $y = f(x)$, т. е. на отрезке $[a, b]$ задана табличная (сеточная) функция

x	x_0	x_1	…	x_n
y	y_0	y_1	…	y_n

Требуется построить некоторую функцию $P_n(x)$ степени не выше n, имеющую в заданных узлах x_0, x_1, …, x_n те же значения, что и функция $f(x)$. С этой целью представим $P_n(x)$ в следующем виде:

$$P_n(x_i) = C_i(x - x_0)(x - x_1)\ldots(x - x_{i-1})(x - x_{i+1})\ldots(x - x_n)$$

где C_i – некоторая константа, которая определяется из следующего уравнения:

$$P_n(x_i) = y_i.$$

В этом случае для многочлена:

$$C_i = \frac{1}{(x_i - x_0)(x - x_1)\ldots(x - x_{i-1})(x - x_{i+1})\ldots(x_i - x_n)}$$

Подставляя выражение для константы C_i в исходную зависимость полинома для каждого $i = 0, 1, 2, \ldots, n$ следующее значение многочлена:

$$P_n(x_i) = y_i \frac{(x - x_0)(x - x_1)\ldots(x - x_{i-1})(x - x_{i+1})\ldots(x - x_n)}{(x_i - x_0)(x_i - x_1)\ldots(x_i - x_{i-1})(x_i - x_{i+1})\ldots(x_i - x_n)},$$

Подставляя значение $P_n(x)$ для всех точек от 0 до n, получим следующее выражение для многочлена Лагранжа:

$$P_n(x) = \sum_{i=0}^{n} \frac{(x - x_0)\ldots(x - x_{i-1})(x - x_{i+1})\ldots(x - x_n)}{(x_i - x_0)\ldots(x_i - x_{i-1})(x_i - x_{i+1})\ldots(x_i - x_n)} y_i$$

Пример 4.1.1. Построить интерполяционный полином для функции $y = \sin x$.

Возьмем сетку, состоящую из трех точек: $x_0 = 0$; $x_1 = \frac{\pi}{6}$; $x_2 = \frac{\pi}{2}$, выпишем соответствующие этим аргументам значения функции $\sin x$: $y_0 = 0$; $y_1 = \frac{1}{2}$; $y_2 = 1$.

Построим по этой таблице интерполяционный полином второй степени, используя формулу Лагранжа:

$$P_2(x) = 0 \cdot \frac{(x-\frac{\pi}{6})(x-\frac{\pi}{2})}{(-\frac{\pi}{6})(-\frac{\pi}{2})} + \frac{1}{2} \cdot \frac{x(x-\frac{\pi}{2})}{\frac{\pi}{6} \cdot (-\frac{\pi}{3})} + 1 \cdot \frac{x(x-\frac{\pi}{6})}{\frac{\pi}{2} \cdot \frac{\pi}{3}} = \frac{7}{2\pi} x - \frac{3}{\pi^2} x^2 .$$

Проверяем, что в точках сетки этот полином принимает нужные значения. Оценим погрешность интерполирования, сравнив значения *sin x* и интерполяционного полинома в точке $x = \frac{\pi}{4}$.

$$sin\frac{\pi}{4} = \frac{1}{\sqrt{2}} \approx 0{,}707107,$$

$$P_2(\frac{\pi}{4}) = \frac{11}{16} = 0{,}6875,$$

$$\varepsilon = \sin\frac{\pi}{4} - P_2(\frac{\pi}{4}) = 0{,}019607 \approx 0{,}02 .$$

Значительная величина погрешности определяется тем, что на отрезке длиной $\frac{\pi}{2}$ мы взяли грубую сетку, состоящую всего из трех точек. Чтобы улучшить точность интерполирования, нужно либо увеличить число точек n и повысить соответственно степень интерполяционного полинома $P_n(x)$, либо уменьшить длину исходного отрезка.

4.1.2. Интерполяционный многочлен Ньютона

Предположим дополнительно, что рассматриваемые значения аргумента являются равноотстоящими, т. е. образуют арифметическую прогрессию.

В этом случае шаг таблицы $h = x_{i+1} - x_i (i = 0, 1, 2, ..., n) = \text{const}$ является величиной постоянной. Для таких таблиц построение интерполяционных формул (как впрочем, и вычисление по этим формулам) заметно упрощается.

Прежде чем перейти к рассмотрению этого вопроса, познакомимся с понятием конечных разностей.

Пусть функция задана таблицей с постоянным шагом. Разности между значениями функции в соседних узлах интерполяции называют конечными разностями первого порядка:

$$\Delta y_i = y_{i+1} - y_i \ (i = 0, 1, 2, ...).$$

Из конечных разностей первого порядка образуются конечные разности второго порядка:

$$\Delta^2 y_i = \Delta y^{i+1} - \Delta y_i \ (i = 0, 1, 2, ...).$$

Для записи конечных разностей используются горизонтальные и диагональные таблицы.

Горизонтальная таблица

x	y	Δy	$\Delta^2 y$	$\Delta^3 y$
x_0	y_0	Δy_0	$\Delta^2 y_0$	$\Delta^3 y_0$
x_1	y_1	Δy_1	$\Delta^2 y_1$	
x_2	y_2	Δy_2		
x_3	y_3			

Диагональная таблица

x	y	Δy	$\Delta^2 y$	$\Delta^3 y$
x_0	y_0	Δy_0	$\Delta^2 y_0$	$\Delta^3 y_0$
x_1	y_1	Δy_1	$\Delta^2 y_1$	
x_2	y_2	Δy_2		
x_3	y_3			

Формула Ньютона «вперед»:

$$P_n(x) = y_0 + \frac{\Delta y}{h}(x - x_0) + \frac{\Delta^2 y_0}{2! h^2}(x - x_0)(x - x_1) + \cdots \frac{\Delta^n y_0}{n! h^n}(x - x_0)(x - x_1)\ldots(x - x_{n-1}),$$

где n – порядок полинома, h – шаг (расстояние между узлами).

Первый порядок:

$x_1 = x_0 - h$ $\quad \Delta y = y_1 - y_0$

$x_2 = x_0 - 2h$ $\quad \Delta y_1 = y_2 - y_1$

$x_n = x_0 - n \times h$ $\quad \Delta y_{n-1} = y_n - y_{n-1}$

Второй порядок:

$\Delta^2 y = \Delta y_1 - \Delta y_0$

$\Delta^2 y_1 = \Delta y_2 - \Delta y_1$

$\Delta^2 y_{n-2} = \Delta y_{n-1} - \Delta y_{n-2}$

Формула Ньютона «назад»:

$$P_n(x) = y_n + \frac{\Delta y_{n-1}}{1! h^1}(x - x_n) + \frac{\Delta^2 y_{n-2}}{2! h^2}(x - x_n)(x - x_{n-1}) + \cdots \frac{\Delta^n y_0}{n! h^n}(x - x_n)(x - x_{n-1})\ldots(x - x_0),$$

Программа для интерполирования «вперед»:

```
program newton;
const n=3;
var d:array[0..n,0..n] of real;
```

Программа для интерполирования «назад»:

```
program newton;
const n=3;
var d:array[0..n,0..n] of real;
```

```
x,y:array[0..n] of real;
p:array[1..n] of real;
f1,f2:text; h,x1,s1:real;i,j:integer;
begin
as-
sign(f1,'n1.pas');assign(f2,'nnn.pas');
reset(f1);rewrite(f2);x1:=1.5;h:=1;
for i:=0 to n do read(f1,x[i],y[i]);
s1:=y[0];
for i:=0 to n do d[i,0]:=y[i];
for j:=1 to n do for i:=0 to n-j do
d[i,j]:=d[i+1,j-1]-d[i,j-1];
for i:=1 to n do begin p[i]:=1;for j:=1 to
i do
p[i]:=p[i]*(x1-x[j-1])/(h*j);
s1:=s1+d[0,i]*p[i]; end;
write(f2,s1:7:4);close(f2);end.
```

```
x,y:array[0..n] of real;
p:array[1..n] of real;
f1,f2:text; h,x1,s1:real;i,j:integer;
begin
as-
sign(f1,'n1.pas');assign(f2,'nn3.pas');
reset(f1);rewrite(f2);x1:=1.5;h:=1;
for i:=0 to n do
read(f1,x[i],y[i]);s1:=y[n];
for i:=0 to n do d[i,0]:=y[i];
for j:=1 to n do for i:=0 to n-j do
d[i,j]:=d[i+1,j-1]-d[i,j-1];
for i:=1 to n do begin p[i]:=1;for j:=1 to
i do
p[i]:=p[i]*(x1-x[n-j+1])/(h*j);
s1:=s1+d[n-i,i]*p[i]; end;
write(f2,s1:5:2);close(f2);end.
```

4.2. Итерационные методы решения нелинейных уравнений

Итерационные методы используются для решения нелинейных алгебраических и трансцендентных уравнений.

Уравнения в общем случае можно представить следующим образом:

$$f(x) = 0.$$

Нелинейные уравнения можно разделить на два класса – алгебраические и трансцендентные.

Алгебраическими считаются уравнения, содержащие только алгебраические функции (целые, рациональные, иррациональные).

Алгебраическое уравнение в общем виде можно представить многочленом n-й степени с действительными коэффициентами:

$$f(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1} + a_n x^n = 0.$$

Уравнения, содержащие другие функции (тригонометрические, показательные, логарифмические и др.), называются трансцендентными, например: $x^3 + x^2 + 2e^x + 5 = 0$; $2x - \sin 3x = 0$.

Задача решения уравнения заключается в нахождении таких значений x, которые обращают его в тождество.

4.2.1. Метод деления отрезка пополам

Рассмотрим некоторые итерационные методы решения нелинейных уравнений, которые часто используются при расчете химико-технологических систем.

Пусть дано уравнение $f(x) = 0$. Допустим, нам удалось найти такой отрезок $[a, b]$, на котором расположено значение корня x, т. е. $a < x < b$. В качестве начального приближения корня x_0 принимаем середину отрезка $x_0 = (a + b)/2$. Далее исследуем значения функции: если $f(x_0) = 0$, то x_0 является корнем уравнения, т. е. $x = x_0$. Если $f(x_0) \neq 0$, то выбираем одну из половин отрезка $[a, x_0]$ или $[x_0, b]$, на концах которой функция $f(x)$ имеет противоположные знаки, т. е. содержит искомый корень, поэтому его принимаем в качестве нового отрезка $[x_0, b]$. Вторую половину отрезка, на концах которого знак $f(x)$ не меняется, отбрасываем: в данном случае $[a, x_0]$. Отрезок $[x_0, b]$ вновь делим пополам. Новое приближение: $x_1 = (x_0 + b)/2$. Вновь исследуем функцию *f(x)* на концах отрезка и отбрасываем отрезок $[x_0, x_1]$, т. к. $f(x_0) > 0$ и $f(x_1) > 0$. Отрезок $[x_1, b]$, на концах которого функция имеет противоположные знаки $f(x_1) < 0$, $f(b) > 0$, вновь делим пополам и получаем новое приближение корня $x_2 = (x_1 + b)/2$ и т. д. Итерационный процесс продолжаем до тех пор, пока значение функции $f(a)$ после n-й итерации не станет меньше некоторого заданного малого числа (погрешности).

4.2.2. Метод простых итераций

Метод итераций представляет собой циклический процесс, очередное приближение которого есть корень с определенной точностью.

Исходя из найденного на предыдущем шаге значения x_{n-1}, вычисляем $y = f(x_{n-1})$. Если $|y - x_{n-1}| >$ eps , то полагают $x_n = y$ и выполняют очередную итерацию. Если же $|y - x_n| <=$ eps, то приближенные вычисления заканчивают и за решение принимают $x_n = y$.

Рассмотрим пример решения уравнения $e^x - 10x = 0$; eps = 0,001 на отрезке [0; 6].

Корни x_1 и x_2 легко отделяются графически. Они являются абсциссами точки пересечения графиков e^x с прямой $y = 10x$. Для определения корня заменим исходное уравнение эквивалентным $x = 0{,}1e^x$. На отрезке [0; 1] $f'(0)$=0,1; $f'(1) = 0{,}271$; т. к. функция e^x монотонная, то $0 < f'(x) < 1$, то есть $q = 0{,}271$. В качестве начального приближения выбираем $x_0 = 1$. Вычисления прекращаются, когда $|X_n - f(x_n)| <=$ eps. Последовательные приближения в этом случае таковы: $x_1 = 0{,}271$, $x_2 = 0{,}131$, $x_3 = 0{,}114$, $x_4 = 0{,}112$, $x_5 = 0{,}111$, то есть $x_1 = 0{,}111$ с точностью 0,001.

Для определения второго корня представляем исходное уравнение в виде $x = \ln(10x)$. В этом случае эквивалентной функций будет являться $f(x) = \ln(10x)$, а $f'(x) = 1/x$;$|1/x| <= 0{,}5$, то есть $q = 0{,}5$, $x_0 = 2$, $x_1 = 2{,}995$, $x_2 = 3{,}399$, $x_3 = 3{,}526$, $x_4 = 3{,}562$, $x_5 = 3{,}576$, то есть $x_1 = 3{,}576$ с точностью 0,001.

4.2.3. Метод Ньютона (метод касательных)

Пусть рассматривается уравнение $f(x) = 0$. Корнем уравнения называется значение $\tilde{x}$, при котором $f(x) = 0$. Корень $\tilde{x}$ называется простым, если $F'(\tilde{x}) \neq 0$ в противном случае корень называется кратным.

При решении нелинейного уравнения методом касательных задаются начальное значение аргумента x_0 и точность ε. Затем в точке $(x_0, F(x_0))$ проводим касательную к графику $F(x)$ и определяем точку пересечения касательной с осью абсцисс x_1. В точке $(x_1, F(x_1))$ снова строим касательную, находим следующее приближение искомого решения x_2 и т. д. Указанную процедуру повторяем, пока $|F(x_i)| > \varepsilon$. Расчетная формула метода Ньютона имеет вид:

$$x^{(n+1)} = x^{(n)} - \frac{f\left(x^{(n)}\right)}{f'\left(x^{(n)}\right)}$$

Геометрически метод Ньютона означает, что следующее приближение к корню $x^{(n+1)}$ есть точка пересечения с осью OX касательной, проведенной к графику функции $y = f(x)$ в точке $(x^{(n)}, f(x^{(n)}))$.

4.2.4. Примеры составления программ

Пример 4.2.1. Программа решения уравнений методом деления отрезка пополам.

Задание: решить уравнение $ex - 10x = 0$.

```
PROGRAM del;
USES Crt;
LABEL 1,2;
VAR a, b, x0, eps, r1, r2: Real;
{ a, b – Начало и конец интервала }
{ x0 – Корень уравнения }
{ eps – Точность расчета }
k: Integer;
fin, fout: Text;
FUNCTION f(x:real): Real; { Исследуемая функция }
BEGIN
f:=exp(x)-(10*x);
END;
BEGIN
ClrScr; { Очистка текстового экрана }
assign(fin,'otrezok.in'); { Файл данных }
assign(fout,'otrezok.out'); { Файл вывода }
reset(fin);
```

```
read(fin,a,b,eps);
close(fin);
(* Р А С Ч Е Т *)
k:=0; 1:k:=k+1;
x0:=(a+b)/2;
IF f(x0)=0 THEN GOTO 2;
IF abs(b-a)<eps THEN GOTO 2;
r1:=f(x0);
r2:=f(a);
IF (r1*r2)<0 THEN b:=x0
ELSE a:=x0;
GOTO 1;
2:  writeln('Число делений пополам: ',k);
writeln('Корень уравнения: ',x0:7:4);
writeln('Точность расчета: ',eps:7:6);
(* Запись в файл *)
rewrite(fout);
writeln(fout,'Число делений пополам: ',k);
writeln(fout,'Корень уравнения: ',x0:7:4);
writeln(fout,'Точность расчета: ',eps:7:6);
close(fout);
readln;
END.
```

ФАЙЛ исходных данных:

0.0 1.0 1.0E-08 a, b – Начало и конец интервала eps – Точность расчета

ФАЙЛ с результатами

Число делений пополам: 28
Корень уравнения: 0,1118
Точность расчета: 0,000000001

Пример 4.2.2. Программа решения уравнения с использованием метода Ньютона.

Уравнение $x^3 - 2x^2 + 1{,}3x = 0$.

```
PROGRAM newton;
LABEL m1, m2;
VAR x, x0, eps:real;
a1, a2:text;
k:integer;
```

```
FUNCTION f(x0:real):real;
BEGIN
f:=x*x*x-2*x*x+1.3*x-0.2;
END;
FUNCTION f1(x0:real):real;
BEGIN
f1:=3*x*x-4*x+1.3;
END;
BEGIN
assign (a1,'dat17-2');
reset(a1);
read(a1,x0,eps);
k:=0;
m2: x:=x0;
x0:=x-f(x)/f1(x);
k:=k+1;
IF abs(x-x0) <= eps THEN GOTO m1
ELSE GOTO m2;
m1: assign(a2,'res-17-2');
rewrite(a2);
writeln(a2,'корень уравнения',x,k);
close (a2);
END.
```

Пример 4.2.3. Программа решения уравнения методом итераций.

```
PROGRAM met;
LABEL m1, m2;
VAR x, x0, eps:real;
a1, a2:text;
k:integer;
FUNCTION f(x0:real):real;
BEGIN
f:=(-x0*x0*x0+2*x0*x0+0.24)/1.3;
END;
FUNCTION f1(x0:real):real;
BEGIN
f1:=-3*x0*x0+4*x0+1.3;
END;
BEGIN
assign (a1,'dat-21'); reset(a1);
```

```
read(a1,x0,eps);
k:=0;
REPEAT
x:=x0;
writeln (x0);
x0:=f(x); k:=k+1;
UNTIL abs(x-x0) < eps ;
m1: assign(a2,'res-21'); rewrite(a2);
writeln(a2,'корень уравнения',x,k);
close (a2);
END.
```

4.3. Приближенное решение обыкновенных дифференциальных уравнений первого порядка

При решении научных и инженерно-технических задач часто бывает необходимо математически описать какую-либо динамическую систему. Лучше всего это делать в виде дифференциальных уравнений (ДУ) или системы дифференциальных уравнений. Наиболее часто такая задача возникает при решении проблем, связанных с моделированием кинетики химических реакций и различных явлений переноса (тепла, массы, импульса) – теплообмена, перемешивания, сушки, адсорбции, при описании движения макро- и микрочастиц.

Обыкновенным дифференциальным уравнением (ОДУ) *n*-го порядка называется следующее уравнение, которое содержит одну или несколько производных от искомой функции y(x):

$$G\left(x, y, y', \ldots, y^{(n-1)}, y^{(n)}\right) = 0$$

где $y^{(n)}$ обозначает производную порядка *n* некоторой функции $y(x)$; x – это независимая переменная.

Решением обыкновенного дифференциального уравнения называется такая функция $y(x)$, которая при любых x удовлетворяет этому уравнению в определенном конечном или бесконечном интервале. Процесс решения дифференциального уравнения называют интегрированием дифференциального уравнения.

Общее решение ОДУ *n*-го порядка содержит *n* произвольных констант $C_1, C_2, \ldots, C_n$

$$y = f(x, y, C_1, C_2, \ldots, C_n$$

Частное решение ОДУ получается из общего, если константам интегрирования придать некоторые значения, определив некоторые дополнительные условия, количество которых позволяет вычислить все неопределенные константы интегрирования.

Точное (аналитическое) решение (общее или частное) дифференциального уравнения подразумевает получение искомого решения (функции $y(x)$) в виде выражения от элементарных функций.

Численное решение ДУ (частное) заключается в вычислении функции $y(x)$ и ее производных в некоторых заданных точках $x_1, x_2, \ldots, x_N$, лежащих на определенном отрезке.

4.3.1. Метод Эйлера

Простейшим численным методом решения обыкновенных дифференциальных уравнений является метод Эйлера. В его основе лежит аппроксимация производной отношением конечных приращений зависимой (y) и независимой (x) переменных между узлами равномерной сетки

$$y' = \frac{dy}{dx} \approx \frac{\Delta y}{\Delta x} = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} = F,$$

где y_{i+1} – это искомое значение функции в точке x_{i+1}.

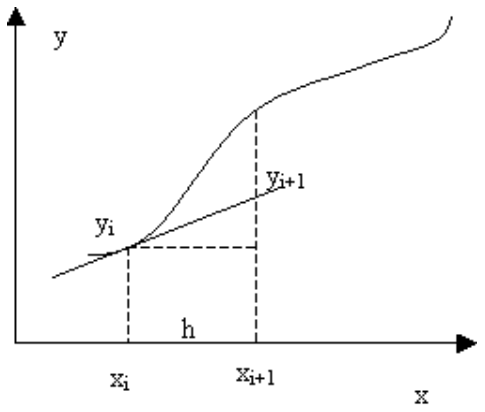


Рис. 4.1. Графическая иллюстрация метода Эйлера

Если теперь преобразовать это уравнение, и учесть равномерность сетки интегрирования, то получится итерационная формула, по которой можно вычислить y_{i+1}, если известно y_i в точке x_i:

$$y_{i+1} = y_i + F(x_i, y_i) \cdot h$$

Таким образом, суть метода Эйлера заключается в замене функции $y(x)$ на отрезке интегрирования прямой линией, касательной к графику в точке $x = x_i$. Если искомая функция сильно отличается от линейной на отрезке интегрирования, то погрешность вычисления будет значительной. Ошибка метода Эйлера прямо пропорциональна шагу интегрирования:

Пример 4.3.1. Рассчитать кинетику гомогенной химической реакции, используя метод Эйлера.

```
VAR ca0, cb0, cc0, cd0, k1, k2, k3, k4, t0, tk, h, ca, cb, cc, cd: Real;
{Описание переменных концентрации, констант скорости, время реагирования, шага}
```

```
f1, f2 :Text;
{Файловые переменные}
BEGIN
assign(f1,'al_ar.in');
{Файл ввода данных}
assign(f2,'al_ar.out');
{Файл вывода расчетов}
reset(f1);
readln(f1,ca0,cb0,cc0,cd0,k1,k2,k3,k4,t0,tk,h);
{ Ca0, Cb0, Cc0, Cd0 – Концентрации веществ для t0 }
{ k1, k2, k3, k4 – Константы скорости для каждой реакции }
{ t0, tk – Начальное и конечное время реагирования }
{ Ca, Cb, Cc, Cd – Переменные для расчета концентраций }
close(f1);
ca:=ca0; cb:=cb0; cc:=cc0; cd:=cd0;
rewrite(f2);
writeln(f2,'Метод Эйлера');
writeln(f2,'Время Концентрация');
writeln(f2,' A B C D ');
REPEAT
ca:=ca0+h*((-1*(k1+k4))*ca0+k3*cd0);
cb:=cb0+h*(k1*ca0-k2*cb0);
cc:=cc0+h*k2*cb0;
cd:=cd0+h*((k4*ca0)-(k3*cb0));
writeln(f2,' ',t0:8:4,' ',ca:8:4,' ',cb:8:4,' ',cc:8:4,' ',cd:8:4,' ');
ca0:=ca; cb0:=cb; cc0:=cc; cd0:=cd; t0:=t0+h;
UNTIL t0>=tk;
writeln(f2,' ',t0:8:4,' ',ca:8:4,' ',cb:8:4,' ',cc:8:4,' ',cd:8:4,' ');
close(f2);
END.
```

4.3.2. Метод Рунге–Кутта

Дальнейшее улучшение точности решения ОДУ первого порядка возможно за счет увеличения точности приближенного вычисления интеграла в выражении.

Воспользовавшись формулой Симпсона, можно получить еще более точную формулу для решения ОДУ первого порядка – широко используемого в вычислительной практике метода Рунге–Кутта.

В формуле Симпсона для приближенного вычисления определенного интеграла используются значения подинтегрального выражения в трех точках. В интеграле их всего две, поэтому введем дополнительную точку в середине отрезка $[x_{i+1}\ x_i]$.

$$x_{i+\frac{1}{2}} = x_i + \frac{h}{2},$$

тогда можно переписать так:

$$y_{i+1} = y_i + \frac{h/2}{3}\left[F(x_i, y_i) + 4F\left(x_{i+\frac{h}{2}}, y_{i+\frac{h}{2}}\right) + F(x_{i+1}, y_{i+1})\right]$$

Полученное выражение является неявным, так как в правой части содержатся еще не определенные значения функции $y_{i+h/2}$ и y_{i+1}. Чтобы воспользоваться этой формулой, надо использовать некоторое приближение для вычисления этих значений

$$y_{i+1} = y_i + \frac{h}{6}\left[F(x_i, y_i) + 4F\left(x_{i+\frac{h}{2}}, \tilde{y}_{i+\frac{h}{2}}\right) + F(x_{i+1}, \tilde{y}_{i+1})\right]$$

При использовании различных методов приближенного вычисления этих величин, получаются выражения для методов Рунге–Кутта различного порядка точности.

Алгоритм Рунге–Кутта третьего порядка (погрешность порядка h^3):

$$y_{i+1} = y_i + \frac{1}{6}(k_0 + 4k_1 + k_2)$$

где

$$k_0 = hF(x_i, y_i)$$
$$k_1 = hF(x_i + \frac{h}{2}, y_i + \frac{k_0}{2})$$
$$k_2 = hF(x_i + h, y_i + 2k_1 - k_0)$$

Алгоритм Рунге–Кутта четвертого порядка (погрешность порядка h^4):

$$y_{i+1} = y_i + \frac{1}{6}(k_0 + 2k_1 + 2k_2 + k_3)$$

где

$$k_0 = hF(x_i, y_i)$$
$$k_1 = hF(x_i + \frac{h}{2}, y_i + \frac{k_0}{2})$$
$$k_2 = hF(x_i + \frac{h}{2}, y_i + \frac{k_1}{2})$$
$$k_3 = hF(x_i + h, y_i + k_2)$$

Пример 4.3.2. Рассчитать кинетику гомогенной химической реакции, используя метод Рунге–Кутта.

```
VAR ca, cb, cc, cd, ca0, cb0, cc0, cd0, k1, k2, k3, k4, t0, tk, h:real;
{Ca, Cb, Cc, Cd – Концентрации веществ до реакции }
{k1, k2, k3, k4 – Скорости для каждой из реакций}
{t0, tk – Начальное и конечное время реагирования}
{h – шаг}
```

```
x, q1, q2, q3, q4, q5, q6, q7, q8, q9, q10, q11, q12, q13, q14, q15, q16:real;
{Qn – Коэффициенты Рунге–Кутты}
f1, f2:text;
BEGIN
assign(f1,'runge.in');{Файл с данными}
assign(f2,'runge.out'); {Файл с результатами }
reset(f1); read(f1,ca0,cb0,cc0,cd0,k1,k2,k3,k4,t0,tk,h);
close(f1);
cc:=cc0;ca:=ca0;cb:=cb0;cd:=cd0; x:=t0;
rewrite(f2);
Writeln(f2,' Метод Рунге–Кутта');
writeln(f2,' Время Концентрация ');
writeln(f2,'A B C D ');
REPEAT
q1:=h*((-1)*(k1+k4)*ca0+(k3*cd0));
q2:=h*((-1)*(k1+k4)*(ca0+q1/2)+(k3*(cd0+q1/2)));
q3:=h*((-1)*(k1+k4)*(ca0+q2/2)+(k3*(cd0+q2/2)));
q4:=h*((-1)*(k1+k4)*(ca0+q3)+(k3*(cd0+q3)));
q5:=h*((k1*ca0)-(k2*cb0));
q6:=h*((k1*(ca0+q5/2))-k2*(cb0+q5/2)));
q7:=h*((k1*(ca0+q6/2))-(k2*(cb0+q6/2)));
q8:=h*((k1*(ca0+q7))-(k2*(cb0+q7)));
q9:=h*(k2*cb0);
q10:=h*(k2*(cb0+q9/2));
q11:=h*(k2*(cb0+q10/2));
q12:=h*(k2*(cb0+q10));
q13:=h*((k4*ca0)-(k3*cb0));
q14:=h*((k4*(ca0+q13/2))-(k3*(cb0+q13/2)));
q15:=h*((k4*(ca0+q14/2))-(k3*(cb0+q14/2)));
q16:=h*((k4*(ca0+q15))-(k3*(cb0+q15)));
ca:=ca0+((q1+q2+q3+q4)/6);
cb:=cb0+((q5+q6+q7+q8)/6);
cc:=cc0+((q9+q10+q11+q12)/6);
cd:=cd0+((q13+q14+q15+q16)/6);
writeln(f2,' ',x:8:4,' ',ca:8:4,' ',cb:8:4,' ',cc:8:4,' ',cd:8:4,' ');
ca0:=ca; cb0:=cb; cc0:=cc; cd0:=cd;
x:=x+h;
UNTIL x>=tk;
writeln(f2,' ',x:8:4,' ',ca:8:4,' ',cb:8:4,' ',cc:8:4,' ',cd:8:4,' ');
close(f2);
END.
```

5. ОСНОВЫ РАБОТЫ С WINDOWS

Эта глава предназначена в первую очередь для тех, кто впервые садится за персональный компьютер. В ней Вы получите обобщенную информацию, начальные знания, советы по работе в операционной среде WINDOWS и некоторых наиболее необходимых программных продуктах.

Прежде чем перейти к конкретному изучению работы в операционной системе WINDOWS XP, хотелось бы привести некоторую информационную справку.

Сейчас трудно себе представить, но еще несколько лет назад работать на персональном компьютере мог только специально для этого обученный профессионал программист. Сегодня можно уверенно сказать, что на персональном компьютере может работать каждый. Необходимость уметь работать на компьютере диктуется жизнью, т. к. немыслимо представить квалифицированного специалиста, не использующего возможности, предоставляемые нам компьютерной техникой.

История Windows берет свое начало в 1986 году, когда появилась первая версия системы. Она представляла собой набор программ, расширяющих возможности существующих операционных систем для большего удобства в работе. Через несколько лет вышла вторая версия, но особой популярности система Windows не завоевала. Однако в 1990 году вышла новая версия – Windows 3.0, которая стала использоваться на многих персональных компьютерах. Популярность новой версии Windows объяснялась несколькими причинами. Графический интерфейс позволяет работать с объектами вашего компьютера не с помощью команд, а с помощью наглядных и понятных действий над значками, обозначающими эти объекты. Возможность одновременной работы с несколькими программами значительно повысила удобство и эффективность работы. Кроме того, удобство и легкость написания программ для Windows привели к появлению все больше разнообразных программ, работающих под управлением Windows. Наконец, лучше была организована работа с разнообразным компьютерным оборудованием, что также определило популярность системы. Последующие версии Windows были направлены на повышение надежности, а также поддержку средств мультимедиа (версия 3.1) и работу в компьютерных сетях (версия 3.11).

Параллельно с разработкой Windows компания Microsoft в 1988 году начала работу над новой операционной системой, названной Windows NT. Перед новой системой были поставлены задачи существенного повышения надежности и эффективной поддержки сетевой работы. При этом интерфейс системы не должен был отличаться от ин-

терфейса Windows 3.0. Интересно, что самой распространенной версией Windows NT также стала третья версия. В 1992 году появилась версия Windows NT 3.0, а в 1994 году – Windows NT 3.5.

Процесс развития операционных систем не стоит на месте, и в 1995 году появилась система Windows 95, ставшая новым этапом в истории Windows. По сравнению с Windows 3.1 значительно изменился интерфейс, выросла скорость работы программ. Одной из новых возможностей Windows 95 была возможность автоматической настройки дополнительного оборудования компьютера для работы без конфликтов друг с другом. Другой важной особенностью системы стала возможность работы с Интернетом без использования дополнительных программ.

Интерфейс Windows 95 стал основным для всего семейства Windows, и в 1996 году появляется переработанная версия Windows NT 4.0, имеющая такой же интерфейс, как и Windows 95. Продолжением развития Windows 95 стала операционная система, появившаяся в 1998 году. При сохранившемся интерфейсе внутренняя структура была значительно переработана. Много внимания было уделено работе с Интернетом, а также поддержке современных протоколов передачи информации – стандартов, обеспечивающих обмен информацией между различными устройствами. Кроме того, особенностью Windows 98 является возможность работы с несколькими мониторами.

Следующим этапом в развитии Windows стало появление Windows 2000 и Windows Me (Millennium Edition – редакция тысячелетия). Система Windows 2000 разработана на основе Windows NT и унаследовала от нее высокую надежность и защищенность информации от постороннего вмешательства. Операционная система Windows Me стала наследницей Windows 98, но приобрела многие новые возможности. Прежде всего, это улучшенная работа со средствами мультимедиа, возможность записывать не только аудио, но и видеоинформацию, мощные средства восстановления информации после сбоев и многое другое. Постепенно разница между разными системами Windows стирается, и новая операционная система Windows XP предназначена для замены как Windows 2000, так и Windows Me.

Управление манипулятором мышь

Несмотря на то, что большинство команд в Windows XP может выполняться с помощью клавиатуры, в первую очередь, операционная система ориентирована на работу с манипулятором мышь, представляющая собой небольшую коробочку с двумя кнопками и колесиком между ними (скрол), при перемещении которой по столу перемещается стрелка-указатель на экране монитора.

Вначале следует научиться правильно держать мышь в руке. Поднесите к мыши ладонь сверху, чтобы пальцы оказались над кнопками. Возьмитесь за бока большим пальцем и мизинцем. При этом не следует сильно сжимать мышь – только слегка касайтесь пальцами боковых граней, указательный палец положите на левую кнопку мыши, а средний палец – на правую. Не следует прижимать ладонь к мыши. Важно, чтобы при работе с мышью ваша рука не уставала, поэтому расслабьте кисть и не напрягайте пальцы.

Мышь на экране управляет указателем (курсором), вид которого зависит от выполняемой операции. Движение мыши по ровной поверхности (столу) автоматически повторяется соответствующим перемещением указателя на экране.

Нажимая кнопки, расположенные на мыши, можно вызвать то или иное действие, определяемое расположением указателя на экране. Левую кнопку нажимают указательным пальцем, а правую – средним. Важно научиться контролировать перемещение указателя при движении мыши, чтобы быстро и легко перемещать указатель к нужному месту экрана. Не стоит расстраиваться, если у вас не получается сразу быстро и без рывков перемещать указатель мыши. Такое умение достигается с опытом и придет к вам со временем. Следует отметить, что большинство мышей лучше двигаются по специальному коврику, так что наличие такого коврика является необходимым условием плодотворной работы с Windows. Исключение составляют современные оптические мыши, которые могут работать практически на любой поверхности.

В Windows XP используются две клавиши мыши:

- левая клавиша – для выполнения действий (основная), при этом для выполнения запуска приложений Вы должны очень быстро «щелкнуть» левой клавишей два раза;
- правая клавиша – для получения информации по свойствам любого объекта.

С помощью мыши пользователь может выполнить следующие операции:

- щелчок (фиксация). Для выполнения необходимо быстро нажать и отпустить кнопку мыши в тот момент, когда вершина стрелки указателя мыши неподвижна и находится на нужном объекте. Одна из самых распространенных операций, выполняемая щелчком мыши – выбор объекта;
- двойной щелчок производится быстрым двойным нажатием на кнопку мыши без ее перемещения. Двойной щелчок выполняют после того, как указатель мыши помещен на объект (элемент), при этом щелкнуть надо быстро иначе система воспримет их как два одиночных щелчка и выполнит другие команды. Двойной щелчок используется при открытии документов, запуске программ;

- перетаскивание (перемещение) объекта производится после установки указателя мыши на нужном объекте (элементе). Затем нажав левую кнопку мыши и не отпуская ее, перемещают мышь двигая объект или элемент к конечному положению. Кнопку мыши отпускают в тот момент, когда хотят закончить операцию. Перетаскивание широко применяется для перемещения значка папки или файла, окна или его границы.

Курсор ввода

Текущую позицию экрана, где будет располагаться символ, вводимый с клавиатуры, определяет курсор ввода в виде вертикальной мигающей черты.

Чтобы задать положение курсора ввода необходимо переместить мышь так, чтобы указатель находился в нужной точке экрана, и щелкнуть кнопкой мыши.

Рабочий стол

Итак, вы включили компьютер, дождались, когда загрузится Windows XP и приступаете к работе.

Вы видите на экране монитора следующую картину:

Где вся поверхность экрана называется «рабочим столом». Рисунки на экране – «значки».

Окно программы мой компьютер

С помощью значка Мой компьютер пользователь получает доступ ко всем файлам и папкам на компьютере, к сетевому диску, различным устройствам: принтерам, модемам и т. п. и главное к их настройке. В диалоговом окне, появляющемся после щелчка по значку Мой компьютер, отображаются имена и значки папок всех гибких и жестких дисков, Панели управления, принтеров. Для того чтобы начать работу с объектом, необходимо дважды щелкнуть по его значку.

Корзина

Корзина располагается на рабочем столе и предназначена для временного хранения удаленных файлов. Она позволяет восстановить файлы, удаленные по ошибке.

Чтобы просмотреть все файлы, хранящиеся в корзине, необходимо дважды щелкнуть по ее значку на рабочем столе. При этом появится окно папки. Строка меню имеет стандартный набор команд Файл, Правка, Вид. Внешний вид диалогового окна Корзина зависит от активной в данный момент команды в меню Вид. Если задействована команда таблица, то видно полное имя файла, его прежнее месторасположение, дата удаления, тип и размер.

Для восстановления и помещения в исходную папку файла/папки или ярлыка необходимо выделить имя восстанавливаемого файла и, зайдя в меню Файл, использовать команду Восстановить.

Панель задач

Полоска внизу экрана называется панель задач:

на ней отображаются:

1. Для того, чтобы быстрее начать работать в WINDOWS XP, щелкните по кнопке 

, указав на нее курсором и нажмите левую кнопку мыши. При этом распахнется главное меню.

2. В правом углу панели отображается текущее компьютерное время и активная (т. е. действующая в данный момент) раскладка клавиатуры или иначе – действующий язык (Ru – русский и En – английский) 

. Соответствующая раскладка клавиатуры – русская или английская позволяет вводить соответственно русский или английский текст. Для того чтобы сменить язык, достаточно щелкнуть мышкой по надписи и выбрать необходимый язык или нажать комбинацию клавиш (как правило, это Alt + Shift, т. е. нажать на Alt и удерживая ее, нажать на Shift).

3. Кроме этого, на панели задач отображаются все запущенные приложения (программы), и чтобы войти в нужное, достаточно щелкнуть по нему мышью 

. Если на кнопке в панели задач написано число и название программы, например 2 Microsoft Office, то с этой кнопкой связано несколько запущенных копий данной программы. Нажав такую кнопку, вы откроете список, в котором сможете выбрать нужную копию. Щелкнув мышью на одном из элементов списка, вы закроете список, при этом текущей станет выбранная копия данной программы.

Для того чтобы быстрее освоиться в новом для Вас мире, необходимо нажать кнопку F1 на клавиатуре. Это кнопка обычно отвечает за запуск помощи по интересующему вас приложению, например по WINDOWS XP. Далее следуйте всем инструкциям, появляющимся на экране компьютера.

Файловая система и структура

Информация в компьютере хранится в памяти или на различных носителях, таких как: гибкие и жесткие диски, или компакт-диски. При выключении питания компьютера информация, хранящаяся в памяти компьютера, теряется, а хранящаяся на дисках – нет. Для уверенной работы за компьютером следует знать основные принципы хранения информации на компьютерных дисках.

Вся информация, предназначенная для долговременного использования, хранится в файлах. Файл представляет собой последовательность байт, объединенных по какому-то признаку и имеющих имя. Система хранения и работы с файлами в компьютере называется файловой системой. Для удобства файлы хранятся в различных папках, которые расположены на дисках. В компьютере может быть установлено несколько дисков. Любой гибкий диск, жесткий диск, компакт-диск, цифровой видеодиск или сетевой диск мы будем называть просто диском, так как принципы организации хранения файлов на них идентичны. Каждому диску присваивается буква латинского алфавита от A до Z, причем существуют некоторые правила обозначения. Буквой A обозначается гибкий диск, буквой C – основной диск вашего компьютера, где расположена система Windows. Буквой D и последующими буквами обозначаются остальные диски. После буквы, обозначающей диск, ставится символ двоеточия «:», чтобы показать, что буква обозначает именно диск, например A: или C:. Кроме буквы, каждый диск имеет свое уникальное имя, также называемое меткой. Чаще всего при указании диска используется метка и буквенное обо-

значение в скобках. Например, надпись Main (C:) означает, что основной диск вашего компьютера имеет метку Main.

На каждом диске помещается множество различных файлов. Любой файл может располагаться как прямо на диске, так и в произвольной папке, которая в свою очередь также может располагаться в другой папке (рис. 5.1).

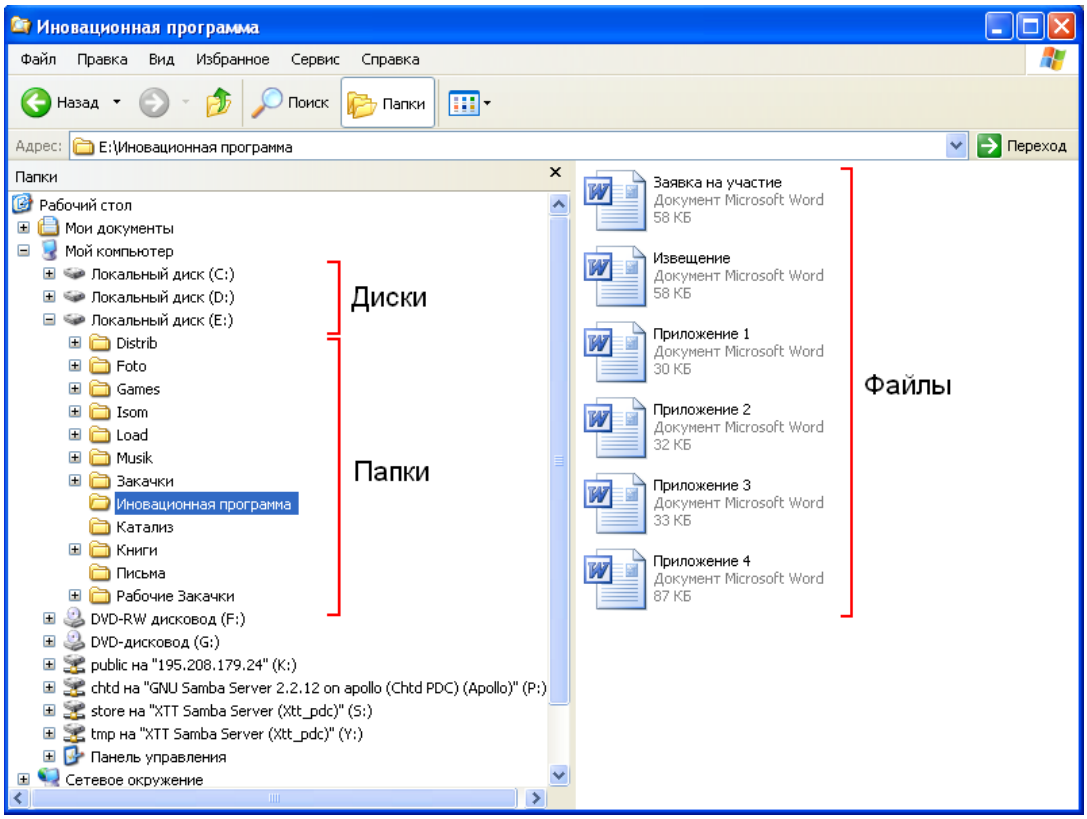


Рис. 5.1. Структура хранения информации на дисках

То, что файлы могут находиться в разных папках, позволяет расположить на диске несколько файлов с одинаковыми именами. Структура хранения информации на диске, при котором одни папки могут располагаться в других папках, называется иерархической или древовидной. Такая структура действительно похожа на реальное дерево, на котором каждый листок представляет собой отдельный файл, а ветка – папку. Листок может расти как непосредственно из ствола, так и из любой ветки. Возможно, что от ствола отходит одна ветка, от нее – другая, а уже на ней расположены листья. Чтобы однозначно определить конкретный файл, требуется задать его название и местоположение, то есть название диска и имена всех вложенных папок, в которых находится данный файл. Часто точное расположение файла на диске называют полным именем файла или путем к файлу.

При указании пути к файлу имена папок отделяются друг от друга и от имени диска с помощью символа обратной косой черты «\», например, **E:\Инновационная программа\Заявка на участие.doc**. Данная запись означает, что файл с именем **Заявка на участие.doc** расположен в папке **Инновационная программа**. Эта папка размещена на диске **E:**

Обратите внимание, что в рассмотренном примере имя файла содержит в себе символ точки и как бы состоит из двух частей – до точки и после нее. Часть имени, расположенная после точки, называется расширением и используется для обозначения вида информации, хранящейся в файле. Например, расширение **doc** обозначает текстовый файл, **wav** – файл, содержащий звуки, а **jpg** – изображение. В Windows XP многие расширения файлов не показываются, так что, скорее всего, в нашем примере файл будет называться просто **Заявка на участие**, но Windows будет знать, что работает с текстом.

Важным понятием в Windows XP является понятие ярлыка. На любой объект Windows можно сослаться из другого места. Такая ссылка и называется ярлыком. Например, в какой-то папке расположен часто использующийся рисунок. Для быстрого доступа к этому рисунку из разных мест можно поместить в эти места ярлыки, содержащие адрес реального местонахождения рисунка. Не требуется копировать программы и данные в разные папки, достаточно просто разместить ярлыки, ссылающиеся на нужный файл, в нескольких местах. Все эти ярлыки будут указывать на оригинальный файл. Удаление и перемещение ярлыка не влияет на расположение оригинального файла, поэтому использование ярлыков может обеспечить дополнительную защиту.

5.1. Основы работы с Microsoft Word

5.1.1. Запуск программы Word

При установке Word для Windows на ваш компьютер установочная программа может поместить значок редактора Word на *Рабочий стол*. Для запуска дважды щелкните мышью на значке запускаемой программы. С другой стороны, Word можно запустить через панель задач, нажав кнопку **Пуск**, выбрав в *Главном меню* раздел **Все программы**. Выбрать подменю **Microsoft Office** в нем программу **Microsoft Word**. Третий способ – запуск редактора Word с одновременной загрузкой редактируемого файла. Для этого, работая в окне со списком файлов (например, в программе **Проводник**), щелкните мышью на имя необходимого файла.

5.1.2. Элементы окна редактора Word

После того как программа Word для Windows запущена, на экране появляется окно программы. Некоторые элементы этого окна такие же, как и у всех других программ Windows, а некоторые присущи только программе Word для Windows.

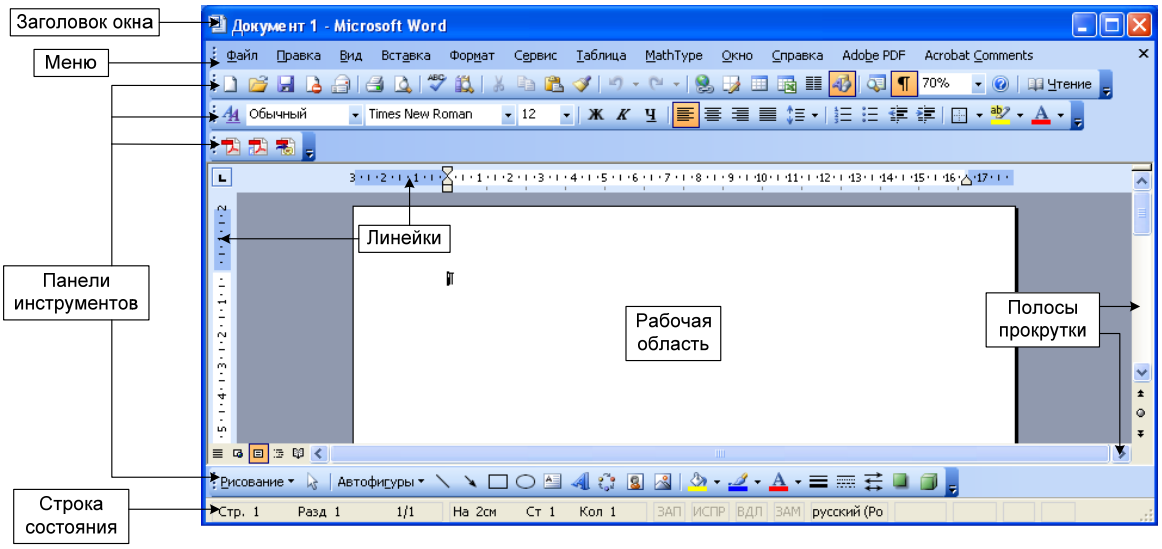


Рис. 5.2. Элементы окна редактора Word

1. Строка заголовка и меню.

Окно-приложение (окно, где представлена отдельная программа в среде Windows) содержит так называемую строку заголовка, которая находится наверху и обычно содержит название программы, в нашем случае это Microsoft Word, и название файла, который в данный момент обрабатывается. Когда вы первый раз запустили Word, то увидите в заголовке название **Документ 1**. Цифра 1 говорит, что Word может обрабатывать одновременно несколько документов, а название **Документ** присваивается автоматически каждому новому тексту, пока вы не сохранили его под другим именем.

В заголовке также расположены следующие кнопки: – кнопка Свернуть (окно), – кнопка Развернуть, (или – кнопка Уменьшить), – кнопка Выход. Щелкнув мышью на любой из них, вы можете вызвать соответствующую функцию.

2. Меню.

Под строкой заголовка обычно находится строка меню программы. Меню – это важнейшая часть окна-приложения, содержащая все необходимые команды. Для вызова соответствующей функции щелкните мышью на название раздела, а затем в открывшемся списке команд – на название необходимой команды.

Основные разделы меню программы Word следующие:

Файл – раздел, содержащий команды работы с файлами (запись и считывание с диска, распечатка на принтере и др.);

Правка – раздел с командами корректировки текста: копирование, вставка, поиск, замена и др.;

Вид – настройка режимов отображения документа на экране;

Вставка – вставка в текст Различных объектов Windows;

Формат – функции форматирования документа;

Сервис – проверка орфографии, настройка системы и другие полезные функции;

Таблица – команды работы с таблицами;

Окно – команды для работы с несколькими документами;

? – Справка.

3. Панели инструментов.

Под строкой меню находятся различные панели инструментов, которые содержат многочисленные кнопки и комбинированные поля. Простым щелчком мыши на кнопке того или иного инструмента вы можете задать команду для редактора Word (например,  – команда «Запись»).

4. Линейки.

Линейки находятся сверху и, если вы находитесь в режиме просмотра разметки страницы, слева от вашего документа. С помощью линеек вы можете изменить абзацный отступ, масштаб изображения страниц текста на экране, ширину колонок текста и размеры ячеек таблиц, а также устанавливать позиции табуляции в тексте.

5.1.3. Набор текста

Большую часть времени при работе с Word занимает ввод текста. Набираемый текст отображается в окне редактора. Место вставки очередного символа отмечается курсором. Как и в других программах – редакторах для набора текста – используются наборные клавиши клавиатуры, а также клавиши Shift для ввода прописных букв и символов верхнего регистра. При наборе текста нет необходимости следить, когда заканчивается строка, т. к. перенос строк и выравнивание текста Word осуществляет автоматически. Нажатие на клавишу Enter при вводе текста обычно означает начало нового абзаца.

Большинство документов, создаваемых с помощью Word, имеют больший размер, чем может поместиться в окне. Чтобы увидеть все части документа, необходимо использовать прокрутку текста. Для прокрутки текста с помощью клавиатуры, в основном, используются кла-

виши управления курсором, которые могут использоваться в сочетании с клавишей Ctrl.

Замечание. Никогда не нажимайте клавишу пробела для передвижения вправо. При нажатии на эту клавишу в текст документа вставляются символы пробела, которые обнаружатся при распечатке документа на принтере.

Для того чтобы переместить текстовый курсор с помощью мыши в видимой части документа, существует простое правило: установите указатель мыши на том месте, где вы хотите поставить текстовый курсор, и щелкните левой кнопкой мыши. Для прокрутки невидимой части документа используются полосы прокрутки: вертикальная – для перемещения текста вверх или вниз и горизонтальная – для перемещения текста влево или вправо. Щелкайте мышью на соответствующем месте полосы прокрутки для того, чтобы передвигаться по тексту.

При вводе текста допускается много ошибок. Но благодаря функциям исправления текста: удаление, добавление, перемещение – все ошибки можно исправить.

Удаление – это простейшая функция редактирования текста. Для удаления символа слева от курсора используется клавиша BackSpace (←), справа от курсора – Del. Для того чтобы удалить большой участок текста, необходимо отметить (выделить) удаляемый участок и один раз нажать клавишу Del.

Вообще, важнейшую концепцию работы Word можно выразить фразой: «Выделить и обработать». Сначала вы определяете, какой участок текста вы хотите обработать, а затем указываете, что надо сделать с этим участком текста: удалить, скопировать, форматировать, изменить шрифт и т. д. При выделении текста с помощью клавиатуры необходимо нажать клавишу Shift и, не отпуская ее, нажимать клавиши управления курсором (см. табл. 5.1.1). Чтобы сделать то же самое с помощью мыши, поставьте указатель мыши слева от первого символа, нажмите левую кнопку мыши, и удерживая ее, передвигайте указатель до тех пор, пока не отметится последний символ. Затем отпустите левую кнопку.

Таблица 5.1.1

Передвижение курсора с помощью клавиш управления курсором

Клавиша	Действие	Действие в сочетании с Ctrl
Home	В начало строки	В начало документа
↑	На строку вверх	На абзац вверх
PgUp	На окно вверх	На страницу вверх
←	На символ влево	На слово влево
→	На символ вправо	На слово вправо
End	В конец строки	В конец документа
↓	На строку вниз	На абзац вниз
PgDn	На окно вниз	На страницу вниз

Если вы случайно удалили нужный вам участок текста, то не беспокойтесь – Word может вернуть ваш текст назад с помощью комбинации клавиш Ctrl-Z. Для восстановления текста можно использовать кнопку  . Щелкая мышью на закругленной стрелке, вы можете последовательно отменять команды, а щелкнув мышью на стрелке, направленной вниз, вы можете выбрать из списка команду, результат выполнения которой вы хотите отменить.

Существуют два режима обработки текста: вставка и замена. Во время режима вставки, вводимые вами символы, слова и предложения будут вставляться в текст. Сам текст при этом будет смещаться вправо, освобождая место для новых символов. При режиме замены вводимые символы будут вставляться в текст вместо уже существующих символов в позиции текстового курсора. Существующие же символы будут автоматически удаляться, т. е. новый текст как бы вводится поверх старого. Для переключения режима вставки / замещения используется клавиша Insert (Ins). Следует отметить, что во время режима замены символов символ конца абзаца (¶) не заменяется, а отодвигается вправо.

Следующая наиболее часто используемая процедура – добавление текста в документ. Например, для начала ввода нового абзаца

- передвиньте текстовый курсор на символ конца предыдущего абзаца;
- включите режим вставки, если необходимо;
- нажмите клавишу Enter и вводите текст.

Каждый шрифт содержит определенный набор символов, причем не все из них могут быть введены в текст с помощью нажатия какой-либо наборной клавиши. Однако такие символы можно вставить в текст с помощью пункта меню **Вставка→Символ**. (В таблице символов необходимо выбрать нужный символ и нажать клавиши Вставить и Закрыть). По умолчанию предлагается вставлять символы из шрифта Simbol, содержащий греческие буквы и математические символы.

После того как мы рассмотрели команды удаления и добавления текста, рассмотрим копирование и перемещение участков текста. Для перемещения или копирования участков текста с помощью Word существуют три различные техники: новая (Drag - and - Drop: «Перетащить и оставить»), которая особенно удобна для перемещения текста на небольшие расстояния в пределах видимого текста, техника, использующая правую кнопку мыши, и возможность копирования текста с использованием буфера обмена Windows.

Суть первого метода заключается в следующем:

- выделите текст, который вы хотите переместить или скопировать;
- установите указатель мыши на выделенном тексте, нажмите левую кнопку мыши и удерживайте ее – в этот момент вы как бы «схватили» выделенный участок текста. При этом изменится форма текстового курсора и указателя мыши. Текстовый курсор примет форму штриховой вертикальной линии, а на нижнем конце указателя мыши появится небольшой прямоугольник;
- нажмите клавишу Ctrl, если вы хотите скопировать выделенный участок текста. При этом на верху указателя мыши появится знак «+»;
- перемещая указатель мыши, установите штриховой текстовый курсор в ту позицию, куда хотите переместить или скопировать выделенный участок текста.

Описанный выше метод удобен для перемещения или копирования текста на небольшие расстояния в пределах видимой части документа. Однако его можно использовать и для перемещения или копирования по всему документу. Чтобы, например, перенести участок текста в часть документа, которая невидима в данный момент на экране, «схватите» участок текста и, удерживая левую кнопку мыши, переместите указатель мыши к верхней или нижней границе документа, слегка «заехав» на эту границу. При этом текст в окне начнет прокручиваться вверх или вниз в зависимости от выбранной границы окна.

При перемещении или копировании участков текста на большие расстояния, т. е. за пределы видимой части документа, удобнее применять следующую технику:

- выделите необходимый участок текста;
- прокрутите текст в окне так, чтобы часть, в которую вы хотите перенести или скопировать текст, стала видимой;
- нажмите клавишу Ctrl и удерживайте ее;
- нажмите дополнительно клавишу Shift, если хотите скопировать текст;
- установите указатель мыши в позицию текста, куда вы хотите вставить ваш текст, и нажмите правую кнопку мыши.

Для перемещения или копирования участков текста может быть использован так называемый буфер обмена Windows. Буфер обмена – это участок памяти, в которой временно помещается вырезанный или скопированный участок текста или графики. Содержимое буфера может быть вставлено в эту же программу или в другую. При помещении в буфер нового участка текста или графики старое содержимое буфера обмена теряется. Ниже рассматриваются клавиши и кнопки для копирования и перемещения участков текста с помощью буфера обмена.

Кнопка	Команда меню	Комбинация клавиш
	Правка\Вырезать	Ctrl – X или Shift – Del
	Правка\Копировать	Ctrl – C или Ctrl – Insert
	Правка\Вставить	Ctrl – V или Shift – Insert

Шаги для копирования (перемещения) участков текста:

- выделите текст;
- скопируйте текст в буфер обмена (или перенесите текст в буфер обмена с помощью команды «Вырезать»);
- установите текстовый курсор в нужное место;
- вставьте текст из буфера обмена.

Достоинство последнего метода заключается в том, что текст можно копировать и переносить из одного окна редактора Word в другое.

5.1.4. Сохранение и загрузка документов

Для сохранения нового документа при работе с Word существует несколько путей:

- Выбрать команду меню **Файл→Сохранить**.
- Нажать комбинацию клавиш Ctrl-S (Shift-F12).
- Нажать кнопку с изображением дискеты на панели инструментов  .

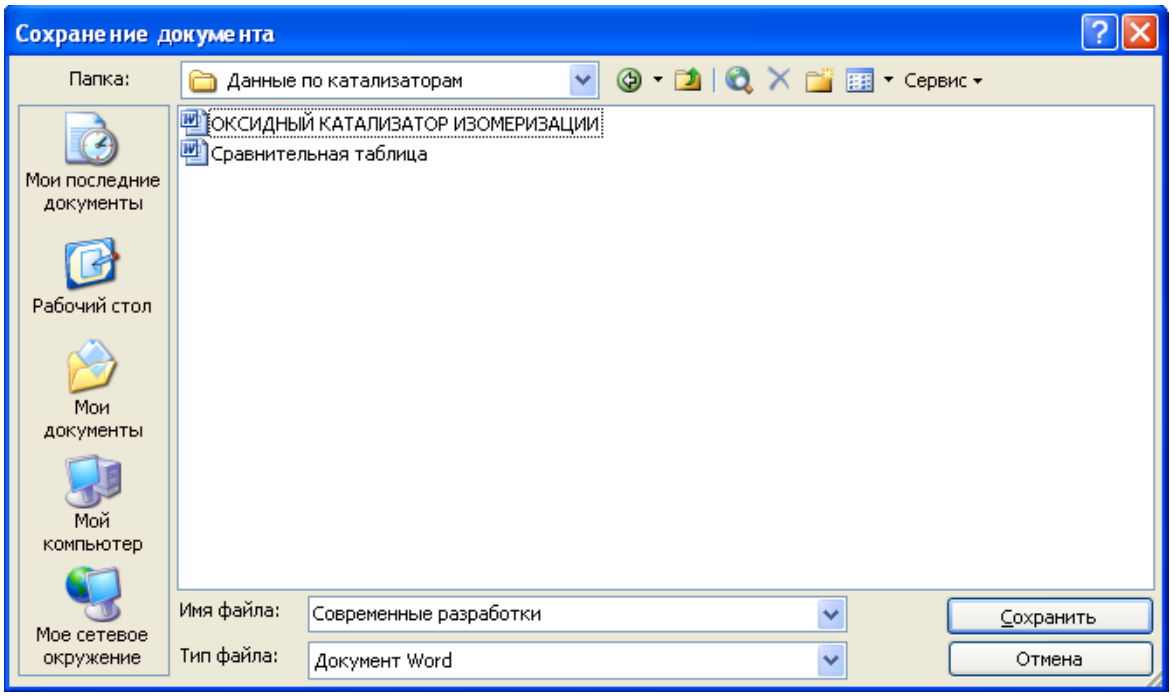


Рис. 5.3. Сохранение документа

После этого на экране появится диалоговое окно (рис. 5.3). В поле «Имя файла» вы должны указать имя документа, причем Word автоматиче-

ски присваивает расширение **.doc**, если вы не указали другого. Рекомендуем не изменять расширение файла для сохранения документа, а пользоваться тем, которое Word предлагает по умолчанию. Наряду с именем файла, можно выбрать каталог (папку), в который будет записан файл (поле «Папка»). Для этого необходимо щелкнуть мышью на кнопку **Мой компьютер** и выбрать нужный вам диск и каталог с помощью двойного щелчка мыши на названия каталогов. После ввода необходимой информации для начала записи файла на диск нажмите кнопку «Сохранить», если вы передумали сохранять файл, нажмите кнопку «Отмена» или  .

Следует отметить, что при использовании вышеуказанных команд данное диалоговое окно появляется, если файл сохраняется первый раз и вы еще не назначили ему имя. При повторном их использовании это окно открываться не будет, а измененный текст будет записываться в файл под прежним именем. Чтобы записать файл под новым именем, следует использовать команду **Файл→Сохранить как**.

Наряду с созданием новых документов, в своей работе вы будете часто редактировать старые, т. е. те документы, которые хранятся на диске. Чтобы загрузить документ в Word для продолжения редактирования, необходимо использовать команду меню **Файл→Открыть** или нажать комбинацию клавиш Ctrl-O. Также вы можете это сделать с помощью кнопки с изображением папки на панели инструментов  . В открывшемся диалоговом окне ввод имени файла и каталога аналогичен тому, как это делалось в случае сохранения документа.

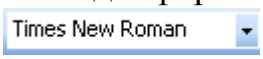
Вы можете создать новый документ с помощью команды меню **Файл→Создать** или кнопки с изображением чистого листа бумаги  . Распечатать файл на принтере можно с помощью команды **Файл→Печать** или нажав комбинацию клавиш Ctrl-P, или с помощью кнопки  на панели инструментов.

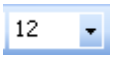
5.1.5. Основы форматирования текста

С помощью форматирования вы создаете внешний вид вашего документа; выбирая шрифты и размеры символов, подчеркивая или выделяя курсивом слова или предложения, вы выполняете форматирование символов. Вы можете выделить участок текста, к которому будут относиться ваши команды. Таким участком может быть абзац. Вы можете выравнивать абзац по правой или левой границе или по центру страницы, а также заключить абзац в рамку. Такое форматирование называется форматированием абзацев. Для вашего документа нужно задать формат бумаги – любой из тех, которые может обрабатывать ваш принтер. Вы устанавливаете размер вашей страницы и отступы текста от краев бума-

ги для того, чтобы Word правильно расположил текст на листе. Данный вид форматирования называется форматированием документов. Обилие возможностей форматирования может запутать начинающего, поэтому рассмотрим только основные возможности.

Форматирование символов

- Выделите текст, который вы желаете форматировать.
- Выберите команду **Формат→Шрифт**. В открывшемся диалоговом окне вы можете выбрать шрифт (поле **Шрифт**), размер букв (поле **Размер**), стиль написания букв: обычный, курсив, полужирный (поле **Начертание**), цвет символов, установить способ подчеркивания и т. д. Наиболее используемые команды форматирования символов вынесены на панель инструментов: 

– выбор шрифта, 

– установка размеров букв, 

– установка (отмена) режима начертания символов: полужирный, курсив, подчеркивание.
- Щелкните мышью в любом месте документа, чтобы снять выделение.

Для форматирования выделенных символов вы можете пользоваться только клавиатурой, нажимая соответствующие комбинации клавиш:

Ctrl – Shift – F	для выбора шрифта
Ctrl – Shift – P	для выбора размера шрифта
Ctrl – B	полужирный шрифт
Ctrl – I	Курсив
Ctrl – U	Подчеркивание

Форматирование абзацев

Как вы уже знаете, абзацем называется участок текста между двумя маркерами (символами) абзаца, которые вставляются в текст с помощью клавиши Enter. Вы можете сделать видимыми маркеры абзаца с помощью кнопки ¶. Чтобы отформатировать абзац, достаточно поместить текстовый курсор в любое место абзаца и выполнить команду форматирования. Это является существенным отличием от форматирования символов, когда необходимо сначала выделить участок текста. Если вы хотите отформатировать сразу несколько абзацев, сначала выделите их, причем достаточно выделять не целый абзац, а только его часть.

Форматируется абзац с помощью команды **Формат→Абзац**. В открывшемся диалоговом окне можно задать: отступы текста от левого и правого края (поля Слева, Справа), отступы перед и после абзаца (поля Перед, После), красную строку (поле Первая строка), межстрочный интервал и выравнивание текста по левому краю, по правому, по центру или по ширине. Команды выравнивания представлены на панели инст-

рументов кнопками  . Кроме того, отступы слева и справа и отступ первой строки можно задавать с помощью бегунков на горизонтальной линейке, расположенной над текстовым окном. Для этого достаточно «схватить» мышкой бегунок и передвинуть в нужную позицию.

Установка параметров страницы. Для установки размеров и границ страницы используется диалоговое окно Параметры страницы, которое открывается с помощью команды меню **Файл→Параметры страницы**. Выбрав вкладку **Поля**, вы сможете установить отступы от всех краев листа бумаги, даже с учетом отступа под переплет, что важно при печати на двух сторонах листа бумаги. Чтобы установить размер самого листа бумаги и его ориентацию при печати, необходимо выбрать вкладку **Размер бумаги**.

5.1.6. Создание таблиц

Для создания таблиц Word предоставляет две возможности:

- Создать пустую таблицу и затем заполнить ее графы.
- Преобразовать в таблицу существующий текст.

Здесь мы рассмотрим только первую возможность. Чтобы создать таблицу, необходимо нажать Таблица→Вставить→Таблица. В появившемся окне необходимо указать количество столбцов в требуемой таблице. Новые строки можно легко добавить при помощи меню Таблица или (что более удобно) поставить курсор в самую правую и нижнюю ячейку таблицы после чего нажать на клавиатуре кнопку Tab. Перемещаться по ячейкам можно при помощи кнопок управления курсором (см. табл. 5.1.1), кнопки Tab и манипулятора (например, мыши).

Для того, чтобы выделить отдельные символы в таблице, вы можете использовать комбинацию клавиш Shift-<клавиши управления курсором>. Если вы выделите маркер конца таблицы ⌑, то автоматически выделится вся ячейка таблицы. С помощью комбинации клавиш Alt-Num 5 (клавиша 5 на цифровом блоке клавиатуры) вы можете выделить всю текущую таблицу в том случае, если выключен светодиод NumLock; если он включен, используйте комбинацию клавиш Shift-Alt-Num 5.

5.1.7. Формульный редактор

Редактор MicroSoft Word позволяет вставлять в текст различные объекты, созданные другими программами в среде Windows. К их числу относятся математические формулы, создаваемые программой MathType Equation.

Для вставки формулы в текст необходимо войти в формульный редактор. Для этого нужно выбрать в меню Вставка пункт Объект и в от-

крывшемся списке объект MathType Equation, либо скопировать уже существующую формулу, а затем отредактировать ее дважды щелкнув мышью на этой формуле.

При входе в формульный редактор на экране появляются меню и панель инструментов формульного редактора, а сама формула или место под формулу выделяется штрихованной рамкой.

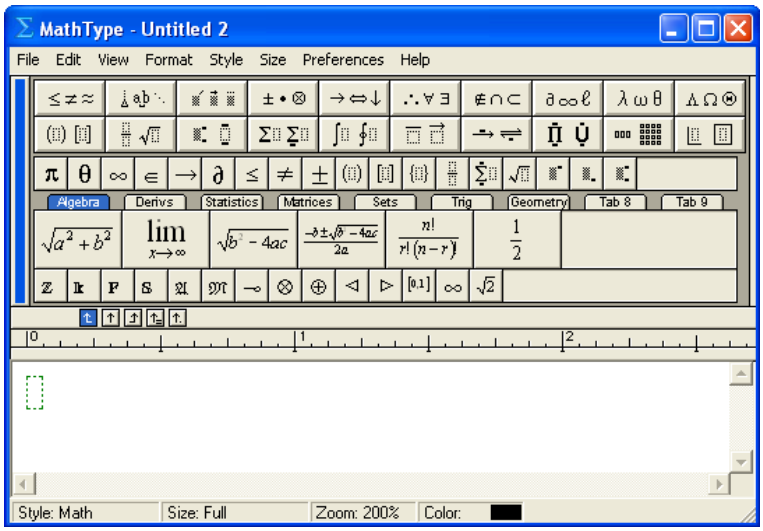


Рис. 5.4. Формульный редактор MathType Equation

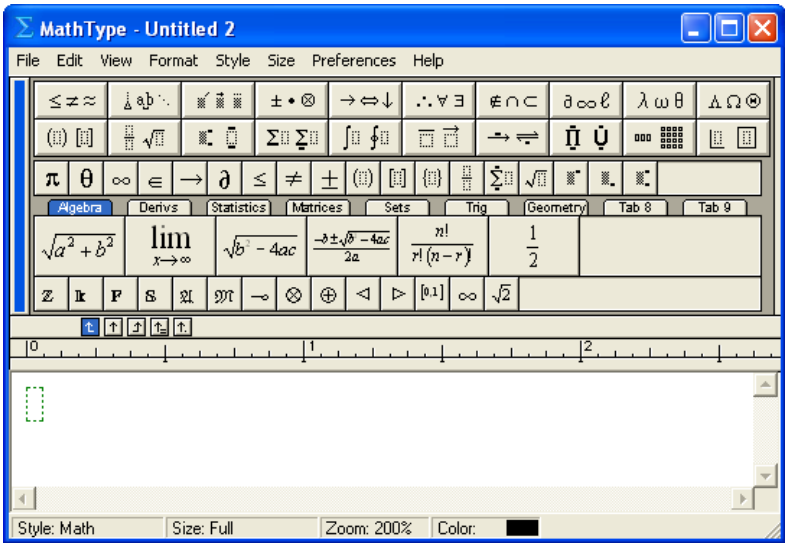


Рис. 5.4. Формульный редактор MathType Equation

Текст формулы вводится с помощью клавиатуры (цифры, буквы латинского и русского алфавитов, составляющие имена переменных и функций), а также с помощью панели инструментов (специальные символы: греческие буквы, скобки, надстрочные и подстрочные символы, знаки сумм, интегралов и др.). Вводимые с помощью панели инструмен-

тов специальные символы могут иметь одно или более дополнительных полей ввода. В этом случае на экране появляется изображение символа, и указываются пустые поля ввода. В них можно вписать любой текст и формулу, используя, если необходимо другие специальные символы.

Например, необходимо записать формулу $y = \ln\left|\frac{1-x}{1+x^2}\right|$.

Рассмотрим этапы ввода формулы с использованием формульного редактора:

1) $y = \ln$ — Вводим с клавиатуры начало формулы

2) $y = \ln|\ |$ — Вводим с помощью панели инструментов специальный символ «модуль»

3) $y = \ln\left|\frac{}{}\right|$ — Вводим в поле ввода спецсимвола «модуль» другой спецсимвол «дробь»

4) $y = \ln\left|\frac{1-x}{1+x}\right|$ — Заполняем поля ввода «дроби»

5) $y = \ln\left|\frac{1-x}{1+x}\right|$ — Добавляем с помощью панели инструментов поле ввода надстрочных символов

И в итоге: $y = \ln\left|\frac{1-x}{1+x^2}\right|$

Для указания места вставки в формульном редакторе существует курсор. Курсор состоит из вертикальной черты, указывающей место вставки и горизонтальной черты, указывающей к какому полю ввода относится вводимый текст. Передвигать курсор можно с помощью клавиш со стрелками, а также с помощью мыши.

Удалять текст можно с помощью клавиш BackSpace (Забой), и Del. Клавиша BackSpace удаляет символы слева от курсора, причем символы введенные с клавиатуры удаляются однократным нажатием этой клавиши, специальные символы – двукратным, после первого нажатия выделяется спецсимвол и его поле (или поля) ввода, после второго – происходит удаление. Клавиша Del удаляет только введенные с клавиатуры символы, расположенные справа от курсора. Специальные символы этой клавишей не удаляются.

Фрагмент формулы можно выделить (клавиши Shift-<стрелки>), а затем

- удалить (Del),
- скопировать Ctrl – Ins.

Вставить фрагмент формулы можно, подведя курсор к месту вставки и нажав Shift-Ins.

Для выхода из формульного редактора достаточно щелкнуть мышью вне заштрихованной области, в которой вводится формула или нажать Esc.

5.2. Построение графиков с использованием Microsoft Excel

Microsoft Excel так же, как и Word, входит в состав программного продукта Microsoft Office. Основное его назначение – это математическая обработка табличных данных и различные способы их графического представления: графические зависимости, диаграммы и т. д. Возможности этой программы очень велики, мы рассмотрим ее применение на примере построения графика по табличным данным.

При входе в Excel на экране появляется бланк электронной таблицы. Строки таблицы нумеруются арабскими цифрами, а столбцы буквами латинского алфавита. Кроме того, в верхней части экрана располагаются меню и панель инструментов. (Способы работы с ними такие же, как и в Word).

Рассмотрим конкретный пример. Пусть даны зависимости концентраций веществ A, B и C в реакторе от времени.

Время, час	Ca	Cb	Cc
0	10,000	6,000	0,000
0,1	7,000	3,000	3,000
0,2	5,950	1,950	4,050
0,3	5,370	1,370	4,630
0,4	5,002	1,002	4,998
0,5	4,751	0,751	5,249

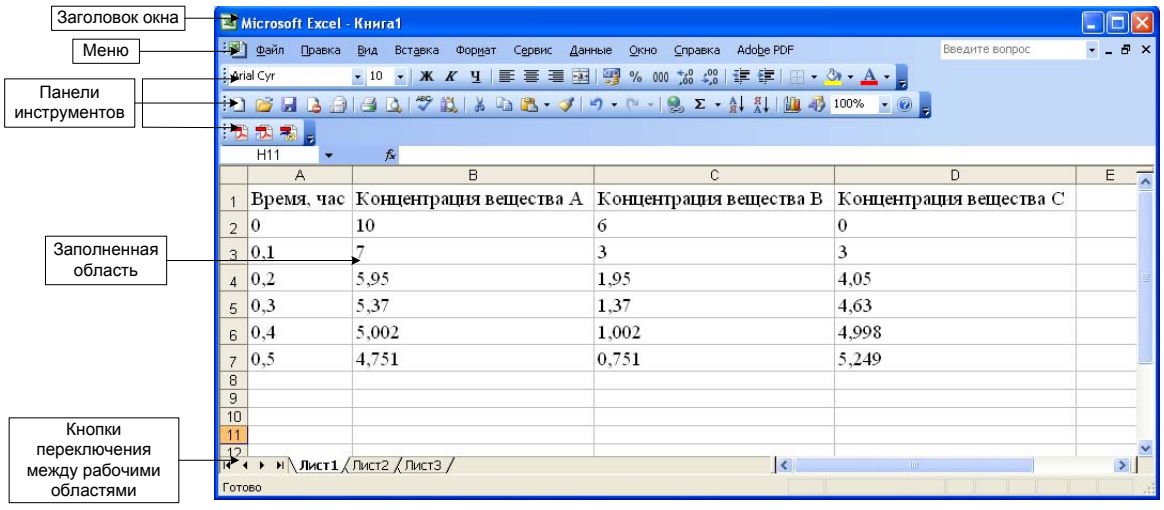


Рис. 5.5. Элементы окна редактора Execl

Введем таблицу значений (рис. 5.5). Обратите внимание, что при наборе чисел используется запятая, а в первой строке – комментарии для легенды.

Далее выделяем те ячейки с данными, которые будут использованы для построения графика. Нажимаем кнопку  **(«Мастер диаграмм»).**

При этом появляется диалоговое окно, с помощью которого можно выбрать тип диаграммы. Для построения графика рекомендуется выбрать тип **«Точечная диаграмма»**. И далее следует ряд диалоговых окон, при помощи которых задаются все необходимые параметры для построения диаграммы: заголовки диаграммы и осей координат, наличие осей и линий сетки, наличие легенды диаграммы и ее расположение.

Диаграмму рекомендуется поместить на отдельном листе.

Для изменения каких-либо параметров построенной диаграммы необходимо нажать на диаграмму правой кнопкой мыши.

В результате приводим диаграмму к требуемому виду (рис. 5.6).

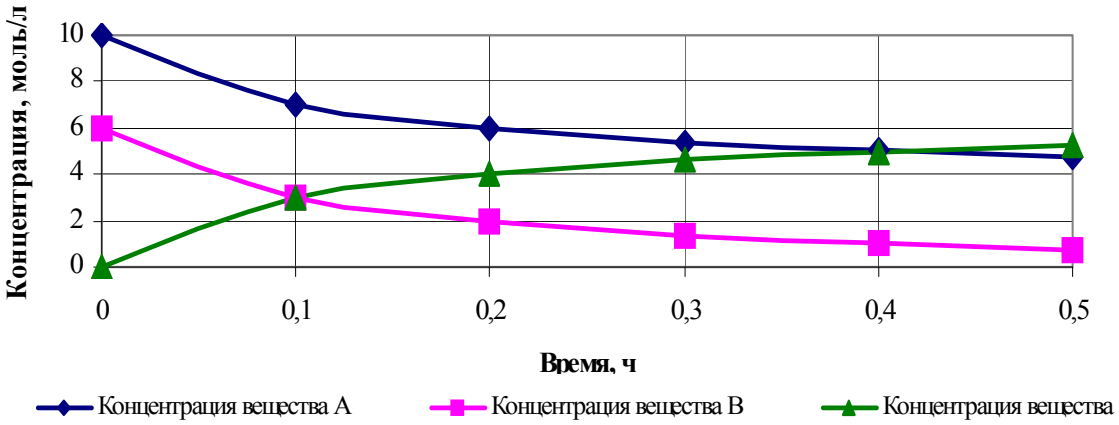


Рис. 5.6. Зависимость концентраций веществ А, В, С от времени

Большим достоинством использования среды Exel является то, что помимо построения обыкновенных графиков, можно строить так называемую **линию тренда**. Линии тренда позволяют графически отображать тенденции данных и прогнозировать их дальнейшие изменения. Подобный анализ называется также регрессионным анализом. Используя регрессионный анализ, можно продлить линию тренда в диаграмме за пределы реальных данных для предсказания будущих значений. Например, приведенный ниже рис. 5.7, использует простую линейную линию тренда, которая является прогнозом на полтора месяца вперед, для демонстрации тенденции коксонакопления на катализаторе.

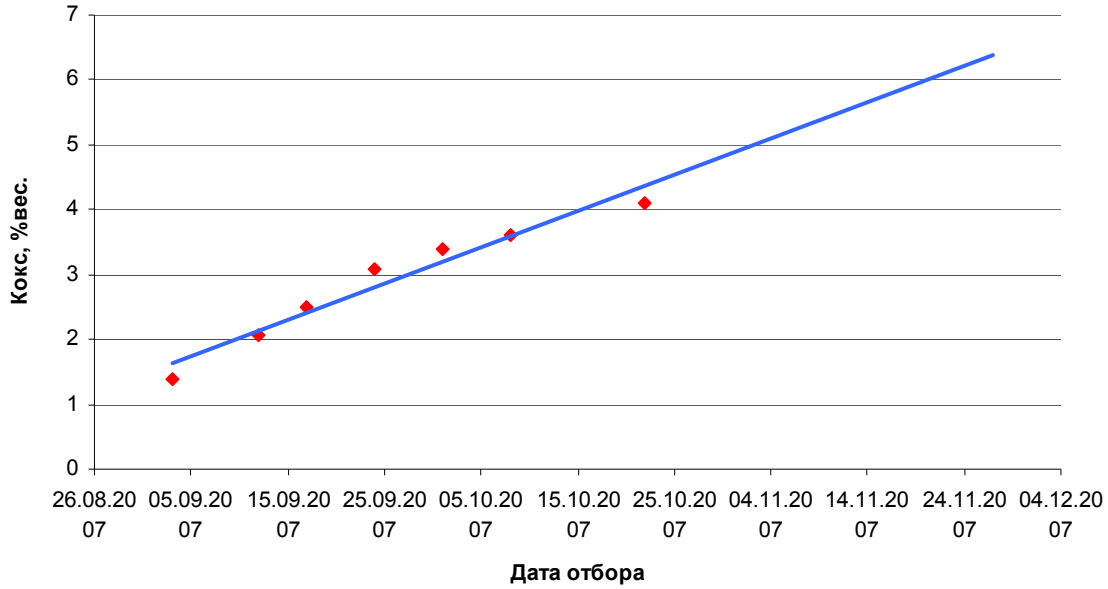


Рис. 5.7. Зависимость количества кокса на катализаторе от срока эксплуатации

Для того чтобы провести линии тренда нужно:

1. Выберите ряд данных (Ряд данных – набор связанных между собой элементов данных, отображаемых на диаграмме. Каждому ряду данных на диаграмме соответствует отдельный цвет или способ обозначения, указанный на легенде диаграммы. Диаграммы всех типов, кроме круговой, могут содержать несколько рядов данных), к которому нужно добавить линию тренда.
2. Выберите команду **Добавить линию тренда** в меню **Диаграмма**.
3. На вкладке **Тип** выберите нужный тип регрессионной линии тренда.

***Примечание*:**

В поле **Построен на ряде** перечислены все ряды данных диаграммы, поддерживающей линии тренда. Для добавления линии тренда к другим рядам выберите нужное имя в поле, а затем выберите нужные параметры.

Формулы для вычисления линий тренда

Линейная

Используется для аппроксимации данных по методу наименьших квадратов в соответствии с уравнением:

$$y = mx + b$$

где m – угол наклона и b – координата пересечения оси абсцисс.

Полиномиальная

Используется для аппроксимации данных по методу наименьших квадратов в соответствии с уравнением:

$$y = b + c_1 x + c_2 x^2 + c_3 x^3 + \ldots + c_6 x^6$$

где b и $c_1 \ldots c_6$ – константы.

Логарифмическая

Используется для аппроксимации данных по методу наименьших квадратов в соответствии с уравнением:

$$y = c \ln x + b$$

где c и b – константы, ln – функция натурального логарифма.

Экспоненциальная

Используется для аппроксимации данных по методу наименьших квадратов в соответствии с уравнением:

$$y = ce^{bx}$$

где c и b – константы, e – основание натурального логарифма.

Степенная

Используется для аппроксимации данных по методу наименьших квадратов в соответствии с уравнением:

$$y = ce^{b}$$

где c и b – константы.

Значение R-квадрат

$$R^2 = 1 - \frac{SSE}{SST}$$

где $SSE = \sum (Y_i - \hat{Y}_i)^2$ и $SST = (\sum Y_i^2) - \frac{(\sum Y_i)^2}{r}$

Значение R в квадрате – число от 0 до 1, которое отражает близость значений линии тренда к фактическим данным. Линия тренда наиболее соответствует действительности, когда значение R в квадрате близко к 1. Оно также называется квадратом смешанной корреляции. Для логарифмической, степенной и экспоненциальной линий тренда в Microsoft Excel используется несколько видоизмененная модель регрессии.

6. ОСНОВЫ РАБОТЫ В СРЕДЕ DELPHI

6.1. Знакомство со средой Delphi

Среда Delphi – это сложный механизм, обеспечивающий высокоэффективную работу программиста. Визуально она реализуется несколькими одновременно раскрытыми на экране окнами. Окна могут перемещаться по экрану, частично или полностью перекрывая друг друга.

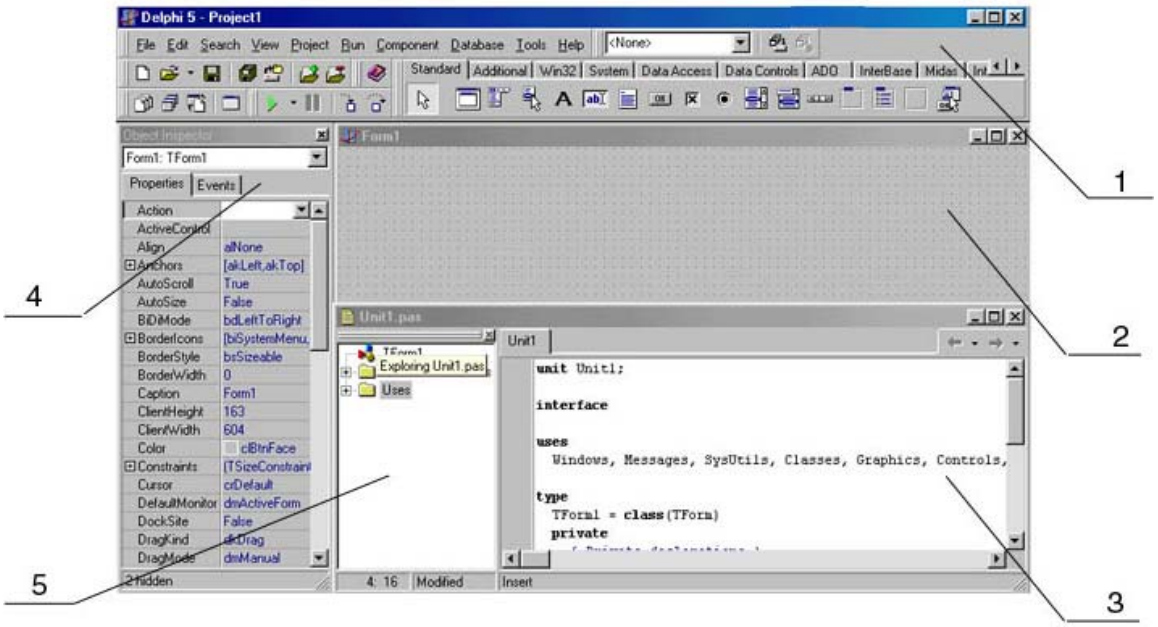


Рис. 6.1. Наиболее важные окна Delphi: 1 – главное окно; 2 – окно формы; 3 – окно кода программы; 4 – окно Инспектора Объектов; 5 – окно браузера

На рис. 6.1 изображены четыре наиболее важных окна Delphi: главное окно (оно имеет заголовок Delphi 5 – Project1), окно формы (заголовок Form1), окно Инспектора Объектов (Object Inspector) и окно кода программы (Unit1. pas). Слева в окне кода располагается вспомогательное окно браузера (на рисунке помечено цифрой 5), с помощью которого можно просматривать структуру всех объявлений в окне кода.

Добиваться максимального сходства того, что вы видите на экране вашего ПК, с изображением, показанным на рис. 4.1 необязательно: расположение и размеры окон никак не влияют на их функциональность.

6.1.1. Главное окно

Главное окно осуществляет основные функции управления проектом создаваемой программы. Это окно всегда присутствует на экране и занимает его самую верхнюю часть.

Связано это с функциональностью главного окна: с одной стороны, оно несет в себе элементы, которые всегда должны быть под рукой у программиста, с другой – окно не должно отнимать у остальных окон *Delphi* значительного пространства экрана. Минимизация главного окна приводит к исчезновению с экрана других окон *Delphi*, а его закрытие означает окончание работы программиста с системой программирования.

В главном окне располагается главное меню *Delphi*, набор пиктографических командных кнопок и палитра компонентов.

Главное меню содержит все необходимые средства для управления проектом. Все опции главного меню представляют собой опции-заголовки, открывающие доступ к выпадающим меню второго уровня.

Все элементы главного окна располагаются на специальных панельках, в левой части которых имеются кнопки управления, позволяющие с помощью мыши перетаскивать панельки с помещенными на них элементами. Любую панельку (кроме главного меню) можно убрать из окна (сделать ее невидимой) или «пустить плавать» по экрану в отдельном окне. Для изменения состава показываемых на панельке кнопок нужно предварительно щелкнуть по ней правой кнопкой мыши. В появившемся после этого окне вспомогательного меню перечислены названия всех панелек и указан их статус (отмеченные флажками панельки видны в главном окне; если отметку убрать, панелька исчезнет). После выбора *Customize* (Настройка) появится окно настройки. Теперь можно «стаскивать» с панелек ненужные кнопки, выбирать из списка в окне *Commands* (закладка *Commands*) нужные кнопки и перетаскивать их на экран.

Пиктографические кнопки

Пиктографические кнопки открывают быстрый доступ к наиболее важным опциям главного меню. По функциональному признаку они разделены на 6 групп. Каждая группа занимает отдельную панельку. В таблице описаны команды, реализуемые стандартным набором пиктографических кнопок.

Кнопка	Реализуемое кнопкой действие
Группа Standard	
	Открывает доступ к Репозиторию Объектов. Эквивалент опции *File \ New*
	Открывает существующий файл. Эквивалент опции *File \ Open File*
	Сохраняет файл на диске. Эквивалент опции *File \ Save File* (клавиши быстрого доступа *Ctrl-S*)
	Сохраняет все файлы проекта. Эквивалент опции *File \ Save All*

Кнопка	Реализуемое кнопкой действие
	Открывает созданный ранее проект программы. Эквивалент опции *File* \ *Open Project* (клавиши быстрого доступа *Ctrl-F11*)
	Добавляет новый файл к проекту. Эквивалент опции *Project* \ *Add to Project* (клавиши быстрого доступа *Shift-F11*)
	Удаляет файл из проекта. Эквивалент опции *Project* \ *Remove from Project*
Группа View	
	Выбирает модуль из списка модулей, связанных с текущим проектом. Эквивалент опции *View* | *Units* (клавиши быстрого доступа *Shift-F12*)
	Выбирает форму из списка форм, связанных с текущим проектом. Эквивалент опции *View* | *Forms* (клавиши быстрого доступа *Ctrl-F12*)
	Переключает активность между окном формы и окном кода программы. Эквивалент опции *View* \ *Toggle Form/Unit* (клавиша быстрого доступа *F12*)
	Создает новую форму и добавляет ее к проекту. Эквивалент опции *File* \ *New*|*Form*
Группа Debug	
	Компилирует и выполняет программу. Эквивалент опции *Run* \ *Run* (клавиша быстрого доступа *F9*)
	Реализует паузу в работе отлаживаемой программы. Эквивалент опции *Run*| *Program Pause*
	Осуществляет пошаговую трассировку программы с прослеживанием работы вызываемых подпрограмм. Эквивалент опции *Run* | *Trace Into* (клавиша быстрого доступа *F7*)
	Осуществляет пошаговую трассировку программы, но не прослеживает работу вызываемых подпрограмм. Эквивалент опции *Run* \ *Step Over* (клавиша быстрого доступа *F8*)
Группа Custome	
	Открывает доступ к встроенной справочной службе. Эквивалент опции *Help*
Группа Desktops	
	Список выбора возможных вариантов настройки остальных окон *Delphi*
	Сохраняет текущую настройку окон *Delphi*
	Выбирает настройку окон, соответствующую отладочному режиму

6.1.2. Окно формы

Окно формы представляет собой проект *Windows*-окна будущей программы. Вначале это окно пусто. Точнее, оно содержит стандартные для *Windows* интерфейсные элементы – кнопки вызова системного меню, максимизации, минимизации и закрытия окна, полосу заголовка и

очерчивающую рамку. Вся рабочая область окна обычно заполнена точками координатной сетки, служащей для упорядочения размещаемых на форме компонентов (вы можете убрать эти точки, вызвав с помощью меню *Tools* | *Environment* Options соответствующее окно настроек и убрав флажок в переключателе *Display Grid* на окне, связанном с закладкой *Preferences*).

Программист «достает» из палитры компонентов нужный компонент и размещает его на окне формы, постепенно заполняя форму интерфейсными элементами, т. е. в любой момент времени программист контролирует содержание окна создаваемой программы и может внести в него необходимые изменения.

6.1.3. Окно Инспектора Объектов

Любой размещаемый на форме компонент характеризуется некоторым набором параметров: положением, размером, цветом и т. д. Часть этих параметров, например, положение и размеры компонента, программист может изменять, манипулируя с компонентом в окне формы. Для изменения других параметров предназначено окно Инспектора Объектов. Это окно содержит две страницы – *Properties* (Свойства) и *Events* (События). Страница *Properties* служит для установки нужных свойств компонента, страница Events позволяет определить реакцию компонента на то или иное событие. Совокупность свойств отображает видимую сторону компонента: положение относительно левого верхнего угла рабочей области формы, его размеры и цвет, шрифт и текст надписи на нем и т. п.; совокупность событий – его поведенческую сторону: будет ли компонент реагировать на щелчок мыши или на нажатие клавиш, как он будет вести себя в момент появления на экране или в момент изменения размеров окна и т. п.

Каждая страница окна Инспектора Объектов представляет собой двухколончатую таблицу, левая колонка которой содержит название свойства или события, а правая – конкретное значение свойства или имя подпрограммы, обрабатывающей соответствующее событие.

Строки таблицы выбираются щелчком мыши и могут отображать простые или сложные свойства. К простым относятся свойства, определяемые единственным значением – числом, строкой символов, значением True (Истина) или False (Ложь) и т. п. Например, свойство *Caption* (Заголовок) представляется строкой символов, свойства *Height* (Высота) и *Width* (Ширина) – числами, свойство *Enabled* (Доступность) – значениями True или False. Сложные свойства определяются совокупностью значений. Слева от имени таких свойств указывается значок «+», а щелчок мышью по этому символу приводит к раскрытию списка состав-

ляющих сложного свойства. Чтобы закрыть раскрытый список, нужно щелкнуть по значку «-» сложного свойства.

В верхней части окна Инспектора Объектов располагается раскрывающийся список всех помещенных на форму компонентов. Поскольку форма сама по себе является компонентом, ее имя также присутствует в этом списке.

В локальном меню окна, которое появляется после щелчка по нему правой кнопкой, имеется ряд опций, позволяющих настроить окно. В частности, после выбора *Stay on Top* окно Инспектора Объектов будет «всплывать» над всеми другими окнами независимо от его активности. Такое состояние окна удобно при частом его использовании, например, при конструировании сложной формы, содержащей множество компонентов. Если выбрать в локальном меню опцию *Arrange* и затем *by Category*, все строки окна Инспектора Объектов будут представлять собой раскрывающиеся списки свойств, упорядоченные по категориям.

Щелчок по знаку «+» слева от категории приводит к раскрытию списка. Любые категории можно сделать невидимыми. Для этого нужно в локальном меню выбрать *View* и затем в дополнительном меню убрать флажок слева от категории. Если вы случайно или намеренно сделаете все окно невидимым, нажмите F11 или выберите опцию *View* | *Object Inspector*, чтобы оно вновь появилось на экране.

6.1.4. Окно кода программы

Окно кода предназначено для создания и редактирования текста программы. Этот текст составляется по специальным правилам и описывает алгоритм работы программы. Совокупность правил записи текста называется языком программирования. В системе *Delphi* используется язык программирования *Object Pascal*, который представляет собой расширенную и усовершенствованную версию широко распространенного языка Паскаль, впервые предложенного швейцарским ученым Н. Виртом еще в 1970 г. и усовершенствованного сотрудниками корпорации *Borland* (созданные ими языки назывались *Turbo Pascal, Borland Pascal* и *Object Pascal*). Несмотря на то, что визуальная среда *Delphi* берет на себя многие рутинные аспекты программирования, знание языка *Object Pascal* является непременным условием для любого программиста, работающего в этой среде.

Первоначально окно кода содержит минимальный исходный текст, обеспечивающий нормальное функционирование пустой формы в качестве полноценного *Windows*-окна. В ходе работы над проектом программист вносит в него необходимые дополнения, чтобы придать программе нужную функциональность. Поскольку для создания даже

простых программ вам понадобится создавать и изменять (редактировать) код программы, ниже описываются основные приемы работы с окном кода.

Сразу после открытия нового проекта в нем будут такие строки:

```
UNIT Unit1;
INTERFACE
USES
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs;
TYPE
TForm1 = CLASS (TForm)
PRIVATE
{ Private declarations}
PUBLIC
{ Public declarations}
END;
VAR
Form1: TForm1;
IMPLEMENTATION
{$R*. DFM)
END.
```

Эти строки *Delphi* автоматически вставляет в окно кода для новой формы. Как уже говорилось, окно кода определяет поведенческую сторону окна программы (т. е. окна, появляющегося после начала работы программы), а окно формы – его внешние проявления. Оба окна тесно связаны друг с другом, причем Delphi вставляет необходимые строки в верхней части окна между

UNIT Unit1;

и

IMPLEMENTATION

Пока не стоит изменять эту часть текста. В дальнейшем мы будем вставлять в окно текст программы между строками

{$R*. DFM}

и

END.

в нижней части окна.

Чтобы вставить в окно новую строку (строки), нужно сначала с помощью клавиш курсора или щелкнув по окну мышью поставить текстовый указатель (мигающую вертикальную черту) на нужное место, а затем с помощью клавиатуры ввести текст. Обычно текст кода программы располагается в нескольких строках. Для перехода на новую строку используйте клавишу *Enter*.

Если в процессе ввода вы ошиблись и тут же заметили свою ошибку, удалите ошибочный символ клавишей *Backspace*. Клавиша *Backspace* удаляет символ слева от мигающего указателя, а клавиша *Delete* – справа от него. Если понадобится удалить сразу всю строку текста, поставьте в любое место строки мигающий указатель, нажмите клавишу *Ctrl* и, не отпуская ее, клавишу с латинской буквой Y. Такое совместное нажатие клавиш в дальнейшем будем обозначать символом «+»: *Ctrl* + Y. Чтобы отменить последнее изменение текста, нажмите *Ctrl* + Z или выберите пункт меню *Edit* | *Undo*.

Вместе с окном кода обычно активизируется также и окно браузера *Code Explorer*, облегчающее поиск нужных элементов в случае, когда в окне набрано много строк кода.

6.2. Основы визуального программирования в среде Delphi

Программирование в *Delphi* строится на тесном взаимодействии двух процессов: процесса конструирования визуального проявления программы (т. е. ее *Windows-окна)* и процесса написания кода, придающего элементам этого окна и программе, в целом, необходимую функциональность. Для написания кода используется окно кода, для конструирования программы – остальные окна *Delphi*, и прежде всего – окно формы.

Между содержимым окон формы и кода существует неразрывная связь, которая строго отслеживается *Delphi.* Это означает, что размещение на форме компонента приводит к автоматическому изменению кода программы и наоборот – удаление тех или иных автоматически вставленных фрагментов кода может привести к удалению соответствующих компонентов. Помня об этом, программисты вначале конструируют форму, размещая на ней очередной компонент, а уже только после этого переходят, если это необходимо, к написанию фрагмента кода, обеспечивающего требуемое поведение компонента в работающей программе.

6.2.1. Пустая форма и ее модификация

Окно формы содержит проект *Windows-окна* программы. Важно помнить, что с самого начала работы над новой программой *Delphi* создает минимально необходимый код, обеспечивающий ее нормальное функционирование в *Windows.* Таким образом, простейшая программа готова сразу после выбора опции *File* | *New Application*, и нам остается просто запустить программу.

6.2.1.1. Настройка Delphi

В процессе работы над проектами программ необходимо создать множество форм и модулей. Полезно сохранять эти данные в виде дисковых файлов в отдельной папке. Более того, для каждой программы в этой папке имеет смысл создать свою вложенную папку. Тогда, чтобы освободить место на диске для серьезной программы, вам будет достаточно уничтожить основную папку, а чтобы передать ту или иную учебную программу своему коллеге – переписать на дискету содержимое соответствующей вложенной папки. Создайте папку с именем, например, *MY_DELPH*, а в нем – вложенную папку для вашей первой программы.

Второе, что нам предстоит сделать, – это внести изменения в стандартную настройку среды *Delphi.* Это необходимо для того, чтобы среда автоматически сохраняла на диске последнюю версию создаваемой вами программы. Выберите опцию меню *Tools* | *Environment Options* и убедитесь, что в появившемся диалоговом окне активна страница *Preferences*. В левом верхнем углу этой страницы в группе *Autosave Options* есть переключатели *Editor Files* и *Desktop* (в других версиях *Delphi* эти переключатели располагаются в правом верхнем углу). Активизация переключателей приведет к автоматическому сохранению текста окна кода программы и общего расположения окон *Delphi* перед началом очередного прогона создаваемой программы, что избавит вас от возможных потерь в случае «зависания» программы.

Теперь все готово для прогона вашей первой программы. Щелкните мышью по кнопке  в Главном окне или, что проще, нажмите клавишу *F9*: именно таким способом подготовленная *Delphi*-программа последовательно проходит три главных этапа своего жизненного цикла – этапы компиляции, компоновки и исполнения. На этапе компиляции осуществляется преобразование подготовленного в окне кода текста программы на языке *Object Pascal* в последовательность машинных инструкций, на этапе компоновки к ней подключаются необходимые вспомогательные подпрограммы, а на этапе исполнения готовая программа загружается в оперативную память и ей передается исполнение.

Как только вы нажмете *F9*, появится диалоговое окно *Save Unit! As*, в котором *Delphi* попросит вас указать имя файла для модуля *Unit1.pas* и папку его размещения. По умолчанию *Delphi* предлагает разместить файл модуля и проекта в системной папке BIN. Поскольку эта папка содержит жизненно важные для *Delphi* файлы, обязательно измените ее на вашу рабочую папку (например, MY_DELPH).

6.2.1.2. Имена в Delphi

Delphi принципиально не признает никаких имен, в которых используются символы, отличные от латинских букв, цифр и знака подчеркивания. Причем имя не должно начинаться цифрой, но может начинаться знаком подчеркивания. Так как в этом перечне нет пробела, имена не могут также состоять из нескольких слов. Если вы работаете с *Delphi 32* (т. е. с любой из версий от 2 до 5), можете не сокращать имена, потому что эти версии рассчитаны на современные 32-разрядные операционные системы, разрешающие использовать длинные имена файлов. Если вы работаете с *Delphi* версии 1, то необходимы сокращения до 8 символов.

6.2.1.3. Изменение свойств формы

Модуль создается каждый раз, когда вы создаете новую форму (в программе может быть и чаще бывает не одна, а несколько – иногда несколько десятков – форм и связанных с ними модулей). При компиляции программы *Delphi* создает файлы с расширениями *PAS, DFM* и *DCU* для каждого модуля: *PAS*-файл содержит копию текста из окна кода программы, в файле с расширением *DFM* хранится описание содержимого окна формы, а в *DCU*-файле – результат преобразования в машинные инструкции текста из обоих файлов. Файлы *DCU* создаются компилятором и дают необходимую базу для работы компоновщика, который преобразует их в единый загружаемый файл с расширением *EXE*.

Попробуем модифицировать программу, например, изменим заголовок ее окна. По умолчанию заголовок окна совпадает с заголовком формы: *Form1*. Чтобы изменить заголовок, нужно обратиться к окну Инспектора Объектов. Закройте окно работающей программы *Form1* и щелкните мышью по строке *Caption* (Заголовок) окна Инспектора Объектов. Теперь правая колонка этой строки с текстом *Form1* выделена цветом, и справа от выделенного виден текстовый мигающий курсор. Переключите клавиатуру в режим ввода кириллицы и введите Моя первая программа, после чего еще раз нажмите *F9*. Новый прогон программы создаст окно с заголовком *Моя первая программа,* что несет в себе гораздо больше информации, чем просто *Form1*.

Итак, простым изменением содержимого строки в окне Инспектора Объектов мы добились важной перемены: изменили одно из свойств окна программы – его заголовок. Таким же образом можно изменять любое другое свойство формы.

6.2.1.4. Размещение нового компонента

Для размещения нового компонента на форме сначала нужно его выбрать (щелкнуть по нему мышью) в палитре компонентов, а затем щелкнуть мышью по точке рабочего пространства формы, где должен располагаться левый верхний угол компонента.

Попробуем таким способом включить в окно программы компонент *Label* (Метка), предназначенный для размещения различного рода надписей. Убедитесь в том, что в палитре компонентов выбрана страница *Standard,* и щелкните мышью по кнопке  (эта кнопка отображает компонент *Label* в палитре компонентов). Теперь щелкните мышью по форме так, чтобы компонент появился на форме и расположился левее и выше ее центра (рис. 6.2).

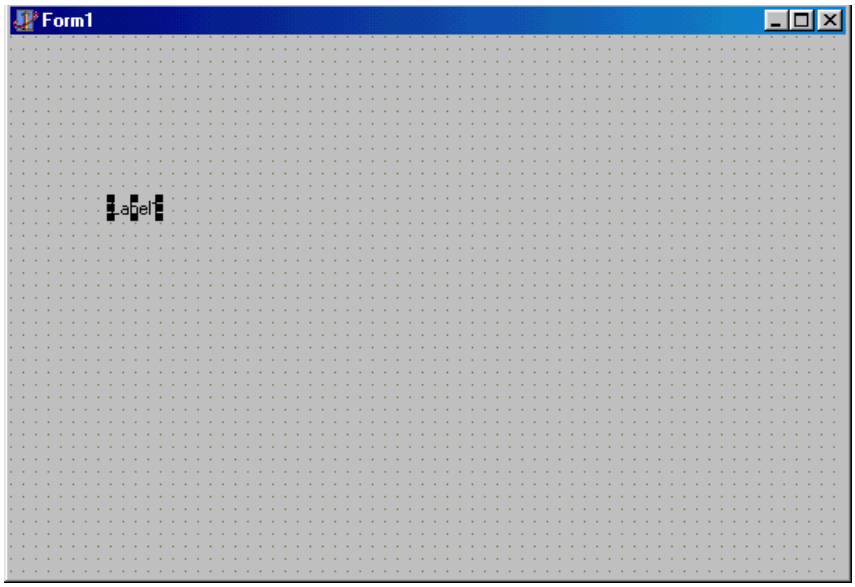


Рис. 6.2. Размещение компонента Label

Первоначальные размеры и положение компонента на форме легко изменяются мышью, поэтому добиваться полного сходства с рисунком необязательно.

Новый компонент имеет стандартное имя *Label1*, и надпись на нем повторяет это имя. Изменим эту надпись: с помощью строки *Caption* окна Инспектора Объектов введите надпись

Я программирую на Delphi

Как только вы начнете вводить новую надпись, вид компонента на форме начнет меняться, динамически отражая все изменения, производимые вами в окне Инспектора Объектов. Выделим надпись цветом и сделаем ее шрифт более крупным. Для этого щелкните мышью по свойству *Font* окна Инспектора Объектов и с помощью кнопки в правой час-

ти строки раскройте диалоговое окно Настройки шрифта. В списке *Size* (Размер) этого окна выберите высоту шрифта 24 пункта, а с помощью списка *Color* (Цвет) выберите нужный цвет (например, красный), после чего закройте окно кнопкой *OK*.

Надпись на компоненте в окне формы тут же соответствующим образом изменит свои свойства. *Delphi* обладает способностью визуальной реализации любых изменений свойств компонента не только на этапе прогона программы, но и на этапе проектирования формы.

Щелкните мышью внутри обрамляющих надпись черных прямоугольников и, не отпуская левую кнопку мыши, сместите ее указатель так, чтобы он расположился левее в центре окна, после чего отпустите кнопку. Таким способом можно буксировать компонент по форме, добиваясь нужного его положения.

С помощью обрамляющих черных квадратиков можно изменять размеры компонента. Для этого следует поместить острие указателя мыши над одним из них (в этот момент указатель меняет свою форму на двунаправленную стрелку), затем нажать левую кнопку мыши и, не отпуская ее, буксировать сторону или угол компонента в нужном направлении, после чего отпустить кнопку.

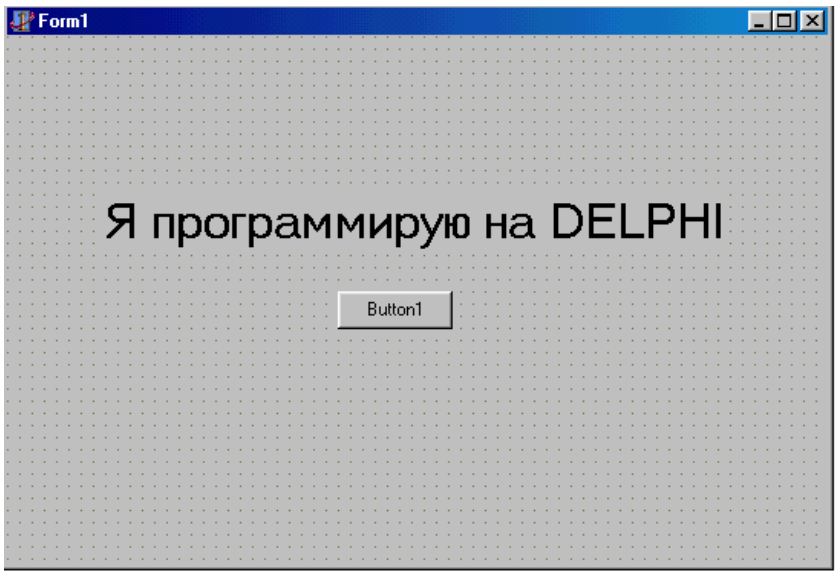


Рис. 6.3. Размещение компонента Button

Все видимые компоненты имеют свойства *Left* (Слева), *Top* (Сверху), *Width* (Ширина) и *Height* (Высота), числовые значения которых определяют положение левого верхнего угла компонента и его размеры в так называемых *пикселях*, т. е. в минимальных по размеру точках экрана, светимостью которых может управлять программа. При буксировании компонента или изменении его размеров мышью эти значения ав-

томатически меняются, и, наоборот, – изменение этих свойств в окне Инспектора Объектов приводит к соответствующему изменению положения и размеров компонента. В *Delphi* 4 и 5 значения *Left* и *Top* автоматически появляются в небольшом окне рядом с указателем мыши при буксировке компонента по форме.

Компонент *кнопка* изображается пиктограммой  на странице *Standard* палитры компонентов. Поместите этот компонент на форму и расположите его ниже метки и посередине формы (рис. 6.3).

6.2.2. Реакция на события

6.2.2.1. Обработчик события OnClick

При щелчке по кнопке мышью в работающей программе возникает событие *OnClick* (По щелчку). Пока это событие никак не обрабатывается программой, и поэтому «нажатие» кнопки не приведет ни к каким последствиям. Чтобы заставить программу реагировать на нажатие кнопки, необходимо написать на языке *Object Pascal* фрагмент программы, который называется *обработчиком события.*

Этот фрагмент должен представлять собой последовательность текстовых строк, в которых программист указывает, что именно должна делать программа в ответ на нажатие кнопки. Фрагмент оформляется в виде специальной подпрограммы языка *Object Pascal – процедуры.*

Чтобы заставить *Delphi* самостоятельно сделать заготовку для процедуры обработчика события *OnClick,* дважды подряд без заметной паузы щелкните мышью по вновь вставленному компоненту. В ответ *Delphi* активизирует окно кода, и вы увидите в нем такой текстовый фрагмент:

```
PROCEDURE TForm1.Button1Click(Sender: TObject);
BEGIN
END;
```

Слово **PROCEDURE** извещает компилятор о начале подпрограммы-процедуры (в *Delphi* могут использоваться также подпрограммы-функции; в этом случае вместо **PROCEDURE** (процедура) используется слово **FUNCTION** (функция). За ним следует имя процедуры *TForm1.ButtonlClick*. Это имя – составное: оно состоит из имени класса *TForm1* и собственно имени процедуры *ButtonlClick.*

Классами в *Delphi* называются функционально законченные фрагменты программ, служащие образцами для создания подобных себе экземпляров.

Однажды создав класс, программист может включать его экземпляры (копии) в разные программы или в разные места одной и той же программы. Такой подход способствует максимально высокой продук-

тивности программирования за счет использования ранее написанных фрагментов программ. В состав *Delphi* входит несколько сотен классов, созданных программистами корпорации *Inprise* (так называемых стандартных классов). Совокупность стандартных классов определяет мощные возможности этой системы программирования.

Каждый компонент принадлежит к строго определенному классу, а все конкретные экземпляры компонентов, вставляемые в форму, получают имя класса с добавленным числовым индексом. По используемому в *Delphi* соглашению все имена классов начинаются с буквы *T.* Таким образом, имя *TFormI* означает имя класса, созданного по образцу стандартного класса *TForm.* Если вы посмотрите начало текста в окне кода, то увидите следующие строки:

```
TYPE
TForm1 = CLASS(TForm)
Button1: TButton;
Label1: TLabel;
PROCEDURE ButtonlClick(Sender: TObject);
PRIVATE
{ Private declarations }
PUBLIC
{ Public declarations }
END;
VAR
Form1: TForm1;
```

Строка

```
TForm1 = CLASS(TForm)
```

определяет новый класс *TForm1,* который **порожден от** (создан по образцу) стандартного класса *TForm.* Строка

```
Form1: TForm1;
```

создает **экземпляр** этого класса с именем *Form1*. Стандартный класс *TForm* описывает пустое *Windows-окно,* в то время как класс *TFormI* описывает окно с уже вставленными в него компонентами *метка* и *кнопка.* Описание этих компонентов содержат строки

```
Button1: TButton;
Label1: TLabel;
```

Они указывают, что компонент Button1 (Кнопка1) представляет собой экземпляр стандартного класса *TButton,* а компонент *Label1* (Метка1) – экземпляр класса *TLabel.*

За именем процедуры TForm1 ButtonlClick в круглых скобках следует описание параметра вызова

```
Sender: TObject
```

(параметр с именем *Sender* принадлежит классу *TObject)*. Как мы увидим дальше, процедуры могут иметь не один, а несколько параметров вызова или не иметь их вовсе. Параметры вызова (если они есть) служат настройке реализованного в процедуре алгоритма на выполнение конкретной работы. Параметр *Sender* вставлен *Delphi* «на всякий случай»: с его помощью подпрограмма *ButtonlClick* может, при желании, определить, какой именно компонент создал событие *OnClick.*

Вся строка в целом

PROCEDURE TForm1.ButtonlClick(Sender: TObject);

называется **заголовком процедуры.** Ее завершает символ «;». Этот символ играет важную роль в *Object Pascal,* т. к. показывает компилятору на конец **предложения** языка. Из отдельных предложений составляется весь текст программы. В конце каждого предложения нужно ставить точку с запятой – это обязательное требование синтаксиса языка. Три следующие строки определяют **тело процедуры:**

BEGIN
END;

Слово **BEGIN** (начало) сигнализирует компилятору о начале последовательности предложений, описывающих алгоритм работы процедуры, а слово **END** (конец) – о конце этой последовательности. Наполнить тело нужными предложениями – задача программиста. Каждый раз при нажатии кнопки Button1 управление будет передаваться в тело процедуры, а значит, между словами **BEGIN** и **END** мы можем написать фрагмент программы, который будет выполняться в ответ на это событие.

6.2.2.2. Динамическое изменение свойств компонента

Поскольку кнопка Button1 в нашей программе способна «звучать», полезно изменить ее надпись: вместо умалчиваемой надписи *Button1,* которую автоматически формирует *Delphi* по имени компонента, назовем кнопку, например, «Звук». Проще всего это сделать с помощью окна формы и Инспектора Объектов, т. е. на этапе конструирования формы (для этого нужно просто изменить свойство *Caption* компонента *Button1* в окне Инспектора Объектов), но для более полного знакомства с *Delphi* мы рассмотрим другой способ – динамического изменения надписи на этапе прогона программы. Изменения на этапе конструирования называются *статическими*, а в ходе прогона программы – *динамическими*. Для этого создадим обработчик события *On Create* (По созданию) для формы и изменим в нем это свойство.

Событие *OnCreate* возникает после создания *Windows-окна*, но до появления этого окна на экране. Чтобы создать обработчик этого события, раскройте список компонентов в верхней части окна инспектора

объектов, выберите компонент *Form1* и дважды щелкните по свойству *OnCreate* на странице *Events* этого компонента (щелкать нужно по правой части строки *OnCreate)*. В ответ *Delphi* вновь активизирует окно кода и покажет вам заготовку для процедуры TForm1. FormCreate. Отредактируйте ее следующим образом:

```
PROCEDURE TForm1.FormCreate(Sender: TObject);
BEGIN
Button1.Caption := 'Звук';
END;
```

Единственная вставленная нами строка представляет собой так называемый **оператор присваивания** языка *Object Pascal.* В левой части оператора указывается свойство Button1.Caption, а в правой части – значение 'Звук', которое мы хотим придать этому свойству. Связывает обе части комбинация символов «:=», которая читается как «присвоить значение». Символы «:=» всегда пишутся слитно, без разделяющих пробелов, хотя перед двоеточием и после знака равенства можно для лучшей читаемости программы вставлять пробелы, что мы и сделали. Как и любое другое предложение языка, оператор присваивания завершается точкой с запятой.

Составное имя Button1.Caption необходимо для точного указания компилятору, о каком свойстве идет речь: в нашей программе используются три компонента (включая саму форму), каждый из которых имеет свойство Caption; уточняющий префикс Button1 заставит изменить это свойство у кнопки, а не у метки или формы. Присваиваемое свойству значение является текстовой строкой. По правилам *Object Pascal* текстовая строка должна заключаться в обрамляющие апострофы. Внутри апострофов можно написать любое количество произвольных символов – именно они (без обрамляющих апострофов) будут определять новую надпись на кнопке.

После очередного прогона программы вы увидите измененную надпись на кнопке, а мы сделаем важный вывод: любое свойство любого компонента можно изменять *динамически,* т. е. в ходе исполнения программы.

6.3. Использование компонентов общего назначения

Компоненты представляют собой элементы, из которых конструируется видимое изображение, создаваемое работающей программой. Следует заметить, что существует значительное количество компонентов, которые не создают видимого изображения, но которые, тем не менее, играют важную роль в тех или иных случаях. Правильнее думать о компонентах как о заранее приготовленных для вас фрагментах про-

граммы, которые можно вставлять, если в этом есть необходимость, в разрабатываемую программу.

Библиотеки компонентов для разных версий *Delphi* строятся по принципу расширения: в первой версии было около 70 компонентов, в то время как в состав *Delphi 5* входит более 200 компонентов.

Страница STANDARD

На странице *Standard* палитры компонентов сосредоточены стандартные для *Windows* интерфейсные элементы, перечисленные в следующей таблице:

Пиктограмма	Имя	Назначение
	*Frame*	Рама. Наравне с формой служит контейнером для размещения других компонентов. В отличие от формы может размещаться в палитре компонентов, создавая заготовки компонентов
	*ManeMenu*	Главное меню программы. Компонент способен создавать и обслуживать сложные иерархические меню
	*PopupMenu*	Вспомогательное или локальное меню. Обычно это меню появляется в отдельном окне после нажатия правой кнопки мыши
	*Label*	Метка. Этот компонент используется для размещения в окне не очень длинных однострочных надписей
	*Edit*	Строка ввода. Предназначена для ввода, отображения или редактирования одной текстовой строки
	*Memo*	Многострочный текстовый редактор. Используется для ввода и/или отображения многострочного текста
	*Button*	Командная кнопка. Обработчик события *OnClick* этого компонента обычно используется для реализации некоторой команды
	*CheckBox*	Независимый переключатель. Щелчок мышью на этом компоненте в работающей программе изменяет его логическое свойство *Checked*
	*RadioButton*	Зависимый переключатель. Обычно объединяется еще с одним таким же компонентом в группу. Щелчок приводит к автоматическому освобождению ранее выбранного переключателя в той же группе

Пиктограмма	Имя	Назначение
	*ListBox*	Список выбора. Содержит список предлагаемых вариантов (опций) и дает возможность проконтролировать текущий выбор
	*ComboBox*	Комбинированный список выбора. Представляет собой комбинацию списка выбора и текстового редактора
	*ScrollBar*	Полоса управления. Представляет собой вертикальную или горизонтальную полосу, напоминающую полосы прокрутки по бокам *Windows*-окна
	*GroupBox*	Группа элементов. Этот компонент используется для группировки нескольких связанных по смыслу компонентов
	*RadioGroup*	Группа зависимых переключателей. Содержит специальные свойства для обслуживания нескольких связанных зависимых переключателей
	*Panel*	Панель. Этот компонент, как и *GroupBox,* служит для объединения нескольких компонентов. Содержит внутреннюю и внешнюю кромки, что позволяет создать эффекты «вдавленности» и «выпуклости»
	*ActionList*	Список действий. Служит для централизованной реакции программы на действия пользователя, связанные с выбором одного из группы однотипных управляющих элементов, таких как опции меню, пиктографические кнопки и т. п.

6.3.1. TFrame – рама и шаблоны компонентов

Этот компонент впервые введен в *Delphi 5*. Он определяет раму – контейнер для размещения других компонентов. В функциональном отношении компонент почти повторяет свойства формы и отличается от нее, в основном, лишь тем, что его можно помещать на формы или в другие рамы. Фактически рама представляет собой удобное средство создания *шаблонов* – произвольных наборов компонентов, максимально приспособленных для нужд конкретного пользователя. Раз созданный шаблон может подобно любому другому компоненту размещаться на форме или другой раме (допускается неограниченная вложенность рам). Любые изменения в базовой раме (т. е. в раме, сохраненной в палитре) тут же отображаются во всех проектах, использующих данную раму.

Первоначально проекту ничего неизвестно о, возможно, ранее созданных рамах, поэтому попытка поместить на пустую форму компонент-раму вызовет сообщение:

No frames in project. To create a frame select File | New Frame.

(В проекте нет рам. Чтобы создать раму выберите File | New Frame.)

Это сообщение и описываемая ниже методика подключения шаблонов – единственное, что отличает механизм использования шаблонов от использования стандартных компонентов.

Создадим простую раму, содержащую две кнопки – *mbOk* и *mbCancel.* Такой шаблон может пригодиться при конструировании различных диалоговых окон.

1. Создадим новый проект (*File*| *New Application*).

2. Создайте новую раму – выберите *File* | *New Frame.*

3. Поместите на раму две кнопки *TBitBtn* и установите следующие свойства для кнопок и рамы:

Установленные параметры создадут минимальную по размерам раму с двумя именованными кнопками. Так как свойство *Anchors* рамы содержит значения [akRight, akBottoms], рама будет все время отслеживать свое положение относительно правого нижнего угла контейнера, в который она будет помещена.

4. Щелкните по раме правой кнопкой мыши и выберите в локальном меню *Add To Palette* – на экране появится диалоговое окно с предложением сохранить модуль шаблона в дисковом файле.

5. Сохраните шаблон в файле с именем *DlgBtnsF* в папке для ваших программ. На экране появится окно регистрации шаблона в палитре компонентов. В окне предлагается установить для нового компонента имя класса и страницу палитры компонентов, в которую он помещен. Обратите внимание на автоматически созданное умалчиваемое имя заготовки – *TFrame1Tamplate.* Так как это имя будет появляться в ярлычке *Hint* при высвечивании компонента мышью, следует дать ему более осмысленное имя, например, *TDialogButtons.* И еще одно замечание: в качестве страницы палитры компонентов в окне предлагается страница Templates (шаблоны), которая отсутствует в начальном наборе страниц палитры. Если вы сохраните это имя, в *Delphi* будет создана новая страница компонентов.

6. Задайте имя класса шаблона и страницу его размещения, после чего щелкните по кнопке *OK* – шаблон готов для использования.

7. После закрытия окна регистрации сам шаблон останется на экране. Подобно дополнительной форме его модуль стал составной частью проекта, поэтому размещение шаблона на форме не вызовет проблем.

8. Щелкните по пиктограмме  на странице *Standard* палитры компонентов и затем по пустому месту в форме *Form1* – на экране появится окно с предложением выбрать нужный шаблон. Нажмите *Enter*, и шаблон появится в форме так, как если бы это был любой другой компонент.

Следует отметить, что, если вы откроете новый проект и попытаетесь разместить на форме раму со страницы *Standard*, на экране вновь появится сообщение о том, что в проекте нет рам. Однако если вы присоедините модуль *DlgBtnsF* к проекту клавишами *Shift-F11*. Если в новом проекте попробовать установить на пустую форму компонент-шаблон со страницы *Templates* (или с любой другой страницы, куда он был ранее помещен на этапе регистрации), на экране появится окно с такой надписью:

The following unit: DIgBtnsF, is needed in your project to create the template. Do you wish to add it?
(Модуль DIgBtnsF необходим в вашем проекте, чтобы создать шаблон. Хотите его добавить?)

Свойства входящих в шаблон компонентов, а также свойства самого размещенного на форме шаблона можно менять, приспосабливая их к конкретным нуждам программы.

Компонент *TFrame* является потомком *TScrollingWinControl,* от которого ему достались три описываемых ниже специфичных свойства (остальные свойства, события и методы унаследованы от *TWinControl*):

PROPERTY AutoScroll: Boolean;	Разрешает/запрещает автоматическую вставку полос прокрутки, если не все размещенные на раме компоненты умещаются в отведенных ей размерах
PROPERTY HorzScrollBar: TControlScrollBar;	Определяет свойства горизонтальной полосы прокрутки
PROPERTY VertScrollBar: TControlScrollBar;	Определяет свойства вертикальной полосы прокрутки

Класс *TControlScrollBar*, к которому принадлежат свойства *HorzScrollBar* и *VertScrollBar*, устанавливает все необходимые свойства и методы для полос прокрутки.

Свойства *TControlScrollBar*:

PROPERTY ButtonSize: Integer;	Определяет размер кнопок полосы прокрутки
PROPERTY Color: TColor;	Определяет цвет полосы.

TYPE TScrollBarInc = 1..32767; **PROPERTY** Increment: TScrollBarInc;	Указывает перемещение бегунка при щелчке мышью на концевой кнопке полосы
TYPE=(sbHorizontal, sbVertical); **PROPERTY** Kind: ScrollBarKind;	Указывает ориентацию полосы (это свойство – только для чтения)
PROPERTY Margin: Word;	Определяет минимальное расстояние от полосы до края компонента, в котором она расположена
PROPERTY ParentColor: Boolean;	Если *True,* цвет полосы определяется системными установками *Windows.* Установка значения в свойство *Color* приводит к автоматической установке значения *False* в свойство *ParentColor*
PROPERTY Position: Integer;	Определяет положение бегунка на полосе прокрутки
PROPERTY Range: Integer;	Определяет размер скроллируемой области
PROPERTY ScrollPos: Integer;	Определяет положение скроллируемой области
PROPERTY Size: Integer;	Определяет ширину полосы в пикселях
PROPERTY Smooth: Boolean;	Если содержит *True,* прокрутка осуществляется стандартным образом: щелчок по концевой кнопке вызывает смещение приблизительно на 1/10 часть всей прокручиваемой области. Если содержит *False,* смещение определяется свойством *Increment*
TScrollBarStyle = (ssRegular, ssFlat, ssHotTrack); **PROPERTY** Style: TScrollBarStyle;	Определяет стиль полосы: *ssRegular* – обычная полоса; *ssFlat* – плоская полоса; *ssHotTrack* – плоская полоса, но ее компоненты выделяются цветом при перемещении над ней указателя мыши
PROPERTY Thumbsize: Integer;	Определяет ширину бегунка в пикселях
PROPERTY Tracking: Boolean;	Разрешает/запрещает динамическую прокрутку при перемещении ползунка
PROPERTY Visible: Boolean;	Определяет видимость полосы

6.3.2. TMainMenu – главное меню формы (программы)

Компонент класса *TMainMenu* определяет главное меню формы. На форму можно поместить сколько угодно объектов этого класса, но отображаться в полосе меню в верхней части формы будет только тот из них, который указан в свойстве *Menu* формы.

После установки компонента на форму необходимо создать его опции. Для этого следует дважды щелкнуть по компоненту левой кнопкой мыши либо нажать на нем правую кнопку и выбрать продолжение *Menu Designer* в появившемся вспомогательном меню, либо, наконец, щелкнуть по кнопке в правой половине строки *Items* Инспектора Объектов.

Создание опций не вызывает проблем. Перейдите в окно Инспектора Объектов и введите текст опции в строке *Caption,* после чего нажмите *Enter* – опция готова и можно переходить к следующей. Каждая опция главного меню может раскрываться в список подопций или содержать конечную команду. Для создания подопций щелкните мышью по строке ниже опции и введите первую подопцию. Продолжайте ввод, пока не будет создан весь список подопций, после чего щелкните по пустому прямоугольнику справа от первой опции и введите вторую опцию. Процесс гораздо сложнее описать, чем выполнить.

В названиях опций можно указать символ «&» перед тем символом, который определит клавишу быстрого выбора опции (в терминологии *Windows* такие клавиши называются акселераторами). Например, опцию *Файл* можно выбрать сочетанием клавиш *Alt+Ф.* При создании меню эта опция в строке *Caption* Инспектора Объектов содержала текст &Файл.

Если вы захотите вставить разделительную черту, отделяющую группы подопций, назовите очередной элемент меню именем «-».

Для создания разветвленных меню, т. е. таких, у которых подопции вызывают новые списки подопций, щелкните по подопции и нажмите *Ctrl+Вправо,* где *Вправо* – клавиша смешения курсора вправо. Такого эффекта можно добиться после щелчка правой кнопкой мыши на подопции и выборе продолжения *Create Submenu.*

Каждый элемент меню является объектом класса *TMenuItem.* Свойства этого класса описаны в следующей таблице. Термин «родительская опция» означает опцию, выбор которой приводит к раскрытию подменю с данной опцией.

PROPERTY Bitmap: TBitmap;	Содержит ссылку на связанное с опцией изображение. Это изображение (если оно есть) появляется слева от опции. Свойство игнорируется, если установлено свойство *ImageIndex* (см. ниже)
PROPERTY Break: TMenuBreak;	Позволяет создать многоколончатый список подменю
PROPERTY Checked: Boolean;	Если *True,* рядом с опцией появляется галочка
PROPERTY Command: Word;	Используется при разработке приложений, обращающихся непосредственно к API-функциям *Windows*
PROPERTY Count: Integer;	Содержит количество опций в подчиненном меню
PROPERTY Default: Boolean;	Определяет, является ли данная опция подменю умалчиваемой?
PROPERTY GroupIndex: Byte;	Определяет групповой индекс для зависимых опций

PROPERTY ImageIndex: Integer;	Содержит индекс связанного с опцией изображения из компонента *TimageList*. Это изображение (если оно есть) появляется слева от опции. Если свойство имеет значение -1, с опцией не связано никакого изображения из *TimageList*
PROPERTY Items[Index: integer]: TMenuItem;	Позволяет обратиться к любой опции подчиненного меню по ее индексу
PROPERTY MenuIndex: Integers-	Определяет индекс опции в списке *Items* родительской опции
PROPERTY RadioItem: Boolean;	Определяет, зависит ли данная опция от выбора других опций в той же группе *Grouplndex.* Только одна опция группы может иметь *True* в свойстве *Checked.* Рядом с такой опцией вместо галочки изображается круг
PROPERTY Shortcut: TShortCut	Задает клавиши-акселераторы для быстрого выбора данной опции

В *Delphi 4* появилась возможность связывать с опциями меню небольшие изображения. Эти изображения можно задать либо свойством *BitMap,* либо свойством *Imagelndex* (этих свойств компонентов *TMenu* нет в предыдущих версиях *Delphi).* Изображение (если оно есть) показывается слева от опции. Опции *New, NewForm, Open* и т. д. снабжены небольшими пиктограммами. Введение пиктограмм в меню, с одной стороны, повышает наглядность меню, а с другой – способствует унификации пиктограмм в рамках механизма *действий*. Если опция меню связана с каким-то действием своим свойством *Action,* а компонент *TActionList,* в котором это действие описано, в свою очередь, связан с хранилищем пиктограмм *TImageList*, индекс нужной пиктограммы можно задать в свойстве *Imagelndex.* В этом случае пиктограмма, указанная в свойстве *BitMap* (если она указана в нем) игнорируется.

Тип *TMenuBreak* определен следующим образом:

TYPE TMenuBreak = (mbNone, mbBarBreak, mbBreak);

Свойство *Break* по умолчанию имеет значение mbNone. Два других возможных значения этого свойства используются для создания многоколончатых списков подменю. Значение mbBarBreak заканчивает предыдущую колонку и начинает новую. Эта опция меню будет открывать новую колонку, которая отделяется от предыдущей вертикальной чертой. Значение mbBreak также создает новую колонку, но не вставляет разделительную черту. Эти значения игнорируются в опциях самого первого уровня. В окне конструктора многоколончатое меню отображается как обычное.

Если в опции *Default* установлено значение True, такая опция выделяется цветом и выбирается двойным щелчком мыши на родительской опции. Только одна опция в подменю может быть умалчиваемой.

В отличие от других видимых компонентов строка *Hint* для опций меню задает только расширенное сообщение, которое отображается на панели статуса.

Для элемента меню определено единственное событие *OnClick,* которое возникает при щелчке на опции или при нажатии *Enter,* если в этот момент, опция была выбрана (подсвечена). Обработчик события становится до двойного щелчка на опции в окне конструктора меню.

6.3.3. TPopupMenu – вспомогательное (локальное) меню

Компоненты класса *TPopupMenu* используются для создания вспомогательных (локальных) меню, появляющихся после нажатия правой кнопки мыши. В отличие от главного меню вспомогательное меню может быть создано для любого оконного компонента. Чтобы связать щелчок правой кнопкой мыши на компоненте с раскрытием вспомогательного меню, в свойство *PopupMenu* компонента необходимо поместить имя компонента-меню.

Вспомогательное меню создается с помощью конструктора меню, содержит элементы описанного класса *TMenuItem,* поэтому процесс создания и свойства вспомогательного меню ничем не отличаются от *TMainMenu.*

6.3.4. TLabel – метка для отображения текста

Компоненты класса *TLabel* (метка) предназначены для размещения на форме различного рода текстовых надписей. Для этого служит центральное свойство компонента – *Caption.* С помощью свойства *Font* можно разнообразить вид надписи (оба эти свойства достались метке от предка *TgraphicControl)*. С компонентом может быть связан оконный управляющий элемент, который выбирается при нажатии *Alt+Буква,* где *Буква* – выделенная подчеркиванием буква в тексте метки. Такие символы в терминологии *Windows* называются акселераторами.

Свойства компонента (кроме унаследованных от *TGraphicControl*):

PROPERTY AutoSize: Boolean;	Указывает, будет ли метка изменять свои размеры в зависимости от помещенного в ее свойство *Caption* текста: *True* – будет
PROPERTY FocusControl: TWinControl;	Содержит имя оконного компонента, который связан с меткой акселератором
TTextLayout = (tITop, tICenter, tIBottom) ; **PROPERTY** Layout: TTextLayout;	Определяет выравнивание текста по вертикали относительно границ метки: *tITop* – текст располагается вверху; *tICenter* – текст центрируется по вертикали; *tIBottom* – текст располагается внизу

PROPERTY ShowAccelChar: Boolean;	Если содержит *True,* символ *&* в тексте метки предшествует символу-акселератору
PROPERTY Transparent: Boolean;	Определяет прозрачность фона метки. Если *False,* фон закрашивается собственным цветом *Color,* в противном случае используется фон родительского компонента
PROPERTY WordWrap: Boolean;	Разрешает/запрещает разрыв строки на границе слова. Для вывода многострочных надписей задайте *AuroSize=False, Word Wrap =True* и установите подходящие размеры метки

Метка *Label* может отображать длинную текстовую строку своего свойства *Caption* в виде нескольких строк: для этого установите в *AutoSize* значение False, задайте достаточно большие размеры метки и поместите в *WordWrap* значение True.

6.3.5. TEdit – ввод и отображение строки

Компонент класса *TEdit* представляет собой однострочный редактор текста. С его помощью можно вводить и/или отображать достаточно длинные текстовые строки.

Центральным свойством компонента является *Text,* которое представляет собой отображаемую компонентом строку. С помощью обработчика события *OnChange* программа может контролировать вводимый пользователем текст и при необходимости фильтровать его, игнорируя недопустимые символы. В следующем примере компонент фильтрует все символы, которые не соответствуют правильному представлению вещественного числа:

```
VAR
OldText: STRING;
PROCEDURE TForm1.EditlChange(Sender:TObject) ;
BEGIN
IF Edit1.Text<>' ' THEN
TRY
StrToFloat(Edit1.Text) ;
OldText := Edit1.Text
EXCEPT
// Ошибка преобразования: восстанавливаем прежний текст
Edit1.Text := OldText;
// и позиционируем текстовый указатель в конец текста:
```

```
Editl.SelStart := Length(Edit1.Text);
Editl.SelText := ' '
END
END;
```

При повторении примера подготовьте глобальную переменную *OldText* типа **STRING**, в которой запоминается последний правильно введенный текст. Лучше всего ее поместить в секцию **PRIVATE** класса формы – тогда ее можно не обнулять в момент начала работы.

Позиционировать текстовый указатель на любой символ строки можно с помощью свойств *SelStart* и *SelText:* в первое нужно поместить порядковый номер символа от начала текста, после которого должен стоять указатель ввода, во второе – пустую строку.

Свойства компонента:

PROPERTY AutoSelect: Boolean	Указывает, будет ли выделяться весь текст в момент получения компонентом фокуса ввода
PROPERTY AutoSize: Boolean;	Если содержит *True* и *BorderStyle=bsSingle,* высота компонента автоматически меняется при изменении свойства *Font. Size*
TBorderStyle = BsNone . . bs Single; **PROPERTY** BorderStyle: TBorderStyle;	Определяет стиль обрамления компонента: *bsNone* – нет обрамления; *bsSingle* – компонент обрамляется одной линией
PROPERTY CanUndo: Boolean;	Содержит *True,* если сделанные пользователем изменения в тексте *Text* можно убрать методом *Undo*
TEditCharCase = (ecNormal, EcUpperCase, ecLowerCase) ; **PROPERTY** CharCase: TEditCharCase	Определяет автоматическое преобразование высоты букв: *ecNormal* – нет преобразования; *ee UpperCase* – все буквы заглавные; ecZowerCase – все буквы строчные. Правильно работает с кириллицей
PROPERTY HideSelection: Boolean;	Если содержит *False,* выделение текста сохраняется при потере компонентом фокуса ввода
PROPERTY MaxLength:Integer;	Определяет максимальную длину текстовой строки. Если имеет значение 0, длина строки не ограничена
PROPERTY Modified: Boolean;	Содержит *True,* если текст был изменен
PROPERTY OnChange:TNotifyEvent;	Определяет обработчик события *OnChange,* которое возникает после любого изменения текста

PROPERTY OEMConvert: Boolean;	Содержит *True,* если необходимо перекодировать текст из кодировки *MS-DOS* в кодировку *Windows* и обратно
PROPERTY PasswordChar: Char;	Если символ *PasswordChar* определен, он заменяет собой любой символ текста при отображении в окне. Используется для ввода паролей
PROPERTY Readonly: Boolean;	Если содержит *True,* текст не может изменяться
PROPERTY SelLength: Integer;	Содержит длину выделенной части текста
PROPERTY SelStart:Integer;	Содержит номер первого символа выделенной части текста
PROPERTY SelText: String;	Содержит выделенную часть текста
PROPERTY Text: String;	Содержит весь текст

Методы компонента:

PROCEDURE Clear;	Удаляет весь текст
PROCEDURE ClearSelection;	Удаляет выделенный текст
PROCEDURE ClearUndo;	Очищает буфер метода Undo
PROCEDURE CopyToClipboard;	Копирует выделенный текст в Clipboard
PROCEDURE CutToClipboard;	Копирует выделенный текст в Clipboard, после чего удаляет выделенный текст из компонента
FUNCTION GetSelTextBuf(Buffer: PChar; BufSize: Integer): Integer	Копирует не более BufSize символов выделенного текста в буфер Buffer
PROCEDURE PasteFromClipboard;	Заменяет выделенный текст содержимым Clipboard, а если нет выделенного текста, копирует содержимое Clipboard в позицию текстового курсора
PROCEDURE SelectAll;	Выделяет весь текст
PROCEDURE SetSelText-Buf(Buffer: PChar);	Заменяет выделенный текст содержимым Buffer, а если нет выделенного текста, копирует содержимое Buffer в позицию текстового курсора
PROCEDURE Undo;	Восстанавливает текст в том виде, в каком он был перед последним получением компонентом фокуса ввода

6.3.6. TMemo – ввод и отображение текста

Компоненты класса *TMemo* предназначены для ввода, редактирования и/или отображения достаточно длинного текста. Текст хранится в свойстве *Lines* класса *TStrings* и, таким образом, представляет собой пронумерованный набор строк (нумерация начинается с нуля). С помощью свойств и методов этого класса (*Count, Add, Delete, Clear* и т. д.) можно динамически формировать содержимое компонента.

Свойства *BorderStyle, CanUndo, HideSelection, MaxLentgh, Modified, OEMConvert, OnChange, ReadOnly, SelLength, SelStart* и *SelText* аналогичны соответствующим свойствам класса *TEdit.* Свойство *Wordwrap* аналогично свойству *TLabel. WordWrap.* Другие специфичные свойства представлены ниже:

PROPERTY CaretPos: TPoint;	Содержит координаты мигающего текстового курсора относительно границ клиентской области компонента (только для *Delphi 4, 5)*
PROPERTY Lines: TStrings;	Содержит строки текста
TScrollStyle = (ssNone, ssHorizontal, ssVertical, ssBoth); **PROPERTY** ScrollBars: TScrollStyle;	Определяет наличие в окне редактора полос прокрутки: *ssNone* – нет полос; *ssHorizontal* – есть горизонтальная полоса; *ssVertical* – есть вертикальная полоса; *ssBoth* – есть обе полосы
PROPERTY WantReturns: Boolean;	Если содержит *True,* нажатие *Enter* вызывает переход на новую строку, в противном случае – обрабатывается системой. Для перехода на новую строку в этом случае следует нажать *Ctrl+Enter*
PROPERTY WantTabs: Boolean;	Если содержит *True,* нажатие *Tab* вызывает ввод в текст символа табуляции, в противном случае – обрабатывается системой. Для ввода символа табуляции в этом случае следует нажать *Ctrl+Tab*

Следует заметить, что, если свойство *ScrollBars* содержит *ssHorizontal* или *ssBoth,* свойство *WordWrap* игнорируется, и длинные строки будут отсекаться границами компонента без переноса текста на следующую строку. Специфичные методы класса аналогичны методам класса *TEdit.* Поскольку компонент является потомком *TControl,* он имеет также свойство *Text,* которое содержит отображаемый компонентом текст в виде одной длинной строки. В этой цепочке символов границы строк многострочного текста выделяются символами #13#10 (признак *EOLN* – конец строки). В отличие от этого свойство *Lines* содержит пронумерованный список строк: первая строка в этом списке имеет индекс О, вторая – 1, а общее количество строк можно узнать с помощью. Lines.Count. Свойство *Text* удобно использовать для поиска в тексте нужного фрагмента. Чтобы, например, найти и выделить в тексте фраг-

мент, содержащийся в компоненте *edSearch* типа *TEdit,* можно использовать такой обработчик события *OnClick* кнопки *btSearch:*

```
PROCEDURETForm1.btSearchClick(Sender: TObject);
VAR
k: Integer;
BEGIN
WITH Memo1 DO
BEGIN
k := pos(edSearch.Text,Text);
IF k>0 THEN
BEGIN
SelStart := k-1;
SelLength := Length(edSearch.Text)
END
END
END;
```

Если вы захотите, чтобы найденный в тексте фрагмент после установки *SelStart* и *SelLength* сразу бы выделился цветом, установите False в свойство *HideSelection.*

Для загрузки в компонент текста из файла и для сохранения текста в файле удобно использовать методы *LoadFromFile* и *SaveToFile* класса *TStrings*. Например, следующий обработчик события *On-Creat* формы *Formi* загружает в *Memol* текст проектного файла программы:

```
PROCEDURE TForm1.FormCreate(Sender: TObject);
BEGIN
Memol.Lines.LoadFromFile(
ChangeFileExt(Application.ExeName,'.dpr'));
END;
```

6.3.7. TButton – кнопка

Кнопки *TButton* широко используются для управления программами. Связанный с кнопкой алгоритм управления реализуется в обработчике события *OnClick.* Свойства компонента:

PROPERTY Cancel: Boolean;	Если имеет значение *True,* событие *OnClick* кнопки возникает при нажатии клавиши *Esc*
PROPERTY Default: Boolean;	Если имеет значение *True,* событие *OnClick* кнопки возникает при нажатии клавиши *Enter*
type TModalResult = Low(Integer)..High(Integer); **PROPERTY** ModalResult: TModalResult;	Определяет результат, с которым было закрыто модальное окно (см. ниже пояснение)

В терминологии *Windows* модальными окнами называются такие специальные окна, которые, раз появившись на экране, блокируют работу пользователя с другими окнами вплоть до своего закрытия. Обычно с их помощью реализуется диалог, требующий от пользователя принятия некоторого решения. Для этого в состав модального окна включается несколько кнопок. Если у кнопки определено свойство *Modal Result,* нажатие на нее приводит к закрытию модального окна и возвращает в программу значение *Modal Result* как результат диалога с пользователем. В *Delphi* определены следующие стандартные значения *Modal Result:*

mrNone Модальное окно не закрывается.
mrOk Была нажата кнопка *OK.*
mrCancel Была нажата кнопка *Cancel.*
mrAbort Была нажата кнопка *Abort.*
mrRetry Была нажата кнопка *Retry*
mrlgnore Была нажата кнопка *Ignore.*
mrYes Была нажата кнопка *Yes.*
mrNo Была нажата кнопка *No.*
mrAll Была нажата кнопка *All*

6.3.8. TCheckBox – независимый переключатель

Независимый переключатель *TCheckBox* используется для того, чтобы пользователь мог указать свое решение типа *Да/Нет* или *Да/Нет/Не знаю* (в последнем случае в окошке компонента устанавливается флаг выбора, но само окошко закрашивается серым цветом). Это решение отражается в свойстве *State* компонента, доступном как для чтения, так и для записи. В составе диалогового окна может быть несколько компонентов *TCheckBox.* Состояние любого из них не зависит от состояния остальных, поэтому такие переключатели называются независимыми. Типичное использование компонента:

```
IF CheckBox1. Checked THEN
. . . . .
ELSE
. . . . .
```

Или:

```
CASE CheckBox1. State OF
cbChecked: . . . . .;
cbUnchecked: . . . . .;
cbGrayed: . . . . .;
END;
```

Свойства компонента:

TYPE TLeftRight = (taLeftJustify, taRightJustify; **PROPERTY** Aligment: TLeftRight	Определяет положение текста: *taLeftJustify* – с левой стороны компонента; *taRightJustify* – с правой стороны
PROPERTY AllowGrayed: Boolean;	Разрешает/запрещает использование состояния *cbGrayed* (*не знаю*)
PROPERTY Caption: **STRING**;	Содержит связанный с компонентом текст.
PROPERTY Checked: Boolean;	Содержит выбор пользователя типа *Да/Нет.* Состояния *cbUnchecked* и *cbGrayed* отражаются как *False*
TYPE TCheckBoxState = (cbUnchecked, cbChecked, cbGrayed); **PROPERTY** State: TCheckBoxState;	Содержит состояние компонента: *cbUnchecked* – нет; *cbChecked* – да; *cbGrayed* – не знаю

Свойство *Color* компонента фактически игнорируется, а свойства *Height* и *Width* определяют размеры прямоугольника, в котором выводится связанный с переключателем текст, и не влияют на размеры прямоугольного окошка. Сам текст указывается в свойстве *Caption.*

6.3.9. TRadioButton – зависимые переключатели

В отличие от *TCheckBox* компоненты *TRadioButton* представляют собой зависимые переключатели, предназначенные для выбора одного из нескольких взаимоисключающих решений. На форму (точнее, в компонент-контейнер) помещается, по меньшей мере, два таких компонента. Они могут иметь только два состояния, определяемых свойством *Checked.* Если в одном компоненте это свойство принимает значение True, во всех других компонентах, расположенных в том же контейнере, свойства *Checked* принимают значения *False.*

Помимо свойства *Checked* компонент *TRadioButton* имеет еще одно специфичное свойство – *Alignment,* аналогичное такому же свойству *TCheckBox.* Как и в *TCheckBox,* программист не может изменять размеры и цвет круглого окошка компонента.

6.3.10. TListBox – список выбора

Компонент класса *TListBox* представляет собой стандартный для *Windows* список выбора, с помощью которого пользователь может выбрать один или несколько элементов выбора. В компоненте предусмотрена возможность программной прорисовки элементов, поэтому список может содержать не только строки, но и произвольные изображения.

Свойства компонента:

TYPE TBorderStyle = bsNone.. -bsSingle; **PROPERTY** Border-Style: TBorderStyle;	Определяет стиль рамки: *bsNone* – нет рамки; *bsSingle* – рамка толщиной 1 пиксель
PROPERTY Canvas: TCanvas;	Канва для программной прорисовки элементов
PROPERTY Columns: Longint;	Определяет количество колонок элементов в списке
PROPERTY ExtendedSelect: Boolean;	Если *ExtendedSelect=True* и *MultiSelect=True,* выбор элемента без одновременного нажатия *Crtl* или *Alt* отменяет предыдущий выбор
PROPERTY IntegralHeight: Boolean;	Если *IntegralHeight=True* и *StyleolbOwnerDrawVariabe,* в списке показывается целое число элементов
PROPERTY ItemHeight: Integer;	Определяет высоту элемента в пикселях для *Style =lbOwnerDrawFixed*
PROPERTY Itemlndex: Integer;	Содержит индекс сфокусированного элемента. Если *MultiSelect=False,* совпадает с индексом выделенного элемента
PROPERTY Items: TStrings;	Содержит набор строк, показываемых в компоненте
PROPERTY MultiSelect: Boolean;	Разрешает/отменяет выбор нескольких элементов
PROPERTY SelCount: Integer;	Содержит количество выбранных элементов

PROPERTY Selected [X: Integer Boolean-	Содержит признак выбора для элемента с индексом X (первый элемент имеет индекс 0).
PROPERTY Sorted: Boolean;	Разрешает/отменяет сортировку строк в алфавитном порядке.
TYPE TListBoxStyle = (IbStandard, IbOwnerDrawFixed, IbOwnerDrawVariable); **PROPERTY** Style: TListBoxStyle;	Определяет способ прорисовки элементов: *lbStandard* - элементы рисует *Windows; lbOwnerDrawFixed* - рисует программа, все элементы имеют одинаковую высоту, определяемую свойством *ItemHeight; IbOwnerDrawVariable* - рисует программа, элементы имеют разную высоту.
PROPERTY TabWidth: Integer;	Задает ширину табуляционного пробела.
PROPERTY Toplndex: Integer;	Индекс первого видимого в окне элемента.

Создание элементов (опций) списка компонента реализуется с помощью методов его свойства *Items - -Add, Append, Insert* или *LoadFromFile*

6.3.11. TComboBox – раскрывающийся список выбора

Комбинированный список *TComboBox* представляет собой комбинацию списка *TListBox* и редактора *Tedit,* и поэтому большинство его свойств и методов заимствованы у этих компонентов. Существуют пять модификаций компонента, определяемые его свойством *Style:* csSimple, csDropDown, csDropDownList, csOwnerDrawFixed и csOwnerDrawVariable. В первом случае список всегда раскрыт, в остальных – он раскрывается после нажатия кнопки справа от редактора. В модификации csDropDownList редактор работает в режиме отображения выбора и его нельзя использовать для ввода новой строки (в других модификациях это возможно). Модификации csOwnerDrawFixed – и csOwnerDrawVariable используются для программной прорисовки элементов списка. Используемые для этого свойства и методы полностью совпадают со свойствами и методами *TListBox* аналогичного назначения.

Фактически «своими» у компонента являются лишь свойства и события, связанные с раскрытием списка:

PROPERTY DropDownCount: Integer;
PROPERTY DroppedDown: Boolean;
PROPERTY OnDropDown: TNotifyEvent;

Свойство *DropDownCount* определяет количество элементов списка, появление которых еще не приводит к необходимости прокрутки списка. По умолчанию это свойство имеет значение 8: если в списке указано 9 и более элементов (т. е. больше чем содержит *DropDownCount*), при его раскрытии к окну будет добавлена полоса прокрутки. Свойство *DroppedDown* определяет, раскрыт ли в данный момент список. Это свойство доступно также для записи, что позволяет программно управлять состоянием списка. Состояние *OnDropDown* происходит при изменении состояния списка.

6.3.12. TScrollBar – управление значением величины

Компонент *TScrollBar* представляет собой стандартный для *Windows* управляющий элемент, похожий на полосу прокрутки окна. Обычно он используется для визуального управления значением числовой величины.

Свойства компонента:

TSrollBarKind = (sbHorizontal, sbVertical); **PROPERTY** Kind: TScrollBarKind**;**	Определяет ориентацию компонента: *sbHorizontal* – бегунок перемещается по горизонтали; *sbVertical* – бегунок перемещается по вертикали

PROPERTY LargeChange: TScrollBarinc;	«Большой» сдвиг бегунка (при щелчке мышью рядом с концевой кнопкой)
PROPERTY Max: Integer;	Максимальное значение диапазона изменения числовой величины
PROPERTY Min: Integer;	Минимальное значение диапазона изменения числовой величины
PROPERTY Position: Integer;	Текущее значение числовой величины
PROPERTY SmallChange: TScrollBarinc;	«Малый» сдвиг бегунка (при щелчке мышью по концевой кнопке

С помощью метода:
PROCEDURE SetParams(APosition, AMax, AMin: Integer);
можно сразу установить свойства *Position, Max* и *Min.*

Подобно *TButton* компонентом полностью управляет *Windows,* поэтому у него нет свойства *Color.*

6.3.13. TGroupBox – панель группирования

Этот компонент служит контейнером для размещения дочерних компонентов и представляет собой прямоугольное окно с рамкой и текстом в разрыве рамки. Обычно с его помощью выделяется группа управляющих элементов, объединенных по функциональному назначению. Свойства и методы этого класса целиком унаследованы им от своих предков *TCustomControl* и *TWinControl*.

6.3.14. TRadioGroup – группа зависимых переключателей

Компонент класса *TRadioGroup* представляет собой специальный контейнер, предназначенный для размещения зависимых переключателей класса *TRadioButton.* Каждый размещаемый в нем переключатель помещается в специальный список *Items* и доступен по индексу, что упрощает обслуживание группы.

Свойства компонента:

PROPERTY Columns: Integer;	Определяет количество столбцов переключателей
PROPERTY Itemlndex: Integer;	Содержит индекс выбранного переключателя
PROPERTY Items: TStrings;	Содержит список строк с заголовками элементов. Добавление/удаление элементов достигается добавлением/удалением строк списка *Items*

После размещения компонента на форме он пуст. Чтобы создать в нем хотя бы один переключатель, следует раскрыть редактор списка *Items* и ввести хотя бы одну строку: строки *Items* используются как поясняющие надписи справа от переключателей, а их количество определяет количество переключателей в группе. Следует заметить также, что после создания компонента его свойство *Itemlndex* по умолчанию имеет

значение -1, это означает, что ни один переключатель в группе не выбран. Если в момент появления компонента на экране в каком-то переключателе выбор уже должен быть установлен, необходимо на этапе конструирования с помощью окна Инспектора Объектов или программно установить в свойство *Itemlndexk* номер соответствующего переключателя (нумерация начинается с 0). Это же свойство позволяет программе проанализировать выбор пользователя.

6.3.15. TPanel – панель

Компонент *TPanel* (панель) представляет собой контейнер общего назначения. В отличие от *TGroupBox* он не имеет заголовка и поэтому менее удобен для функционального группирования элементов. С другой стороны, его свойство *Caption* отображается в виде текстовой строки и может использоваться для вывода сообщений. Компоненты этого класса часто помещаются на форму для того, чтобы располагать вставленные в них дочерние компоненты вдоль одной из сторон окна независимо от изменения размеров этого окна.

Компонент имеет развитые средства создания различных эффектов трехмерности за счет использующихся в нем двух кромок – внешней и внутренней.

Свойства компонента:

TBorderStyle = bsNone..bsSingle; **PROPERTY** BorderStyle: TBorderStyle;	Определяет стиль рамки: *bsNone* – нет рамки; *bsSingle* – компонент по периметру обводится линией толщиной в 1 пиксель
PROPERTY FullRepaint: Boolean	Разрешает/запрещает перерисовку панели и всех ее дочерних элементов при изменении ее размеров
PROPERTY Locked: Boolean;	Используется при работе с объектами *OLE*

Для компонента объявлено событие *OnResize,* в обработчике которого программист может предусмотреть необходимую реакцию на изменение размеров компонента.

6.3.16. TActionList – механизм действий

Этот компонент впервые введен в *Delphi 4*. Он не имеет видимого изображения и служит для поддержки механизма действий. Основная схема его использования такова. Вначале с помощью его редактора создается *действие* – объект класса *TAction* (редактор вызывается двойным щелчком на компоненте либо с помощью опции *Action List Editor* его вспомогательного меню, которое раскрывается после щелчка на нем правой кнопкой мыши).

Этот объект (на рисунке он имеет умалчиваемое имя Action1) имеет ряд свойств и событий, с помощью которых уточняется характер действия. Доступ к этим свойствам и событиям можно получить с помощью окна Инспектора Объектов. С действием можно связать группу свойств (*Caption, Checked, Enabled, ShortCut* и т. д.), которые будут помещаться в одноименные свойства компонентов, реализующих общее действие. Если с компонентом связан контейнер пиктограмм типа *TImageList* (свойство *Images* – не действия, а самого компонента *TActionList),* при реализации действия можно использовать одну из хранящихся в нем пиктограмм *(ImageIndex).* Чтобы действие подкреплялось программным кодом, для него обязательно следует определить обработчик события *OnExecute.*

Свойства компонента:

PROPERTY ActionCount: Integer;	Содержит количество определенных в компоненте действий (только для чтения)
PROPERTY Actions[Index TcontainedAction: Integer]:	Позволяет программе обратиться к нужному действию (объекту класса *TcontainedAclion)* по его индексу *Index*
PROPERTY Images: TCustomImageList;	Содержит имя компонента класса *TImageList*

Содержит имя компонента класса *TImageList.*

Редактор компонента создает объекты класса *TAction.* Свойства класса *TAction,* в основном, определяют те значения, которые будут иметь поименные свойства всех компонентов или опций меню, связанных одним действием.

Свойства *TAction:*

PROPERTY Caption: String;	Содержит строку, которая будет устанавливаться в свойствах *Caption* всех компонентов, связанных данным действием
PROPERTY Checked: Boolean;	Содержит значение, которое будет устанавливаться в свойствах *Checked*
PROPERTY DisableIfNoHandler: Boolean;	Указывает, будут ли запрещены для выбора связанные компоненты, если для действия не определен обработчик *OnExecute*
PROPERTY Enabled: Boolean;	Содержит значение, которое будет устанавливаться в свойствах *Enabled*
PROPERTY HelpContext: elpContext;	Содержит значение, которое будет устанавливаться в свойствах *HelpContext*

PROPERTY Hint: String;	Содержит строку, которая будет устанавливаться в свойствах *Hint*
PROPERTY Imagelndex: Integer;	Содержит индекс изображения в хранилище, указанном в свойстве *Images* компонента. Это изображение будет связано с компонентами данного действия
PROPERTY Shortcut: TShortCut;	Содержит значение, которое будет устанавливаться в свойствах *ShortCut*
PROPERTY Visible: Boolean	Содержит значение, которое будет устанавливаться в свойствах *Visible*

События *TAction:*

PROPERTY OnExecute: TnotifyEvent;	Возникает при щелчке мышью на одном из компонентов, связанных общим действием. Обработчик этого события должен реализовать нужное действие
ThintEvent = **PROCEDURE** (**VAR** HintStr: **STRING**; **VAR** CanShow: Boolean) **OF OBJECT**; **PROPERTY** OnHint: THintEvent;	Возникает при перемещении указателя мыши над одним из связанных общим действием компонентов. Его умалчиваемый обработчик создает окно оперативной подсказки и показывает в нем строку *HintStr*
PROPERTY OnUpdate: TnotifyEvent;	Возникает, когда очередь сообщений для приложения пуста или когда обновляется содержимое списка действий

7. БАЗЫ ДАННЫХ И БАЗЫ ЗНАНИЙ. СЕТЬ ИНТЕРНЕТ

7.1. Системы сбора, хранения и обработки информации о протекании промышленного процесса

Эффективность управления промышленными процессами существенно зависит от глубины проработки проблем контроля и анализа производственных данных. Ядром существующих систем контроля технологических процессов, предоставляющих пользователям различные инструменты мониторинга и анализа производства, являются различного рода базы данных (БД).

7.1.1. Общая характеристика систем управления базами данных

В самом общем смысле база данных – это набор записей и файлов, организованных специальным образом. В компьютере, например, можно хранить данные о поступающем на завод сырье. Один из типов баз данных – это документы, набранные с помощью текстовых редакторов и сгруппированные по темам. Другой тип – файлы электронных таблиц, объединяемые в группы по характеру их использования.

С ростом популярности систем управления базами данных (СУБД) в 70–80-х годах появилось множество различных моделей данных. У каждой из них имелись свои достоинства и недостатки, которые сыграли ключевую роль в развитии реляционной модели данных, появившейся во многом благодаря стремлению упростить и упорядочить первые модели данных.

До появления СУБД все данные, которые содержались в компьютерной системе постоянно, хранились в виде отдельных файлов. Система управления файлами, которая обычно является частью операционной системы компьютера, следила за именами файлов и местами их расположения. В системах управления файлами модели данных, как правило, не использовались; эти системы ничего не знали о внутреннем содержимом файлов. Для такой системы файл, содержащий данные о составе сырья ничем не отличается от файла содержащего данные об используемом катализаторе.

Знание о содержимом файла – какие данные в нём хранятся и какова их структура – было уделом прикладных программ, использующих этот файл, что иллюстрирует рис. 7.1. В приложении для расчета реакторного процесса каждая из программ, обрабатывающих файл с информацией о составе сырья, содержит в себе описание структуры данных (ОСД), хранящихся в этом файле.

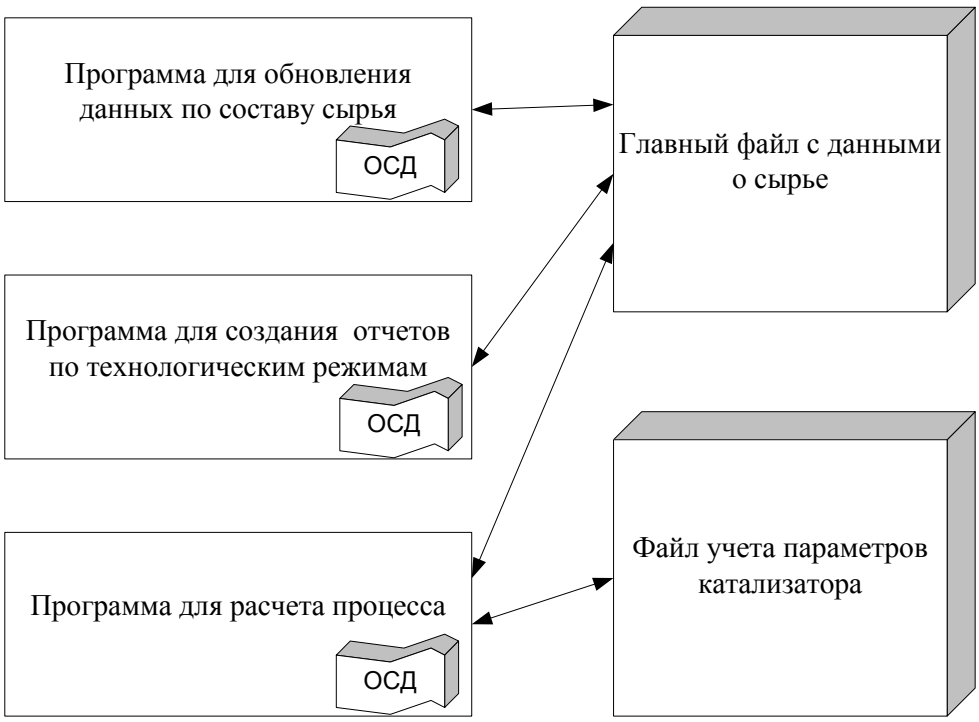


Рис. 7.1. Приложение для расчета процесса, использующее систему управления файлами

Когда структура данных изменялась, например, в случае добавления нового элемента данных в составе сырья, необходимо было модифицировать каждую из программ, обращавшихся к файлу. Со временем количество файлов и программ росло, и на сопровождение существующих приложений приходилось затрачивать всё больше и больше усилий, что замедляло разработку новых приложений.

Проблемы сопровождения больших систем, основанных на файлах, привели в конце 60-х годов к появлению СУБД. В основе СУБД лежала простая идея: изъять из программ определение структуры содержимого файла и хранить её вместе с данными в базе данных.

7.1.2. Иерархические СУБД

Одной из наиболее важных сфер применения первых СУБД было планирование производства для компаний, занимающихся выпуском продукции. Например, если НПЗ планировал выпустить за определенный период 10000 т линейного алкилбензола марки А и 15000 т линейного алкилбензола марки Б, необходимо было знать, какое количество сырьевой следует получить с соседних установок. Чтобы ответить на этот вопрос, необходимо сопоставить данные о составе сырья с данными по технологии производства конечного продукта.

Список составных частей изделия по своей природе является иерархической структурой. Для хранения данных, имеющих такую структуру, была разработана иерархическая модель данных, которую иллюстрирует рис. 7.2.

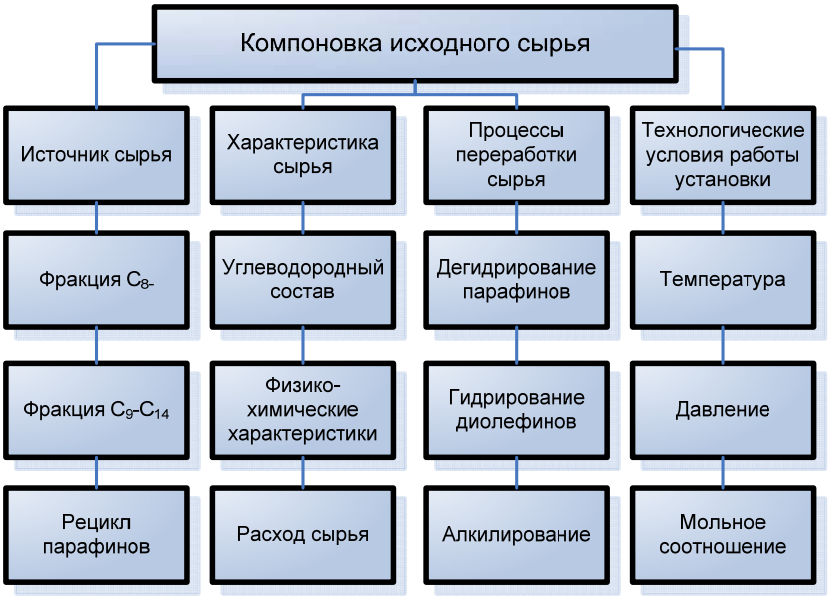


Рис. 7.2. Иерархическая база данных, содержащая информацию о компоновке исходного сырья

В этой модели каждая запись базы данных представляла конкретную деталь. Между записями существовали отношения «предок / потомок».

Для чтения данных из иерархической базы данных требовалось перемещаться по записям, за один раз переходя на одну запись вверх, вниз или в сторону.

Иерархическая модель базы данных состоит из объектов с указателями от родительских объектов к потомкам, соединяя вместе связанную информацию. Такие базы данных могут быть представлены как дерево, состоящее из объектов различных уровней. Верхний уровень занимает один объект, второй – объекты второго уровня и т. д. Между объектами существуют связи, каждый объект может включать в себя несколько объектов более низкого уровня. Такие объекты находятся в отношении предка (объект более близкий к корню) к потомку (объект более низкого уровня), при этом возможно, когда объект-предок не имеет потомков или имеет их несколько, тогда как у объекта-потомка обязательно только один предок. Объекты, имеющие общего предка, называются близнецами.

В этой модели запрос, направленный вниз по иерархии, прост; однако запрос, направленный вверх по иерархии, более сложен. Также

трудно представить неиерархические данные при использовании этой модели.

Иерархической базой данных является файловая система, состоящая из корневой директории, в которой имеется иерархия поддиректорий и файлов.

Типичным представителем иерархической СУБД является Information Management System (IMS) фирмы IBM, появившаяся в 1968 году. Преимущества IMS и реализованной в ней иерархической модели:

- Простота модели. Принцип построения IMS был легок для понимания. Иерархия базы данных напоминала структуру компании или генеалогическое дерево.
- Использование отношений предок/потомок. СУБД IMS позволяла легко представлять отношения предок/потомок, например: «А является частью В» или «А владеет В».
- Быстродействие. В СУБД IMS отношения «предок / потомок» были реализованы в виде физических указателей из одной записи на другую, вследствие чего перемещение по базе данных происходило быстро. Поскольку структура данных в этой СУБД отличалась простотой, IMS могла размещать записи предков и потомков на диске рядом друг с другом, что позволяло свести к минимуму количество операций записи-чтения.

СУБД IMS все ещё является одной из наиболее распространённых СУБД для больших ЭВМ компании IBM. Доля мэйнфреймов этой компании, на которых используется данная СУБД, превышает 25 %.

Другие известные представители:

- Time-Shared Date Management System (TDMS) компании Development Corporation;
- Mark IV Multi – Access Retrieval System компании Control Data Corporation;
- System – 2000 разработки SAS-Institute;
- Серверы каталогов, такие как LDAP и Active Directory;

Кроме того по принципу иерархической БД построен и Реестр Windows.

7.1.3. Сетевые базы данных

Если структура данных оказывалась сложнее, чем обычная иерархия, простота структуры иерархической базы данных становилась её недостатком. Например, в базе данных для хранения заказов один заказ мог участвовать в трёх *различных* отношениях «предок / потомок», связывающих заказ с клиентом, разместившим его, с процессом, в ходе которого получается продукт, и с сырьем, из которого будет получен ко-

нечный продукт. Такие структуры данных не соответствовали строгой иерархии IMS.

В связи с этим для таких приложений, как обработка заказов, была разработана новая сетевая модель данных. Она являлась улучшенной иерархической моделью, в которой одна запись могла участвовать в нескольких отношениях «предок / потомок». В сетевой модели такие отношения назывались множествами. В 1971 г. на конференции по языкам систем данных был опубликован официальный стандарт сетевых баз данных, который известен как модель CODASYL. Компания IBM не стала разрабатывать собственную сетевую СУБД и вместо этого продолжала наращивать возможность IMS. Но в 70-х гг. независимые производители программного обеспечения реализовали сетевую модель в таких продуктах, как IDMS компании Cullinet, Total компании Cincom и СУБД Adabas, которые приобрели большую популярность.

К основным понятиям сетевой модели базы данных относятся: уровень, элемент (узел), связь. Узел – это совокупность атрибутов данных, описывающих некоторый объект. На схеме иерархического дерева узлы представляются вершинами графа. В сетевой структуре каждый элемент может быть связан с любым другим элементом.

Сетевые базы данных подобны иерархическим, за исключением того, что в них имеются указатели в обоих направлениях, которые соединяют родственную информацию.

Несмотря на то что эта модель решает некоторые проблемы, связанные с иерархической моделью, выполнение простых запросов остается достаточно сложным процессом.

Также, поскольку логика процедуры выборки данных зависит от физической организации этих данных, то эта модель не является полностью независимой от приложения. Другими словами если необходимо изменить структуру данных, то нужно изменить и приложение.

Примеры сетевых СУБД: СООБЗ Cerebrum.

Сетевые базы данных обладали рядом преимуществ:

- Гибкость. Множественные отношения «предок / потомок» позволяли сетевой базе данных хранить данные, структура которых была сложнее простой иерархии.
- Стандартизация. Появление стандарта CODASYL популярность сетевой модели, а такие поставщики мини-компьютеров, как Digital Equipment Corporation и Data General, реализовали сетевые СУБД.
- Быстродействие. Вопреки своей большой сложности, сетевые базы данных достигали быстродействия, сравнимого с быстродействием иерархических баз данных. Множества были представлены указателями

на физические записи данных, и в некоторых системах администратор мог задать кластеризацию данных на основе множества отношений.

Конечно, у сетевых баз данных были недостатки. Как и иерархические базы данных, сетевые базы данных были очень жесткими. Наборы отношений и структуру записей приходилось задавать наперёд. Изменение структуры базы данных обычно означало перестройку всей базы данных.

Как иерархическая, так и сетевая база данных были инструментами программистов. Реализация пользовательских запросов часто затягивалась на недели и месяцы, и к моменту появления программы информация, которую она предоставляла, часто оказывалась бесполезной.

7.1.4. Реляционные базы данных

Недостатки иерархической и сетевой моделей привели к появлению новой, реляционной модели данных, созданной Коддом в 1970 году и вызвавшей всеобщий интерес. Слово «реляционный» происходит от английского «relation» (отношение). Отношение – это формальное название того, что принято называть таблицей.

Реляционная модель была попыткой упростить структуру базы данных. В ней отсутствовали явные указатели на предков и потомков, а все данные были представлены в виде простых таблиц, разбитых на строки и столбцы.

Реляционной называется база данных, в которой все данные, доступные пользователю, организованы в виде таблиц, а все операции над данными сводятся к операциям над этими таблицами.

В реляционной базе данных информация организована в виде *таблиц*, разделённых на строки и столбцы, на пересечении которых содержатся значения данных. У каждой таблицы имеется уникальное имя, описывающее её содержимое. Более наглядно структуру таблицы иллюстрирует таблица 7.1.1. Каждая горизонтальная *строка* этой таблицы представляет отдельную физическую сущность – один параметр. Все строки таблицы вместе дают полную информацию. Все данные, содержащиеся в конкретной строке таблицы, относятся к параметру, который описывается этой строкой.

Каждый вертикальный *столбец* таблицы представляет один элемент данных (дату).

Таблица 7.1.1

Реляционная база данных, содержащая информацию о сырье

Дата	Тип сырья	C_9H_{20}	$C_{10}H_{22}$	……	$C_{14}H_{30}$	ЛАБ	Остаток
Сентябрь 2007 г.	1	0,00	16,45	……	0,32	0,07	3,42
Январь 2008 г.	2	0,00	13,09	……	0,34	0,13	2,96
Март 2008 г.	3	0,01	17,64	……	0,48	0,14	4,90

На пересечении каждой строки с каждым столбцом таблицы содержится в точности одно значение данных. Все значения, содержащиеся в одном и том же столбце, являются данными одного типа. Множество значений, которые могут содержаться в столбце, называется доменом этого столбца.

У каждого столбца в таблице есть своё *имя*, которое обычно служит заголовком столбца. Все столбцы в одной таблице должны иметь уникальные имена, однако разрешается присваивать одинаковые имена столбцам, расположенным в различных таблицах.

Столбцы таблицы упорядочены слева направо, и их порядок определяется при создании таблицы. В любой таблице всегда есть как минимум один столбец. В стандарте ANSI / ISO не указывается максимально допустимое число столбцов в таблице, однако почти во всех коммерческих СУБД этот предел существует и обычно составляет до 255 столбцов.

В отличие от столбцов, строки таблицы не имеют определённого порядка. Это значит, что если последовательно выполнить два одинаковых запроса для отображения содержимого таблицы, нет гарантии, что оба раза строки будут перечислены в одном и том же порядке.

В таблице может содержаться любое количество строк. Вполне допустимо существование таблицы с нулевым количеством строк. Такая таблица называется пустой. Пустая таблица сохраняет структуру, определённую её столбцами, просто в ней не содержатся данные. Стандарт ANSI / ISO не накладывает ограничений на количество строк в таблице, и во многих СУБД размер таблиц ограничен лишь свободным дисковым пространством компьютера. В других СУБД имеется максимальный предел, однако он весьма высок – около двух миллиардов строк, а иногда и больше.

Базовыми понятиями реляционных СУБД являются: 1) атрибут; 2) отношения; 3) кортеж. Атрибут в реляционных базах данных – элемент данных в кортеже. Часто используется менее строгий термин поле. Кортеж в реляционных базах данных – элемент отношения, строка таблицы; упорядоченный набор из *N* элементов.

Таким образом, особенности реляционной базы данных кратко можно сформулировать следующим образом:

- данные хранятся в таблицах, состоящих из столбцов («атрибутов») и строк («записей», «кортежей»);
- на пересечении каждого столбца и строчки стоит в точности одно значение;
- у каждого столбца есть своё имя, которое служит его названием, и все значения в одном столбце имеют один тип;

- запросы к базе данных возвращают результат в виде таблиц, которые тоже могут выступать как объект запросов.

Каждая реляционная таблица представляет собой двумерный массив и обладает следующими свойствами:

- каждый элемент таблицы – один элемент данных;
- все столбцы в таблице однородные, то есть все элементы в столбце имеют одинаковый тип (числовой, символьный и т. д.);
- каждый столбец имеет уникальное имя;
- одинаковые строки в таблице отсутствуют;
- порядок следования строк и столбцов может быть произвольным.

Общепринятым стандартом языка работы с реляционными базами данных является язык SQL.

Реляционные модели характеризуются простотой структуры данных, удобным для пользователя табличным представлением и возможностью использования формального аппарата алгебры отношений и реляционного исчисления для обработки данных.

7.1.5. Среда быстрой разработки приложений Delphi

Современные масштабы разработки программного обеспечения немыслимы без средств RAD.

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 в. RAD получила широкое распространение и одобрение. Концепцию RAD также часто связывают с концепцией визуального программирования.

Основные принципы создания RAD-проектов:

- инструментарий должен быть нацелен на минимизацию времени разработки;
- создание прототипа для уточнения требований заказчика;
- цикличность разработки: каждая новая версия продукта основывается на оценке результата работы предыдущей версии заказчиком;
- минимизация времени разработки версии, за счёт переноса уже готовых модулей и добавления функциональности в новую версию;
- команда разработчиков должна тесно сотрудничать, каждый участник должен быть готов выполнять несколько обязанностей;
- управление проектом должно минимизировать длительность цикла разработки.

Концепция RAD стала ответом на неуклюжие методы разработки программ 1970-х и начала 1980-х гг. Эти методы предусматривали настолько медленный процесс создания программы, что зачастую даже требования к программе успевали измениться до окончания разработки. Основателем RAD считается сотрудник IBM Д. Мартин, который в 1980-х гг. сформулировал основные принципы RAD, основываясь на идеях Б. Бойма и С. Шульца. В настоящее время RAD становится общепринятой схемой для создания средств разработки программных продуктов. Именно средства разработки, основанные на RAD, имеют наибольшую популярность среди программистов.

Среды разработки, использующие принципы RAD:

- Borland Delphi.
- Borland C++ Builder.
- IBM Lotus Domino Designer.
- Microsoft Visual Studio.
- Macromedia Flash и др.

Одной из самых распространенных сред RAD является Delphi.

Borland Delphi – это интегрированная среда разработки программного обеспечения фирмы Borland, использует язык программирования Delphi (начиная с 7 версии язык в среде именуется Delphi, ранее – Object Pascal), разработанный фирмой Borland и изначально реализованный в её пакете Borland Delphi, от которого и получил в 2003 году своё нынешнее название. Object Pascal по сути является наследником языка Pascal с объектно-ориентированными расширениями.

Delphi – результат развития языка Турбо Паскаль, который, в свою очередь, развился из языка Паскаль. Паскаль был полностью процедурным языком, Турбо Паскаль начиная с версии 5.5 добавил в Паскаль объектно-ориентированные свойства, а Delphi – объектно-ориентированный язык программирования с возможностью доступа к метаданным классов (то есть к описанию классов и их членов) в компилируемом коде, также называемом интроспекцией. Де-факто Object Pascal, а затем и язык Delphi являются функциональными наращиваниями Turbo Pascal.

Среди многих распространенных программных продуктов, сделанных на Delphi, можно найти:

- продукция Borland: Borland Delphi, Borland C++ Builder;
- администрирование / разработка баз данных: MySQL Tools (Administrator, Query Browser), TOAD;
- инженерное ПО: Altium Designer / Protel (проектирование электроники);
- доставка информации в Интернете: Skype (VoIP и IM), QIP и QIP Infium, The Bat! (клиент электронной почты).

В процессе построения приложения в Delphi разработчик выбирает из палитры компонент готовые компоненты. Основной упор в объектно-ориентированной модели программных компонент в Delphi делается на максимальном использовании кода. Это позволяет разработчикам строить приложения весьма быстро из заранее подготовленных объектов, а также дает им возможность создавать свои собственные объекты для среды Delphi.

Среда Delphi включает в себя полный набор визуальных инструментов для скоростной разработки приложений, поддерживающей разработку пользовательского интерфейса и подключение к корпоративным базам данных. VCL – библиотека визуальных компонент, включает в себя стандартные объекты построения пользовательского интерфейса, объекты управления данными, графические объекты, объекты мультимедиа, диалоги и объекты управления файлами, управление OLE.

На рис. 7.3 приведено основное окно среды Delphi во время разработки моделирующей программы для процессов производства ЛАБ.

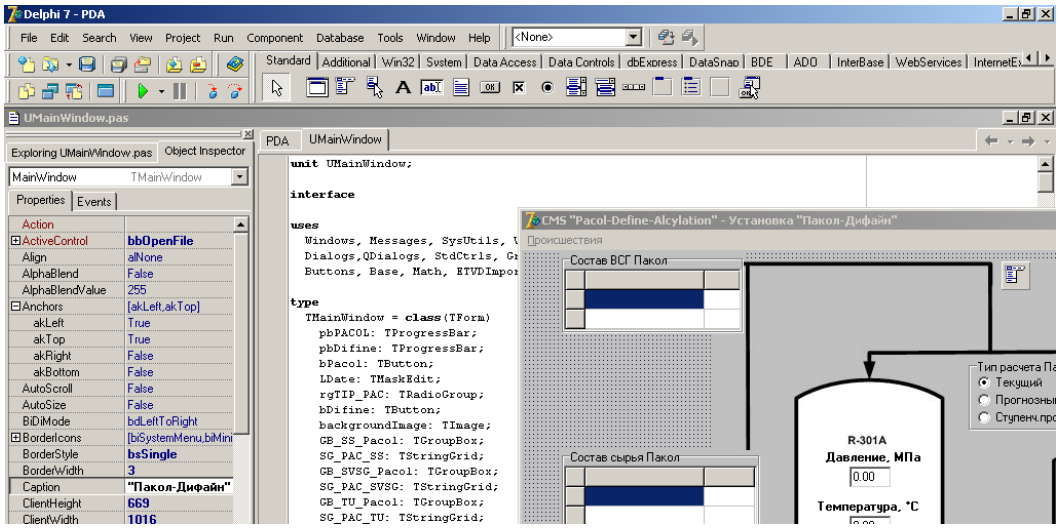


Рис. 7.3. Основное окно среды Delphi во время разработки моделирующей программы для процессов производства ЛАБ

Объекты баз данных в Delphi основаны на SQL. В состав Delphi также включен Borland SQL Link, поэтому доступ к СУБД Oracle, Sybase, Informix и InterBase происходит с высокой эффективностью. Кроме того, Delphi включает в себя локальный сервер Interbase для того, чтобы можно было разработать расширяемые на любые внешние SQL-сервера приложения. Разработчик в среде Delphi, проектирующий информационную систему, может использовать для хранения информации файлы формата *.dbf (как в dBase или Clipper) или *.db (Paradox).

Готовое приложение может быть изготовлено либо в виде исполняемого модуля, либо в виде динамической библиотеки, которую можно использовать в приложениях, написанных на других языках программирования.

Благодаря открытой компонентной архитектуре приложения, изготовленные при помощи Delphi, работают надежно и устойчиво. Delphi поддерживает использование уже существующих объектов, включая DLL, написанные на C и C++, OLE сервера, VBX, объекты, созданные при помощи Delphi. Кроме того, поскольку Delphi имеет полностью объектную ориентацию, разработчики могут создавать свои повторно используемые объекты для того, чтобы уменьшить затраты на разработку.

Библиотека визуальных компонент включает в себя стандартные объекты построения пользовательского интерфейса, объекты управления данными, графические объекты, объекты мультимедиа, диалоги и объекты управления файлами, управление DDE и OLE.

Delphi позволяет разработчикам настроить среду для максимального удобства. Возможно легко изменить палитру компонент, инструментальную линейку, а также настраивать выделение синтаксиса цветом.

7.1.6. Язык SQL как стандартный язык реляционных БД

Стремительный рост популярности SQL является одной из самых важных тенденций в современной компьютерной промышленности. За несколько последних лет SQL стал единственным языком баз данных. Официальный международный стандарт на SQL был принят в 1987 г., обновлен в 2003 г. Язык SQL является важным звеном в архитектуре систем управления базами данных, выпускаемых всеми ведущими поставщиками программных продуктов, и служит стратегическим направлением разработок компании Microsoft в области баз данных. Зародившись в результате выполнения второстепенного исследовательского проекта компании IBM, SQL сегодня широко известен и в качестве мощного рыночного фактора.

SQL является инструментом, предназначенным для обработки и чтения данных, содержащихся в компьютерной базе данных. SQL – это сокращенное название структурированного языка запросов (Structured Query Language – язык структурированных запросов). Как следует из названия, SQL является языком программирования, который применяется для организации взаимодействия пользователя с базой данных. На самом деле SQL работает только с реляционными базами данных. На рисунке 7.4 изображена схема работы SQL. Согласно этой схеме, в вычислительной системе имеется база данных, в которой хранится важная информация. Компьютерная программа, которая управляет базой данных, называется системой управления базой данных, или СУБД.

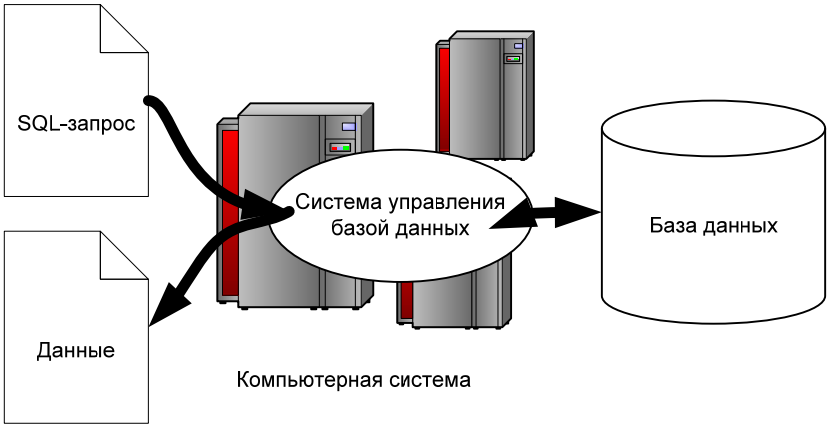


Рис. 7.4. Применение SQL для доступа к базе данных

Если пользователю необходимо прочитать данные из базы данных, он запрашивает их у СУБД с помощью SQL. СУБД обрабатывает запрос, находит требуемые данные и посылает их пользователю. Процесс запрашивания данных и получения результата называется запросом к базе данных: отсюда и название – структурированный язык запросов.

Однако это название не совсем соответствует действительности. Во-первых, сегодня SQL представляет собой нечто гораздо большее, чем простой инструмент создания запросов, хотя именно для этого он и был первоначально предназначен. Несмотря на то, что чтение данных по-прежнему остается одной из наиболее важных функций SQL, сейчас этот язык используется для реализации всех функциональных возможностей, которые СУБД предоставляет пользователю, а именно:

- Организация данных. SQL дает пользователю возможность изменять структуру представления данных, а также устанавливать отношения между элементами базы данных.
- Чтение данных. SQL дает пользователю или приложению возможность читать из базы данных содержащиеся в ней данные и пользоваться ими.
- Обработка данных. SQL дает пользователю или приложению возможность изменять базу данных, т. е. добавлять в нее новые данные, а также удалять или обновлять уже имеющиеся в ней данные.
- Управление доступом. С помощью SQL можно ограничить возможности пользователя по чтению и изменению данных и защитить их от несанкционированного доступа.
- Совместное использование данных. SQL координирует совместное использование данных пользователями, работающими параллельно, чтобы они не мешали друг другу.
- Целостность данных. SQL позволяет обеспечить целостность базы данных, защищая ее от разрушения из-за несогласованных изменений или отказа системы.

Таким образом, SQL является достаточно мощным языком для взаимодействия с СУБД.

Вопреки существующим заблуждениям, SQL является информационно-логическим языком, а не языком программирования.

В SQL входит около тридцати операторов, предназначенных для управления базами данных. Операторы SQL встраиваются в базовый язык, например COBOL, FORTRAN или C, и дают возможность получать доступ к базам данных. Кроме того, из такого языка, как C, операторы SQL можно посылать СУБД в явном виде, используя интерфейс вызовов функций.

Наконец, SQL – это слабо структурированный язык, особенно по сравнению с такими сильно структурированными языками, как C или Pascal. Операторы SQL напоминают английские предложения и содержат «слова-пустышки», не влияющие на смысл оператора, но облегчающие его чтение. В SQL почти нет нелогичностей, к тому же имеется ряд специальных правил, предотвращающих создание операторов SQL, которые выглядят как абсолютно правильные, но не имеют смысла.

Несмотря на не совсем точное название, SQL на сегодняшний день является единственным стандартным языком для работы с реляционными базами данных. SQL – это достаточно мощный и в то же время относительно легкий для изучения язык.

На рис. 7.5 изображена структурная схема типичной СУБД, компоненты которой соединяются в единое целое с помощью SQL.

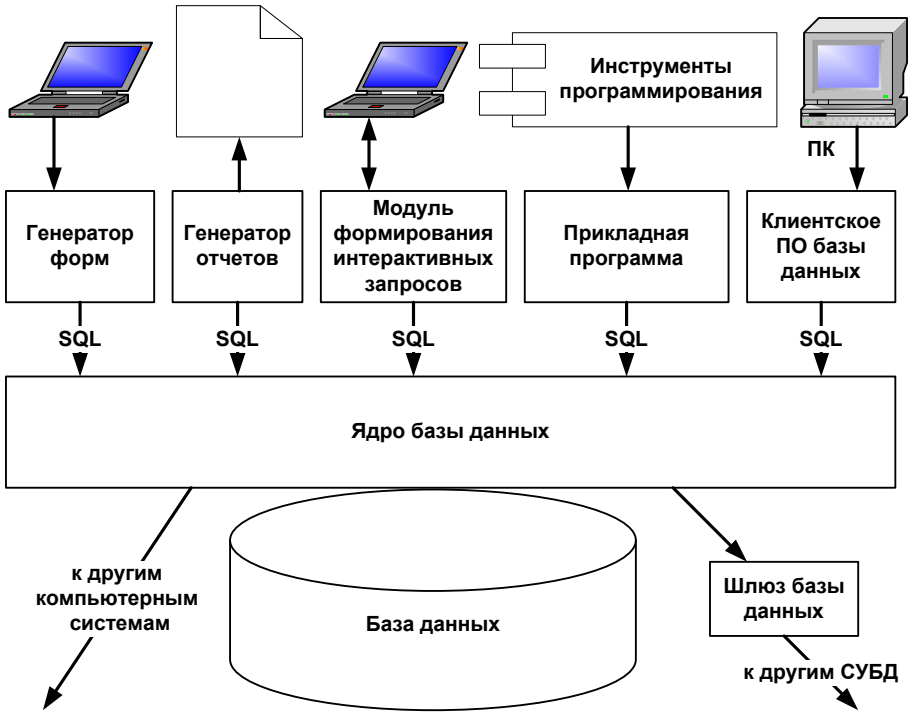


Рис. 7.5. Компоненты типичной СУБД

Ядро базы данных является сердцевиной СУБД; оно отвечает за физическое структурирование и запись данных на диск, а также за физическое чтение данных с диска. Кроме того, оно принимает SQL-запросы от других компонентов СУБД (таких как генератор форм, генератор отчетов или модуль формирования интерактивных запросов), от пользовательских приложений и даже от других вычислительных систем. Как видно из рисунка, SQL выполняет много различных функций.

- SQL – интерактивный язык запросов. Пользователи вводят команды SQL в интерактивные программы, предназначенные для чтения данных и отображения их на экране. Это удобный способ выполнения специальных запросов.
- SQL – язык программирования баз данных. Чтобы получить доступ к базе данных, программисты вставляют в свои программы команды SQL. Эта методика используется как в программах, написанных пользователями, так и в служебных программах баз данных (таких как генераторы отчетов и инструменты ввода данных).
- SQL – язык администрирования баз данных. Администратор базы данных, находящейся на мини-компьютере или на большой ЭВМ, использует SQL для определения структуры базы данных и управления доступом к данным.
- SQL – язык создания приложений клиент / сервер, и программа для персональных компьютеров SQLкоторый используется для организации связи через локальную сеть с сервером базы данных, в которой хранятся совместно используемые данные. В большинстве новых приложений используется архитектура клиент / сервер, которая позволяет свести к минимуму сетевой трафик и повысить быстродействие как персональных компьютеров, так и серверов баз данных.
- SQL – язык распределенных баз данных. В системах управления распределенными базами данных SQL помогает распределять данные среди нескольких взаимодействующих вычислительных систем. Программное обеспечение каждой системы посредством использования SQL связывается с другими системами, посылая им запросы на доступ к данным.
- SQL – язык шлюзов базы данных. В вычислительных сетях с различными СУБД SQL часто используется в шлюзовой программе, которая позволяет СУБД одного типа связываться с СУБД другого типа.

Таким образом, SQL превратился в полезный и мощный инструмент, обеспечивающий людям, программам и вычислительным системам доступ к информации, содержащейся в реляционных базах данных.

Успех языку SQL принесли следующие его особенности:

- Независимость от конкретной СУБД.

Несмотря на наличие диалектов и различий в синтаксисе, в большинстве своём тексты SQL-запросов могут быть достаточно легко перенесены из одной СУБД в другую. Существуют системы, разработчики которых изначально закладывались на применение, по меньшей мере, нескольких СУБД (например, система электронного документооборота Documentum может работать как с Oracle Database, так и с Microsoft SQL Server)

- Наличие стандартов.

Наличие стандартов и набора тестов для выявления совместимости и соответствия конкретной реализации SQL общепринятому стандарту только способствует «стабилизации» языка.

- Декларативность.

С помощью SQL программист описывает только то, какие данные нужно извлечь или модифицировать. То, каким образом это сделать решает СУБД непосредственно при обработке SQL запроса.

Все перечисленные выше факторы явились причиной того, что SQL стал стандартным инструментом для управления данными на персональных компьютерах, мини-компьютерах и больших ЭВМ.

Однако SQL не лишен и ряда недостатков:

- Несоответствие реляционной модели данных. Создатель реляционной модели данных Э. Кодд, К. Дейт и их сторонники указывают на то, что SQL не является истинно реляционным языком. В частности они указывают на следующие проблемы SQL:
  - повторяющиеся строки;
  - неопределённые значения;
  - явное указание порядка колонок слева направо;
  - колонки без имени и дублирующиеся имена колонок;
  - отсутствие поддержки свойства «=»;
  - использование указателей;
  - высокая избыточность.
- Сложность. Хотя SQL и задумывался как средство работы конечного пользователя, в конце концов, он стал настолько сложным, что превратился в инструмент программиста.
- Отступления от стандартов. Несмотря на наличие международного стандарта ANSI SQL-92, многие компании, занимающиеся разработкой СУБД (например, Oracle, Sybase, Microsoft, MySQL AB, Borland), вносят изменения в язык SQL, применяемый в разрабатываемой СУБД, тем самым отступая от стандарта. Таким образом, появляются специфичные для каждой конкретной СУБД диалекты языка SQL.

7.1.7. Базы данных на нефтеперерабатывающих заводах. Единая тематическая витрина данных на ООО «ПО "Киришинефтеоргсинтез"»

Одним из примеров эффективного использования возможностей современных СУБД на НПЗ служит т. н. «Единая тематическая витрина данных» на Киришском НПЗ.

До 1999 г. две подсистемы автоматизации предприятия – АСУП и АСУТП развивались обособленно и независимо друг от друга. Это выразилось в первую очередь в различии протоколов обмена данными, принятых в этих подсистемах.

Каждая из этих подсистем накладывает свои требования на средства сбора и хранения информации. АСУТП – большую скорость записи и большое время хранения данных, а АСУП – открытый интерфейс для доступа к ним из различных пользовательских приложений.

Создание информационной сети АСУТП верхнего уровня решило проблемы сбора, архивирования, структурирования информации и предоставления тем пользователям, чьи решения должны основываться на ее базе. Сеть верхнего уровня объединяет в единое информационное пространство большое количество распределенных систем управления и информационных систем различных производителей. Нижний уровень данной системы представлен коммуникационными серверами, выполняющими функции разделения управляющих и информационных сетей и передачи технологической информации на следующий уровень.

В зоне информационной сети, охватывающей все предприятие, находится сервер сбора технологической информации, позволяющий хранить большие массивы данных о технологическом процессе.

Для обобщения информации, поступающей из различных источников, был разработан программный комплекс «Единая тематическая витрина данных».

Потенциальными источниками информации для витрины могут выступать любые базы данных, существующие на предприятии. Это позволит анализировать информацию в различных срезах. Например, определить реальную зависимость потребления электроэнергии от загрузки технологического объекта, выявить причины брака.

Идея этого программного комплекса состоит в предоставлении пользователю удобного графического интерфейса к данным, характеризующим тот или иной технологический процесс, данным лабораторного контроля его характеризующим и их совокупное представление.

Сдерживающим фактором подключения новых объектов в первую очередь выступает устаревшая техника. Объекты, оснащенные совре-

менными РСУ, сразу попадают в списки объектов охваченных системой. Остальные подключаются по мере их оснащения более современной техникой.

Сегодня ЕТВД – это программный продукт, предназначенный для автоматизированного мониторинга технологических и производственных процессов.

ЕТВД реализована в виде клиент-серверного приложения с возможностью распределения по локальной сети. В качестве серверной части используется СУБД MS SQL Server, клиентскими приложениями являются клиент ЕТВД (*Etvdclient.exe*).

К целевой группе пользователей Витрины относятся должностные лица, отвечающие за управление производством, и разработчики математических моделей технологических объектов.

Основные понятия, используемые при работе с ЕТВД: запрос – обращение к БД за получением первичных данных, удовлетворяющих параметрам запроса; выборка – сохранённые данные, полученные при запросе, а также сам запрос.

К основным задачам, которые решает Витрина можно отнести следующие:

- предоставление первичных данных с производственных серверов (мониторинг);
- объединение представления информации с нескольких серверов;
- сохранение первичных данных в Витрине;
- представление вторичных данных;
- аналитическая обработка вторичных данных;
- визуализация первичных и вторичных данных;
- экспорт данных в Microsoft Office Excel.

Существующие СУБД повсеместно используются в областях промышленности, где ведется работа с большим объемом данных, в том числе и на крупных современных предприятиях. ЕТВД как инструмент инженера, используемый им для анализа текущего состояния процесса, зарекомендовал себя неотъемлемой частью рабочего пространства сотрудника завода. Именно поэтому любые прикладные программы, получающие возможность взаимодействия с ЕТВД, значительно расширяют свою функциональную пригодность и могут быть использованы для оперативного решения специфических вопросов, постоянно возникающих у начальников установок и другого инженерно-технического персонала.

7.1.8. Базы знаний

База знаний – результат обработки и систематизации информации о процессе в виде опыта технического персонала промышленной установки, сведений о неполадках и аварийных ситуациях, возникающих в процессе эксплуатации, а также данные технологического регламента производства.

Наличие баз знаний необходимо для оперативного проведения диагностики отклонений в работе промышленных установок, выбора оптимального варианта технологической схемы переработки определенного сырья и нахождения оптимальных режимных параметров. Гибкая структура базы знаний должна быть функционально увязана с задачами, решаемыми технологами. Для использования знаний необходима формализация и организация всех знаний в некоторую систему для того, чтобы этими знаниями можно было воспользоваться. Выбор модели из числа возможных полностью определяется спецификой конкретной задачи.

Рассмотрим возможные варианты моделей.

Логическая модель

Данная модель отображает знания в предметной области в виде совокупности простых фактов, а также утверждений и суждений. Факты считаются базовыми элементами логической модели. Утверждения и суждения составляются в виде формул. Формула записывается на основе базовых элементов и специальных синтаксических и семантических правил.

Любая логическая модель задается совокупностью множеств:

$$L =< T, S, A, M >;$$

где L – логическая модель, T – множество базовых элементов, S – множество семантических и синтаксических правил, которые позволяют строить из базовых элементов Т правильные синтаксические выражения или формулы, A – множество аксиом или утверждений, не подлежащие опровержению, M – множество семантических правил или правил вывода, позволяющих расширить список аксиом новыми правилами и формулами, которые после этого также становятся неопровержимыми истинами или аксиомами.

Высказывания могут быть истинными и ложными. Например, такое высказывание, как «колонна ректификации разделяет жидкости, имеющие различную летучесть» всегда является истиной для азеотропных смесей.

Каждое простое высказывание в логической модели можно записать в символической форме. Например, А=’насос увеличивает давление в системе’ или В=’теплообменники используются для изменения температуры потоков’.

В данном примере А и В являются компонентами логической модели и представляют собой истинные высказывания.

Основным недостатком логических моделей является их не структурированность. Это приводит к тому, что для сбора всей информации по одному объекту приходится прибегать к множеству логических формул.

Поскольку в основе понимания ХТС лежит структура логической модели, неудобная для оптимизации и прогнозирования ХТС, поэтому получили распространение **фреймовые модели**, которые в отличие от логических, являются структурированными.

Фреймовые модели

Такие модели используются в том случае, когда имеет место описание некоторой ситуации. Они также удобны для обозначения структуры и характеристики однотипных объектов ХП.

Под **объектами** понимают определенные химико-технологические процессы, протекающие на производстве, а также события и ситуации.

Фреймовая модель представляет собой некоторую оболочку для описания некоторого класса событий. Основу таких моделей составляют хорошо структурированные знания. Совокупность фреймов (фрейм – это смысловое описание ситуации), которые описывают некоторую предметную область, представляет собой иерархическую, в которой элементами фрейма является **слот**. В слоты объединяются в пределах одного фрейма по смысловым признакам. На верхнем уровне иерархической структуры находится фрейм, который содержит наибольшую информацию, истинную для естественных фреймов, которые находятся на низшем уровне иерархии. При этом различные вещества отличаются друг от друга, но для каждого из веществ можно указать набор характеристик, которые его определяют. К таким характеристикам относятся: химическая формула, агрегатное состояние, цвет и другое.

В этом случае фреймом является понятие вещества, а слотами, которые входят в этот фрейм, являются характеристиками, определяющими это вещество.

Семантическая модель

Данные модели для представления знаний имеют сетевую структуру, поэтому они используют понятие **дерево решений**.

Ствол дерева – это проблема, которую нужно решить. А возможные варианты решения распределяются по веточкам дерева. При этом событие или предметная область является следствием другого

события. Любая семантическая модель – это определенная последовательность явлений в цепочке: причина → следствие.

Например, [промышленные стоки] → [загрязнение сточных вод] → [загрязнение питьевой воды].

Семантические модели наряду с фреймовыми используются для логического вывода решения в виде единой цепочки рассуждений.

Фреймовые модели, по сравнению с семантическими, дают возможность подробного описания ситуаций в виде слотов. Поэтому в ХТ они используются намного чаще, чем другие модели.

Продукционные модели

Эти модели представляют собой перечень определенных правил. Например, если <условие>, то <заключение>.

По сути, продукционные модели очень схожи с фреймовыми моделями. С помощью фреймовых моделей можно описать достаточно сложные ситуации.

В продукционной модели в общем виде левая часть представляет всегда правило, а правая часть – действие.

Продукционные модели часто используют в системах управления. Компьютерные программы, построенные на основе продукционной модели, позволяют осуществить диагностику отклонений и выбрать определенную последовательность действий. Условия между собой в продукционной модели связываются определенными логическими знаками.

Например, если <концентрация вещества А <30%> или <концентрация вещества В>10%>, то <режим работы установки в интервале температур $T_1<T<T_2$>.

В данном примере в зависимости от состава переработанного сырья планируется температурный интервал в работе установки. Численные значения температуры Т можно определить только на математической модели. То есть, сочетание математической и продукционной модели позволяет определить интервал изменения температуры в определенный момент времени в зависимости от состава сырья.

Таким образом, выбор соответствующей модели для представления знаний определяется пользователем и конкретной задачей при разработке какой-либо модели.

Набор правил, фактов, заключений формируется на основе опытов эксплуатации промышленных установок эмпирическим путем. Эти правила должны быть тщательно проверены и подтверждены на работе нескольких

промышленных установок, которые функционируют при различающихся технологических условиях и перерабатывают различное сырье.

ИНТЕЛЛЕКТУАЛЬНЫЕ СИСТЕМЫ В ХТ

Интеллектуальная система – это технически организованная человеко-машинная система, представляющая собой совокупность базы данных и базы знаний о некоторой предметной области, а также пакета прикладных программ, обеспечивающих функционирование интеллектуальной системы на основе сбора, хранения и обработки различной информации.

С понятием интеллектуальной системы связаны понятия предметной области и проблемной среды.

Предметной областью будем называть совокупность предметов, которыми являются некоторые объекты, включающие химические производства, лабораторные и исследовательские комплексы, информация о которых накапливается в интеллектуальной системе.

Проблемной средой называется совокупность задач, которые решаются с использованием интеллектуальной системы.

Например, в предметную область входят конкретные химические производства, которые являются объектом для изучения.

Сбор информации в интеллектуальной системе проводится по этому конкретному объекту.

Проблемную среду составляют технико-экономические показатели выбранного производства, совокупность критериям, по которым оценивается эффективность функционирования выбранного производства.

Решение задач с использованием интеллектуальной системы требует привлечения математических моделей и численных методов для расчета и оптимизации производств.

Кроме математических моделей, используются также модели, для представления знаний, которые называются **лингвистическими или символьными.**

Интеллектуальная система позволяет выполнять решение задач с привлечением пользователя, так как интеллектуальная система содержит пояснения, сообщения, объяснения на доступном для всех языке. Это позволяет пользователям, которые являются, как правило, инженерно-техническими работниками предприятия, проводить адаптацию и усовершенствование интеллектуальной системы.

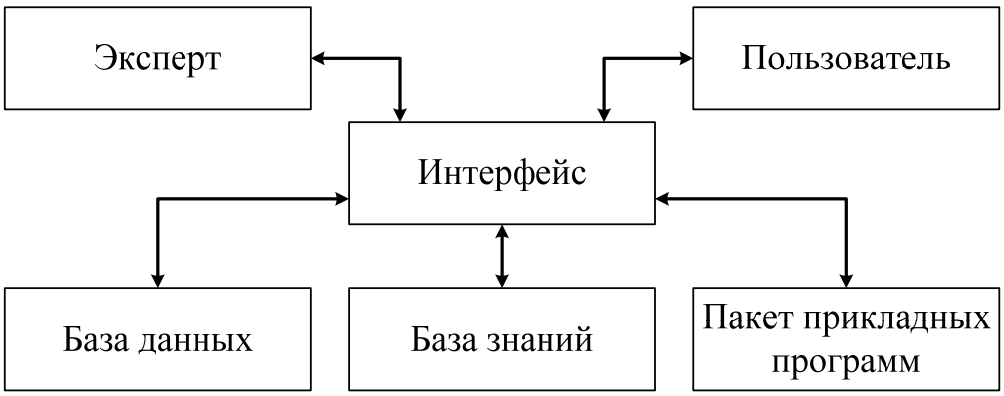


Рис. 7.6. Основные блоки интеллектуальной системы

Интеллектуальная система в общем случае состоит из следующих основных блоков (рис. 7.6):

- База данных, где содержится информация или численные значения характеристик ХТС, которая положена в основу данного химического производства. База данных содержит кинетические, термодинамические, теплофизические параметры процессов, которые протекают в ХТС.
- База знаний, включающая информацию о функционировании данного химического производства в виде правил, заключений, рекомендаций. Эта информация используется как входные данные в модели представления знаний.
- Пакет прикладных программ. Этот блок включает совокупность процедур для расчета ХТС. Сюда входят компьютерные программы, которые позволяют рассчитать типовые процессы, протекающие в ХТС, такие как процессы химического превращения, теплообмена, смешения, разделения и другие.
- Организующая программа. Этот блок представляет собой объект принятия решения. Сюда поступает информация от разработчиков и от пользователей.

Пользователи интеллектуальной системы выступают в качестве экспертов и помогают пополнить информацию предприятия относительно изучаемого предприятия и тем самым расширить круг задач, которые система решает. Данная структура интеллектуальной системы является обобщенной. Интеллектуальные системы отличаются одна от другой предметной областью и проблемной средой, а также способами представления данных и знаний, которые определяются методами построения математических и символьных моделей.

На рис. 7.7 представлена схема взаимодействия с ЕТВД.

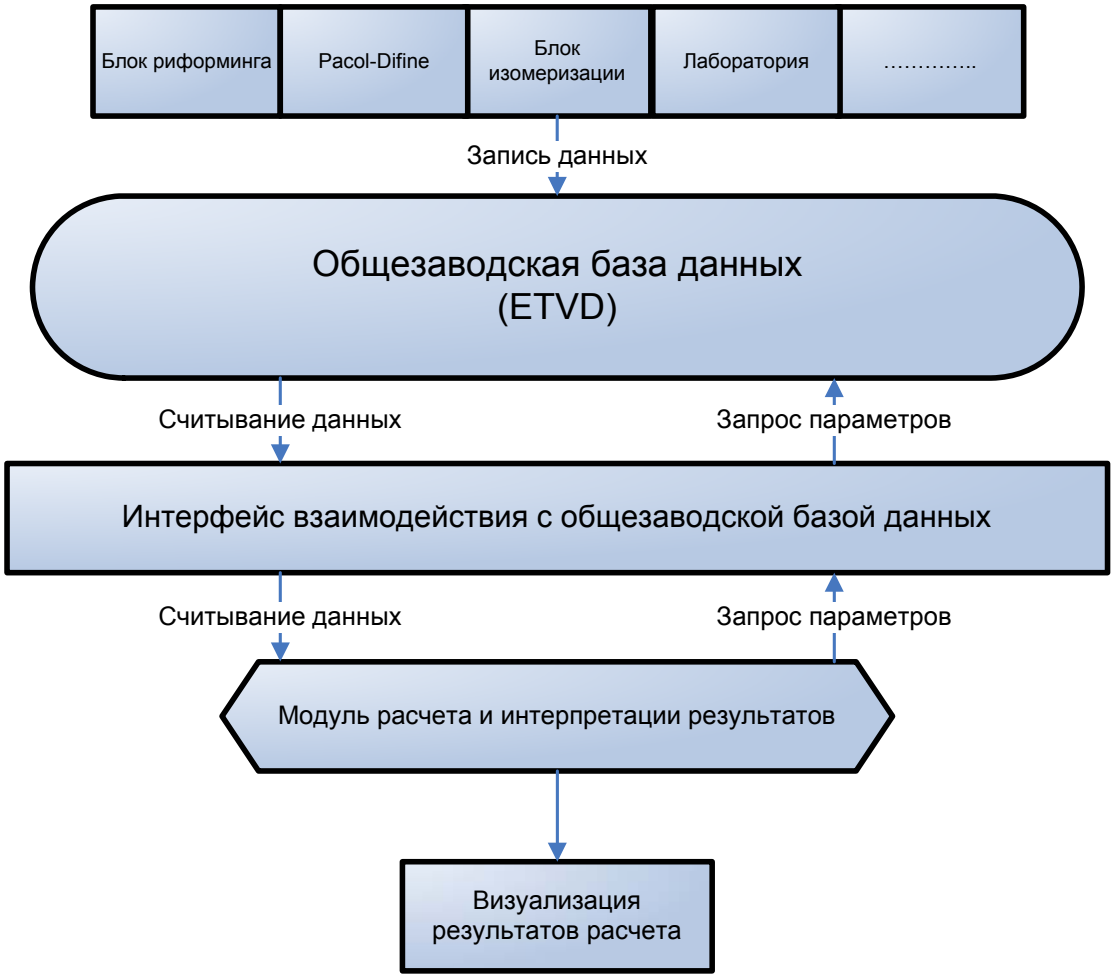


Рис. 7.7. Схема взаимодействия общезаводской базы данных с базой знаний

На общезаводскую базу данных (ЕТВД) поступают и накапливаются данные с различных установок и цехов завода. Интеллектуальная система считывает из ЕТВД информацию, проводит ее обработку, на основе заложенной базы знаний, выдает результат, например, рекомендации. Такая система взаимодействия позволяет оперативно корректировать технологические параметры для повышения эффективности процесса, а также – прогнозировать работу установки.

7.2. Сеть интернет

7.2.1. Что такое Интернет

Определения, которые давались сети Интернет: Глобальная компьютерная сеть, Сеть сетей, Всемирная Сеть.

Интернет представляет собой **распределенную децентрализованную систему**, т. е. в нем нет центральных/главных узлов. Правила его функционирования **стандартизованы и общедоступны**.

С технической точки зрения, Интернет состоит из большого числа менее крупных сетей, которые также неоднородны.

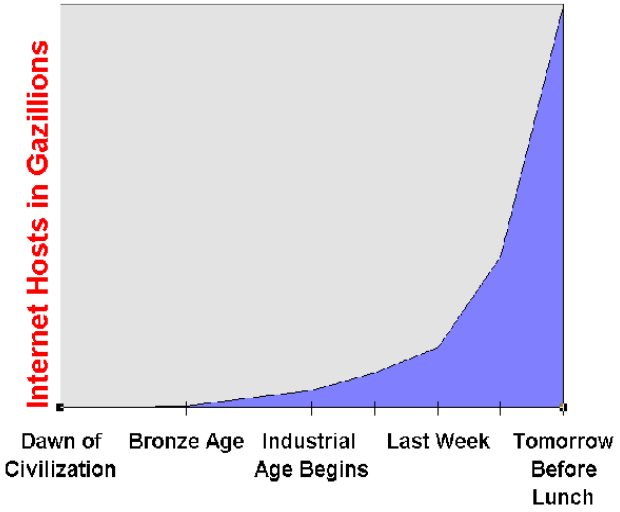
Формально Интернет никому в отдельности не принадлежит.

Оплата расходов по поддержанию в рабочем состоянии старых каналов ложится на конечных пользователей.

Расходы по прокладке новых каналов – правительство, крупные телекоммуникационные компании, инвестиционные фонды.

Пользователи Интернет изобретают все более эффективные способы для создания, хранения и получения информации.

Больше пользователей в Сети – тем большее количество информации они производят.



Интернет является на сегодняшний день основной информационной средой, местом циркуляции информации и ее огромным всемирным хранилищем.

Так рост Интернет видится журналистам «New-York Times».

В новом тысячелетии информация будет играть все более важную роль в жизни человека, доступ к информационным потокам и интеграция в них станут неотъемлемой частью нашей жизни.

Сегодня Вы становитесь пользователем Интернет, и **теперь Вы будете готовы к приходу 2000-го года**.

7.2.2. История возникновения Интернет

1957 г.: Д. Эйзенхауер, **DARPA** (Defense Advanced Research Projects Agency), ответ на запуск Советами первого искусственного спутника Земли. Dr. J.C.R. Licklider.

60-е годы: идеи, которые легли в основу Интернет. Проекты предусматривали разработку стандарта для децентрализованной сети, которая смогла бы продолжать функционировать в случае отказа части ее узлов в случае ядерного удара со стороны СССР.

70-е годы: построение такой сети – **ARPANET**.

В ARPANET сеть использовалась главным образом для передачи файлов и электронных сообщений.

Начало 80-х: стали бурно развиваться альтернативные сети (передача электронной почты): **BitNet и CSNet**. Позже к ним прибавились и другие (FidoNet etc).

86 год: **NSF** (National Science Foundation) начал работы по созданию **NSFNET**, сети магистральных каналов для американского Интернет'а. Создание **ISOC** (Internet Society) – добровольной организации по обеспечению глобального обмена информацией посредством Internet, Internet Architecture Board (**IAB**) – сертификация стандартов и распределения ресурсов, Internet Engineering Task Force (**IETF**) – эксплуатационные и назревающие технические проблемы. Эти организации успешно действуют по сей день.

В 90-е годы появляется Интернет как международная глобальная сеть. В 90-е г. В Интернете – 300000 пользователей. В 1991 году происходит изобретение **gopher**'а и **World Wide Web**, служб Интернет, которые привлекут миллионы новых пользователей.

Сегодня Интернет не охватывает только несколько стран Африки и Восточной Азии. Количество машин в Интернет – около 20 млн, количество пользователей – в 5–10 раз больше.

7.2.3. Интернет – компьютерные сети и сетевые протоколы. Семейство протоколов TCP/IP

Когда два или более компьютеров связывают постоянным соединением для обмена данными, говорят о создании **компьютерной сети**.

Сетевой протокол определяет правила передачи информации в компьютерной Сети. Уместно сравнение сетевого протокола с языком – это такая же оболочка для передачи информации.

Различают **открытые и закрытые** сетевые протоколы, в зависимости от того, является ли открытым определяющий их стандарт.

Можно говорить также о протоколах различных уровней, в частности, о протоколах **уровня приложений и протоколах уровня передачи данных**.

Операционная система – основная программа, работающая на компьютере. Запускается раньше всех других программ и делает возможным их запуск. Пример: MS DOS, Windows 95, UNIX, Linux, Novell Netware.

UNIX – операционная система и семейство мощных современных многопользовательских операционных систем, на которых построен Интернет.

RFC – Request For Comments, стандарты Интернет, публикуемые IAB.

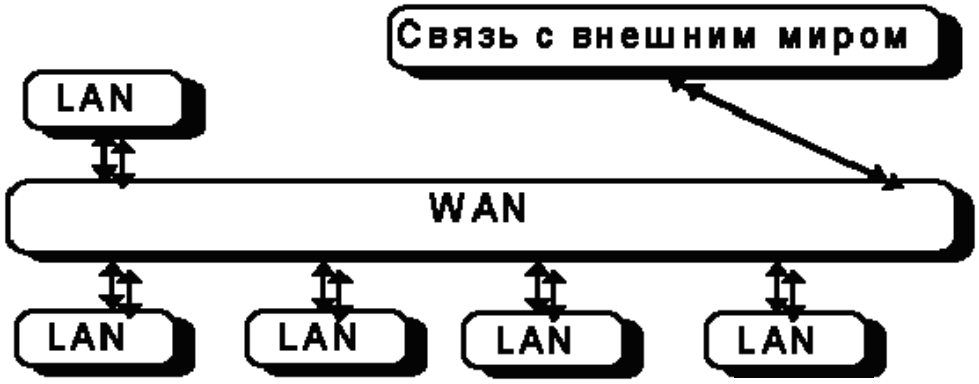
Провайдер Интернет – поставщик услуг Интернет, тот, через кого Вы или Ваша организация подключаетесь к Сети.

Модем (**мод**улятор-**дем**одулятор) – устройство для соединения удаленных компьютеров. Работает по выделенным и телефонным (ком-

мутируемым) линиям. Используется, среди прочего, для подключения к компьютерной сети домашних пользователей.

LAN (local area network) – локальная компьютерная сеть, соединяющая компьютеры непосредственно, обычно с помощью кабелей.

WAN (wide area network) – сеть, соединяющая локальные сети (LAN).



Режим работы **on-line/off-line** – с подключением к Интернет/без него.

PPP – Point-to-Point Protocol – используются наряду с прочими при подключении хоста или сети к Интернет, в частности – при подключении к Интернет домашних компьютеров пользователей в режиме on-line.

SLIP – Serial Line IP (Internet Protocol) – менее совершенный аналог PPP.

UUCP – Unix-to-Unix Copy Protocol – устаревший протокол передачи файлов и почтовых сообщений.

Броузер (browser) – программа для доступа к самым популярным службам Интернет. Самые известные броузеры – Netscape Navigator (http://www.netscape.com) и Microsoft Internet Explorer (http://www.microsoft.com/ie).

Единицы измерения информации и скорости ее передачи

Бит – минимальное количество информации – «да» или «нет», «0» или «1».
Байт – 8 бит для описания буквы или символа «00000110» = «6»
Бит/с(bps) или бод (baud) – импульс в секунду. (Кбит/с); (Мбит/с).
Приставки: кило – 10^3, мега – 10^6, тера – 10^9.

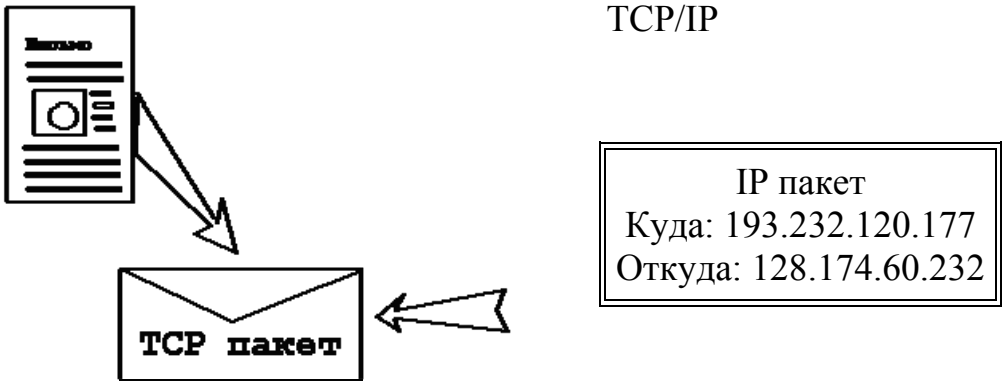
Семейство протоколов TCP/IP

TCP/IP (Transmission Control Protocol/Internet Protocol – протокол контроля за передачей данных/протокол передачи данных) – семейство протоколов, на которых построен Интернет, от второго из них он получил свое имя. В TCP/IP входит около 100 протоколов различного уровня.

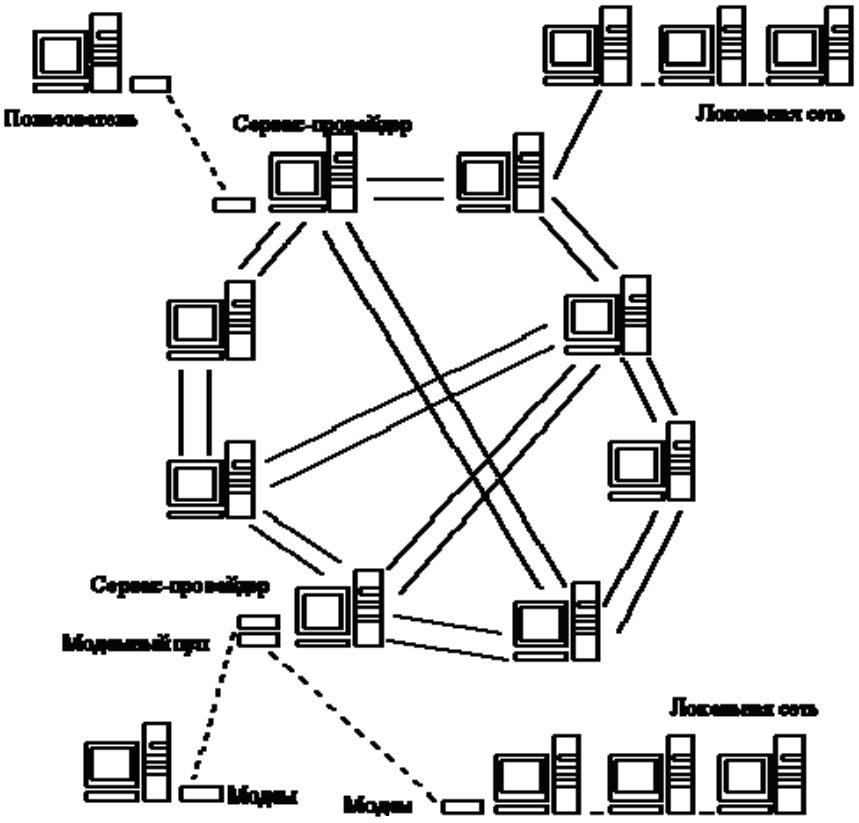
IP-Адрес (IP-address) – уникальный идентификатор компьютера в Интернет. Каждый компьютер, работающий по TCP/IP имеет его. IP-Адрес представляет собой четыре групы по восемь двоичных чисел, и часто записывется в десятеричной форме, например 193.232.121.66

Маска подсети (Subnet mask) – параметр конфигурации TCP/IP, задаваемый администратором сети. Имеет такой же вид, как и IP-Адрес, например 255.255.255.0

Шлюз (Gateway) – компьютер, соединяющий локальную сеть (LAN) с провайдером (WAN).



Общий вид Интернет:

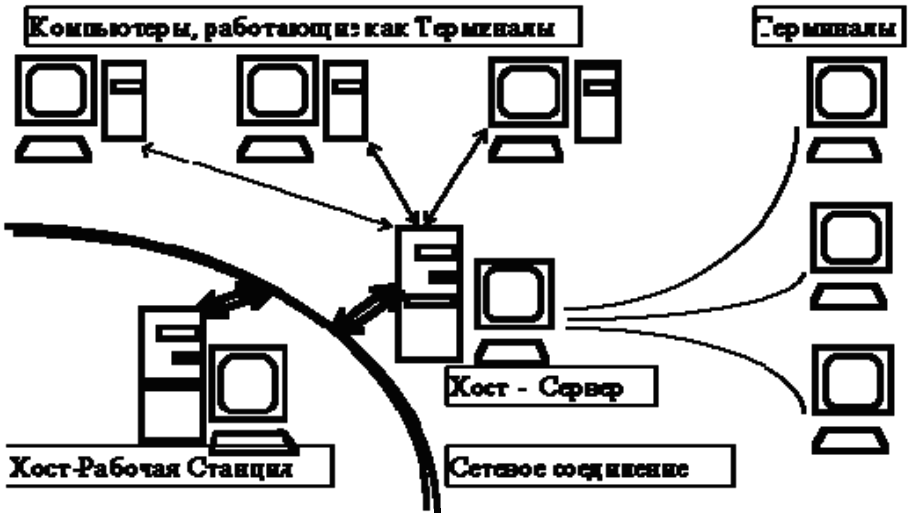


7.2.4. Службы (сервисы) Интернет и архитектура клиент-сервер

Интернет вызывает интерес пользователей возможностями, заложенными в **службах (сервисах)** Интернет.

Службы Интернет используют архитектуру **клиент-сервер**, т. е. если Вы – пользователь, клиентская программа на Вашем компьютере должна подключиться к серверу и послать ему запрос на получение информации. Программа на сервере, в свою очередь, вышлет эту информацию и будет ожидать следующего запроса (это, конечно, некоторое упрощение).

Серверы и Клиенты



Сервер (server), предоставляет определенные ресурсы. Другая программа – **клиент** (client) – эти ресурсы использует.

Некоторые сервисы Интернет:

Служба (сервис)	Протокол передачи данных	Программа доступа
WWW (World Wide Web) Всемирная Паутина, гипермедиаданные	HTTP – HyperText Transfer Protocol	Броузер
FTP (File Transfer Protocol) Доступ к файловым архивам, анонимный/авторизованный	FTP	Броузер, специализированные программы
Gopher – текстовые данные, иерархически организованные	Gopher	Броузер
E-mail (electronic mail) электронная почта	SMTP (Simple Mail Transfer Protocol)/POP3(Post Office Protocol) – прием/передача почты	Встроенная в броузер программа, специализированные программы

FQDN (Fully Qualified Domain Name) – Имя хоста, имя домена, например: машина tom в домене foobar.com имеет FQDN **tom.foobar.com**

Другие примеры: **fadr.msu.ru, www.ford.com**

URL – Universal Resource Locator – общая форма представления адреса ресурса в Интернет

URL формируется следующим образом:

<имя протокола>://<FQDN или адрес хоста>{/<путь к ресурсу>}

Примеры:

- ftp://ftp.funet.fi/
- http://www.microsoft.com/ie,
- nntp://fido7.pvt.exch.cars
- http://193.232.127.161/~cstore/index.html

7.2.5. Интернет в цифрах

«Ежемесячно по каналам Internet перемещается более 30 Тбит информации (по скромным подсчетам – это около 30 млн книг по 700 страниц каждая) между приблизительно 50 млн пользователей» – INTERNET И ЭКОНОМИКА, вып. № 14, 6–12 апреля 1998 г.

Машин (хостов) в Интернет: 36 миллионов.

Доменных имен: порядка 1.500.000.

Серверов Всемирной Паутины: 2.215.195. Их количество удвоилось за последние 8 месяцев (вспомните карикатуру из Нью-Йорк Таймс).

Пользователей Интернет: можно только догадываться – приблизительно между 1.000.000 до 6.000.000.000 – хотя, может быть, и больше (one never knows if you're a dog on the Net). Порядок цифры – сотни миллионов.

8. СРЕДСТВА ЗАЩИТЫ ИНФОРМАЦИИ

Требования к безопасности должны быть определены и согласованы до разработки информационных систем. Средства защиты оказываются значительно более дешевыми и эффективными, если их встроить в прикладные системы на стадиях задания требований и проектирования. Все требования к безопасности, включая необходимость перехода на аварийный режим для продолжения обработки информации, следует определить на стадии задания требований к проекту, а также обосновать, согласовать и задокументировать их в рамках общего плана работ по созданию информационной системы.

8.1. Анализ и задание требований к безопасности

Анализ требований к безопасности следует проводить на стадии анализа требований к каждому проекту разработки систем. При формулировании производственных требований к новым системам или модернизации существующих систем, необходимо задать требования к средствам управления безопасностью. Такие требования обычно сосредоточены на автоматических средствах контроля, встраиваемых в системы, однако следует также рассмотреть необходимость использования вспомогательных ручных средств управления безопасностью. Эти соображения следует также принять во внимание при качественной оценке пакетов программ для производственных приложений.

Требования к безопасности и средства управления ею должны отражать ценность информационных ресурсов для организации, а также возможные последствия от нарушения режима безопасности или отсутствия средств защиты для производственных процессов.

Основу анализа требований к безопасности составляют:

- рассмотрение необходимости обеспечения конфиденциальности, целостности и доступности информационных ресурсов;
- определение возможностей использования различных средств контроля для предотвращения и выявления случаев нарушения защиты, а также восстановления работоспособности систем после их выхода из строя и инцидентов в системе безопасности.

В частности, при проведении такого анализа следует рассмотреть необходимость:

а) управления доступом к информации и сервисам, включая требования к разделению обязянностей и ресурсов;

б) регистрации значительных событий в контрольном журнале для целей повседневного контроля или специальных расследований, в том числе как свидетельство при проведении переговоров с подрядчиками и другими лицами;

в) проверки и обеспечения целостности жизненно важных данных на всех или избранных стадиях их обработки;

г) защиты конфиденциальных данных от несанкционированного раскрытия, в том числе возможное использование средств шифрования данных в специальных случаях;

д) выполнения требований инструкций и действующего законодательства, а также договорных требований, в том числе составление специальных отчетов для удовлетворения определенных правовых требований;

е) снятия резервных копий с критически важных производственных данных;

ж) восстановления систем после их отказов, особенно для систем с повышенными требованиями к доступности;

з) защиты систем от внесения несанкционированных дополнений и изменений;

и) предоставления возможности безопасного управления системами и их использования сотрудникам, не являющимся специалистами (но имеющих надлежащую подготовку);

к) обеспечения соответствия систем требованиям аудиторов, например, посредством использования таких средств, как встроенные программы-утилиты для выборочного контроля и независимое программное обеспечение для повторения критически важных вычислений.

Средства управления безопасностью, встроенные в компьютерные системы, могут быть скомпрометированы, если обслуживающий их персонал и пользователи не будут их знать. Поэтому необходимо явно определить эти средства контроля в соответствующей документации.

8.2. Безопасность в прикладных системах

При проектировании прикладных систем необходимо встроить в них надлежащие средства управления безопасностью, в том числе средства регистрации событий в контрольном журнале.

Проектирование и эксплуатация систем должны соответствовать общепринятым промышленным стандартам обеспечения надежной защиты, определенным в настоящих практических правилах.

Системы, которые поддерживают или оказывают влияние на исключительно уязвимые, ценные или критически важные информационные ресурсы организации, могут потребовать принятия дополнительных мер противодействия. Такие меры следует определить исходя из рекомендаций специалиста по безопасности с учетом идентифицированных угроз нарушения защиты и возможных последствий от их реализации для организации.

8.3. Проверка достоверности входных данных

Чтобы обеспечить правильный ввод данных в прикладные системы, необходимо проверять их на достоверность. Предлагаются следующие средства контроля:

а) проверки с целью выявления следующих ошибок:

- величины, выходящие за заданные пределы;
- неправильные символы в полях данных;
- пропущенные или неполные данные;
- превышенные верхние и нижние пределы на объем вводимых данных;
- несанкционированные или противоречивые управляющие данные;

б) периодический анализ содержания ключевых полей или файлов данных для подтверждения их достоверности и целостности;

в) осмотр печатной входной документации на предмет внесения несанкционированных изменений во входные данные (необходимо получить разрешение на внесение всех изменений во входные документы);

г) процедуры реагирования на ошибки, связанные с проверкой достоверности входных данных;

д) определение обязанностей всех сотрудников, участвующих в процессе ввода данных.

8.4. Проверка достоверности внутренней обработки данных

Данные, которые были правильно введены в прикладную систему, могут быть повреждены в результате ошибок обработки или преднамеренных действий. Чтобы выявить такие случаи повреждения данных, необходимо встроить средства проверки в системы. Требуемые для этого средства контроля определяются характером приложения и последствиями от повреждения данных для организации.

Примерами средств проверки, которые можно встроить в системы, являются:

а) контроль сеанса связи и пакетной обработки для согласования файлов данных о платежном балансе после проведения операций с ними;

б) контроль платежного баланса для сверки начального сальдо с предыдущим конечным сальдо:

- контроль за выполнением операций;
- подведение итогов по обновлению файлов;
- контроль за выполнением программ;

в) проверка достоверности данных, сгенерированных системой (см. Проверка достоверности входных данных);

г) проверка целостности данных и программ, пересылаемых между центральным и удаленными компьютерами (см. Аутентификация сообщений);

д) подведение итогов по обновлению файлов.

8.5. Аутентификация сообщений

Аутентификация сообщений – это метод, используемый для выявления несанкционированных изменений, внесенных в передаваемые электронные сообщения, или их повреждения. Его можно реализовать на аппаратном или программном уровне с помощью физического устройства аутентификации сообщений или программного алгоритма.

Возможность аутентификации сообщений следует рассмотреть для тех приложений, для которых жизненно важным является обеспечение целостности сообщений, например, электронные передачи информации о денежных средствах или другие электронные обмены данными. Для определения необходимости аутентификации сообщений и выбора наиболее подходящего метода ее реализации необходимо провести оценку риска нарушения режима безопасности.

Аутентификация сообщений не предназначена для защиты содержания сообщений от перехвата. Для этих целей подходит шифрование данных, которое можно также использовать для аутентификации сообщений. (Электронная подпись – специальный вид аутентификации сообщений, обычно основанный на методах шифрования с открытым ключом, который обеспечивает аутентификацию отправителя, а также гарантирует целостность содержимого сообщения.)

8.6. Защита файлов прикладных систем

Цель: Обеспечить надежную реализацию проектов разработки информационных систем и их поддержку.

Доступ к системным файлам необходимо контролировать.

Поддержание целостности прикладных систем должно быть обязанностью пользователя или группы разработки, которой прикладная система или программное обеспечение принадлежит.

8.7. Контроль рабочего программного обеспечения

Следует осуществлять жесткий контроль за реализацией программного обеспечения в рабочих системах. Чтобы свести риск повреждения рабочих систем к минимуму, необходимо реализовать следующие средства контроля:

а) обновление рабочих библиотек программ должен осуществлять только назначенный библиотекарь после получения санкции на доступ к приложению от руководителя персонала, обслуживающего информационные системы (см. Управление доступом к библиотекам исходных текстов программ);

б) в рабочих системах следует хранить только выполняемые программы (по возможности);

в) выполняемые программы не следует запускать на рабочих системах до тех пор, пока они не пройдут тестирование и не будут приняты пользователями, а соответствующие библиотеки исходных текстов программ не будут обновлены;

г) необходимо фиксировать все случаи обновления рабочих библиотек программ в контрольном журнале;

д) предыдущие версии программ следует сохранить – мера предосторожности при чрезвычайных ситуациях.

8.8. Защита системных тестовых данных

Тестовые данные необходимо защищать и контролировать. Тестирование систем и их приемка обычно требуют значительные объемы тестовых данных, которые близки к реальным данным настолько, насколько это возможно. Необходимо избегать использовния реальных баз данных, содержащих персональные данные. Прежде чем использовать такие данные, их необходимо обезличить. Для защиты реальных данных при их использовании для целей тестирования, предлагаются следующие средства контроля:

а) процедуры управления доступом, которые применяются для рабочих прикладных систем, должны также применяться для тестируемых прикладных систем;

б) необходимо получить отдельное разрешение всякий раз, когда реальные данные копируются в тестируемую прикладную систему;

в) реальные данные следует удалить из тестируемой прикладной системы сразу после завершения процесса тестирования;

г) случаи копирования реальных данных необходимо регистрировать в контрольном журнале.

8.9. Безопасность в среде разработки и рабочей среде

Среду разработки и рабочую среду необходимо жестко контролировать. Администраторы, отвечающие за прикладные системы, должны также отвечать за защиту среды разработки и рабочей среды. Они должны анализировать все изменения, которые предлагается внести в системы, чтобы гарантировать, что они не нарушат безопасность системы или рабочей среды.

8.10. Процедуры управления процессом внесения изменений

Чтобы свести риск повреждения информационных систем к минимуму, следует осуществлять жесткий контроль за внесением изменений в них. Для этого требуются формальные процедуры управления процессом внесения изменений. Эти процедуры должны гарантировать, что безопасность и процедуры управления ею не будут скомпрометированы, что программистам, отвечающих за поддержку систем, предоставлен доступ только к тем компонентам системы, которые необходимы для их работы, и что получено формальное разрешение на внесение изменений. Такой процесс должен включать в себя следующее:

а) регистрацию согласованных уровней полномочий, в том числе:

- служба приема запросов на внесение изменений группой, обслуживающей информационные системы;
- полномочия пользователей на подачу запросов на внесение изменений;
- уровни полномочий пользователей на принятие подробных предложений;
- полномочия пользователей на принятие вносимых изменений;

б) принятие изменений, предлагаемых только зарегистрированными пользователями;

в) проверку средств управления безопасностью и процедур обеспечения целостности на предмет их компрометации внесенными изменениями;

г) выявление всех компьютерных программ, файлов данных, баз данных и аппаратных средств, которые требуют внесения поправок;

д) утверждение подробных предложений до начала работы;

е) обеспечение принятия предлагаемых изменений зарегистрированными пользователями до их внесения;

ж) обновление системной документации по завершении процесса внесения каждого изменения, а также архивирование или уничтожение старой документации;

з) осуществление контроля над версиями всех обновляемых программ;

и) регистрацию всех запросов на внесение изменений в контрольном журнале.

8.11. Технический анализ изменений, вносимых в операционную систему

Необходимость во внесении изменений в операционную систему возникает периодически, например, инсталляция новой версии, предоставляемой поставщиком. В таких случаях следует проводить анализ прикладных систем на предмет возможного нарушения режима безопасности, проистекающий от таких изменений. Этот процесс должен включать в себя следующее:

а) проверка процедур контроля приложений и обеспечения их целостности на предмет компрометации вследствие внесения изменений в операционную систему;

б) обеспечить включение в ежегодный план поддержки проверку и тестирование систем, связанные с изменениями, вносимыми в операционную систему, а также выделить для этого необходимые финансовые средства;

в) обеспечить своевременное уведомление сотрудников о предлагаемых изменениях в операционной системе для проведения надлежащего анализа до их внесения.

8.12. Ограничения на внесение изменений в пакеты программ

Не рекомендуется вносить изменения в пакеты программ. По возможности следует использовать пакеты програм, предоставляемые поставщиками, без их модификации. В тех случаях, когда возникает необходимость во внесении изменений в пакеты программ, следует рассмотреть следующие пункты:

а) риск компрометации встроенных средств контроля и процессов обеспечения целостности;

б) необходимость получения согласия поставщика;

в) возможность получения требуемых изменений от поставщика в рамках стандартного обновления программ;

г) возможность взятия организацией ответственности за дальнейшее сопровождение программного обеспечения в результате внесенных изменений.

Если изменения считаются крайне необходимыми, то следует сохранить исходное программное обеспечение, а изменения внести в четко определенную копию. Эти изменения необходимо полностью задокументировать так, чтобы их можно было вносить в будущие обновленные версии программ в случае необходимости.

СПИСОК ЛИТЕРАТУРЫ

1. Кравцов А.В., Иванчина Э.Д. Интеллектуальные системы в химической технологии и инженерном образовании: Нефтехимические процессы на Pt-катализаторах / А.В. Кравцов, Э.Д. Иванчина. – Новосибирск: Наука. Сибирская издательская фирма РАН, 1996. – 200 с.
2. Фигурнов В.Э. IBM PC для пользователя. – 5-е изд., исправл. и доп. – Уфа, ПК «Дегтярев и сын», НПО «Информатика и компьютеры», 1993. – 352 с.
3. Фаронов В.В. Турбо-Паскаль: учебное пособие / В.В. Фаронов. – 7-е изд., перераб. – М.: Нолидж, 2001. – 575 с.
4. Моргун А.Н. Программирование на языке Паскаль. Основы обработки структур данных / А.Н. Моргун, И.А. Кривель. – М.: Вильямс, 2006. – 567 с.
5. Фаронов В.В. Программирование баз данных в Delphi 7 / В.В. Фаронов. – СПб.: Питер, 2004. – 459 с.
6. Фаненштих Клаус, Хаселир Райнер. Текстовый процессор Word 6.0 для Windows: практ. пособ/ пер. с нем. – 2-е изд., исправл. и доп.– М.: ЭКОМ., 1995. – 352 с.
7. Мир Windows XP // http://onlyxp.narod.ru
8. Пол Мак-Федрис. Microsoft Windows // http://ru.wikipedia.org
9. BorlanD Russian Community // http://bdrc.ru
10. Кириллов В.В. Структурированный язык запросов (SQL). – СПб.: ИТМО, 1994. – 80 с.
11. «SQL Полное руководство» BHV. – Киев, 1998.
12. «Эффективная работа с Microsoft Access 7.0». – СПб.: Питер, 1997.
13. Гершберг А.Ф., Безручко О.А. Автоматизация производства ООО «ПО "Киришинефтеоргсинтез"». Применение современных информационных технологий // Нефтепереработка и нефтехимия. – 2006. – № 2. – С. 45–49.
14. Единая тематическая витрина данных. Руководство оператора.
15. http://www.istocnikugroz.ru

ОГЛАВЛЕНИЕ

Учебное издание

КРАВЦОВ Анатолий Васильевич
ЧЕКАНЦЕВ Никита Витальевич
ШАРОВА Екатерина Сергеевна
ГЫНГАЗОВА Мария Сергеевна
СМЫШЛЯЕВА Юлия Александровна
ИВАНЧИНА Эмилия Дмитриевна

ПРОБЛЕМНО ОРИЕНТИРОВАННАЯ ИНФОРМАТИКА ХИМИКО-ТЕХНОЛОГИЧЕСКИХ ПРОЦЕССОВ

Учебное пособие

Научный редактор *доктор технических наук, профессор Э.Д. Иванчина*

Выпускающий редактор *Д.В. Заремба*
Редактор *Е.О. Фукалова*
Компьютерная верстка *Н.В. Чеканцев*
Дизайн обложки *А.И. Сидоренко*

Подписано к печати 07.04.2014. Формат 60x84/16. Бумага «Снегурочка».
Печать XEROX. Усл. печ. л. 9,31. Уч.-изд. л. 8,42.
Заказ 269-14. Тираж 100 экз.

Национальный исследовательский Томский политехнический университет
Система менеджмента качества
Издательства Томского политехнического университета
сертифицирована в соответствии с требованиями ISO 9001:2008

ИЗДАТЕЛЬСТВО ТПУ. 634050, г. Томск, пр. Ленина, 30
Тел./факс: 8(3822)56-35-35, www.tpu.ru