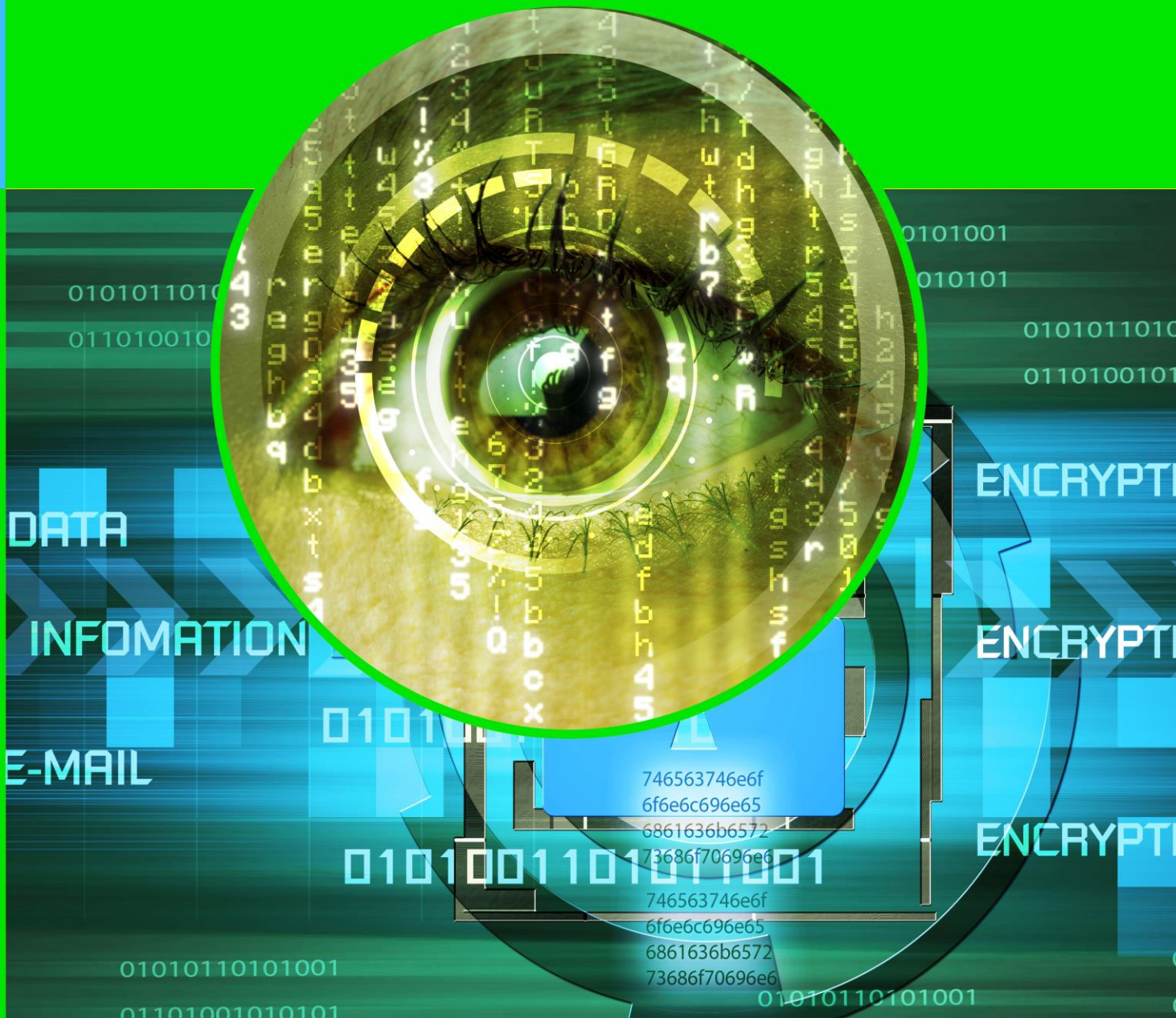
+16

Сеченовский Университет

С.В. Глушков
И.А. Иконникова
Н.Н. Пронькин
И.Ф. Семёнычева

АЛГОРИТМИЧЕСКИЕ ЯЗЫКИ

УЧЕБНОЕ ПОСОБИЕ



ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
ПЕРВЫЙ МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ МЕДИЦИНСКИЙ
УНИВЕРСИТЕТ ИМЕНИ И.М. СЕЧЕНОВА
МИНИСТЕРСТВА ЗДРАВООХРАНЕНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
(СЕЧЕНОВСКИЙ УНИВЕРСИТЕТ)

Глушков Сергей Владимирович
Иконникова Ирина Александровна
Пронькин Николай Николаевич
Семёнычева Ирина Флюровна

АЛГОРИТМИЧЕСКИЕ ЯЗЫКИ

Учебное пособие

Москва
2020

УДК 681.5
ББК 32.97
А45

Рецензент:

Герасимов А.Н., д.ф.-м.н., профессор, заведующий кафедры «Медицинская информатика и статистика» Первого Московского государственного медицинского университета имени И.М. Сеченова Министерства здравоохранения Российской Федерации (Сеченовский Университет)

Авторы:

Глушков С.В., Иконникова И.А., Пронькин Н.Н., Семёнычева И.Ф.

АЛГОРИТМИЧЕСКИЕ ЯЗЫКИ [Электронный ресурс]: учебное пособие – Эл. изд. - Электрон. текстовые дан. (1 файл pdf: 134 с.). - Глушков С.В., Иконникова И.А., Пронькин Н.Н., Семёнычева И.Ф. 2020. – Режим доступа: http://scipro.ru/conf/algorithmiclanguages.pdf. Сист. требования: Adobe Reader; экран 10'.

ISBN 978-5-6044576-2-7

Учебное пособие по изучению дисциплины «Алгоритмические языки» разработано в соответствии с требованиями государственного образователь-ного стандарта.

В пособии представлены методические материалы по курсу «Алгорит-мические языки». Изложены базовые сведения и освещен круг вопросов, свя-занных с применением современных технологий программирования при ре-шении профессиональных задач.

Материал излагается простым, доступным языком.

Учебное пособие рассмотрено и одобрено на заседании кафедры «Медицинская информатика и статистика» 21 января 2020 г., протокол №5.

ISBN 978-5-6044576-2-7

Содержание

Введение

Дисциплина «Алгоритмические языки» входит в федеральный компонент блока общепрофессиональных дисциплин для подготовки бакалавров по направлениям 09.03.02 «Информационные системы и технологии» и 01.03.03 «Механика и математическое моделирование».

Целью дисциплины является формирование теоретических основ и практических навыков применения современных технологий программирования в процессах медико-социального обслуживания населения. Изучение основных принципов проектирования и реализации информационных моделей, проведение прикладного анализа полученных результатов с их последующим применением в профессиональной деятельности врача, социального работника, специалиста санитарно-сестринского звена.

Задачи дисциплины – владеть: навыками по разработке и применению современных технологий программирования при решении профессиональных задач; принципами функционирования специализированных инструментальных программ, применяемых в современных вычислительных системах; техникой использования современного программного обеспечения для задач обработки биомедицинской информации.

ГЛАВА 1. ПОНЯТИЕ АЛГОРИТМА. БАЗОВЫЕ АЛГОРИТМИЧЕСКИЕ СТРУКТУРЫ

Понятие алгоритма такое же основополагающее, как и понятие информации. Именно поэтому важно в нем разобраться.

Название **"алгоритм"** произошло от латинской формы имени величайшего среднеазиатского математика **Мухаммеда ибн Муса ал-Хорезми** (Alhorithmi), жившего в 783-850 гг. В своей книге "Об индийском счете" он изложил правила записи натуральных чисел с помощью арабских цифр и правила действий над ними "столбиком", знакомые теперь каждому школьнику. В XII веке эта книга была переведена на латынь и получила широкое распространение в Европе.

Человек ежедневно встречается с необходимостью следовать тем или иным правилам, выполнять различные инструкции и указания. Например, переходя через дорогу на перекрестке без светофора надо сначала посмотреть направо. Если машин нет, то перейти полдороги, а если машины есть, ждать, пока они пройдут, затем перейти полдороги. После этого посмотреть налево и, если машин нет, то перейти дорогу до конца, а если машины есть, ждать, пока они пройдут, а затем перейти дорогу до конца.

Алгоритм – заранее заданное понятное и точное предписание возможному исполнителю совершить определенную последовательность действий для получения решения задачи за конечное число шагов.

Основные свойства алгоритмов следующие:

1. **Понятность** для исполнителя – исполнитель алгоритма должен понимать, как его выполнять. Иными словами, имея алгоритм и произвольный вариант исходных данных, исполнитель должен знать, как надо действовать для выполнения этого алгоритма.
2. **Дискретность** (прерывность, раздельность) – алгоритм должен представлять процесс решения задачи как последовательное выполнение простых (или ранее определенных) шагов (этапов).
3. **Определенность** – каждое правило алгоритма должно быть четким, однозначным и не оставлять места для произвола. Благодаря этому свойству выполнение алгоритма носит механический характер и не требует никаких дополнительных указаний или сведений о решаемой задаче.
4. **Результативность** (или конечность) состоит в том, что за конечное число шагов алгоритм либо должен приводить к решению задачи, либо после конечного

числа шагов останавливаться из-за невозможности получить решение с выдачей соответствующего сообщения, либо неограниченно продолжаться в течение времени, отведенного для исполнения алгоритма, с выдачей промежуточных результатов.

5. **Массовость** означает, что алгоритм решения задачи разрабатывается в общем виде, т.е. он должен быть применим для некоторого класса задач, различающихся лишь исходными данными. При этом исходные данные могут выбираться из некоторой области, которая называется областью применимости алгоритма.

На практике наиболее распространены следующие формы представления алгоритмов:

- **словесная** (запись на естественном языке);
- **графическая** (изображения из графических символов);
- **псевдокоды** (полуформализованные описания алгоритмов на условном алгоритмическом языке, включающие в себя как элементы языка программирования, так и фразы естественного языка, общепринятые математические обозначения и др.);
- **программная** (тексты на языках программирования).

Словесный способ записи алгоритмов представляет собой описание последовательных этапов обработки данных. Алгоритм задается в произвольном изложении на естественном языке.

Например. Записать алгоритм нахождения **наибольшего общего делителя (НОД)** двух натуральных чисел (алгоритм Эвклида).

Алгоритм может быть следующим:

1. задать два числа;
2. если числа равны, то взять любое из них в качестве ответа и остановиться, в противном случае продолжить выполнение алгоритма;
3. определить большее из чисел;
4. заменить большее из чисел разностью большего и меньшего из чисел;
5. повторить алгоритм с шага 2.

Описанный алгоритм применим к любым натуральным числам и должен приводить к решению поставленной задачи. Убедитесь в этом самостоятельно, определив с помощью этого алгоритма наибольший общий делитель чисел 125 и 75.

Словесный способ не имеет широкого распространения, так как такие описания:

- строго не формализуемы;
- страдают многословностью записей;
- допускают неоднозначность толкования отдельных предписаний.

Графический способ представления алгоритмов является более компактным и наглядным по сравнению со словесным.

При графическом представлении алгоритм изображается в виде последовательности связанных между собой функциональных блоков, каждый из которых соответствует выполнению одного или нескольких действий.

Такое графическое представление называется схемой алгоритма или **блок-схемой**. В блок-схеме каждому типу действий (вводу исходных данных, вычислению значений выражений, проверке условий, управлению повторением действий, окончанию обработки и т.п.) соответствует геометрическая фигура, представленная в виде **блочного символа**. Блочные символы соединяются **линиями переходов**, определяющими очередность выполнения действий.

Блок "**процесс**" применяется для обозначения действия или последовательности действий, изменяющих значение, форму представления или размещения данных. Для улучшения наглядности схемы несколько отдельных блоков обработки можно объединять в один блок. Представление отдельных операций достаточно свободно.

Блок "**решение**" используется для обозначения переходов управления по условию. В каждом блоке "решение" должны быть указаны вопрос, условие или сравнение, которые он определяет.

Блок "**модификация**" используется для организации циклических конструкций. (Слово модификация означает видоизменение, преобразование). Внутри блока записывается параметр цикла, для которого указываются его начальное значение, граничное условие и шаг изменения значения параметра для каждого повторения.

Блок "**предопределенный процесс**" используется для указания обращений к вспомогательным алгоритмам, существующим автономно в виде некоторых самостоятельных модулей, и для обращений к библиотечным подпрограммам.

Алгоритмы можно представлять как некоторые структуры, состоящие из отдельных **базовых** (т.е. основных) **элементов**. Естественно, что при таком подходе к алгоритмам изучение основных принципов их конструирования должно начинаться с изучения этих базовых элементов. Для их описания будем использовать язык схем алгоритмов и школьный алгоритмический язык.

Логическая структура любого алгоритма может быть представлена комбинацией трех базовых структур: следование, ветвление, цикл

Характерной особенностью базовых структур является наличие в них **одного входа и одного выхода.**

1. **Базовая структура "следование"** образуется последовательностью действий, следующих одно за другим.
2. **Базовая структура "ветвление"** обеспечивает в зависимости от результата проверки условия (**да** или **нет**) выбор одного из альтернативных путей работы алгоритма. Каждый из путей ведет к **общему выходу**, так что работа алгоритма будет продолжаться независимо от того, какой путь будет выбран. Структура **ветвление** существует в четырех основных вариантах:

    - если – то;
    - если – то – иначе;
    - выбор;
    - выбор – иначе.

3. **Базовая структура "цикл"** обеспечивает **многократное выполнение некоторой совокупности действий,** которая называется **телом цикла**.

Особенностью итерационного цикла является то, что число повторений операторов тела цикла заранее неизвестно. Для его организации используется цикл типа пока. Выход из итерационного цикла осуществляется в случае выполнения заданного условия

На каждом шаге вычислений происходит **последовательное приближение к искомому результату и проверка условия достижения последнего.**

В итерационных алгоритмах необходимо обеспечить обязательное достижение условия выхода из цикла (**сходимость итерационного процесса**). В противном случае произойдет **"зацикливание"** алгоритма, т.е. не будет выполняться основное свойство алгоритма – **результативность**.

Возможны случаи, когда внутри тела цикла необходимо повторять некоторую последовательность операторов, т. е. организовать внутренний цикл. Такая структура получила название **цикла в цикле** или **вложенных циклов**. Глубина вложения циклов (то есть количество вложенных друг в друга циклов) может быть различной.

При использовании такой структуры для экономии машинного времени необходимо выносить из внутреннего цикла во внешний все операторы, которые не зависят от параметра внутреннего цикла.

1.1. ПРОГРАММНЫЙ СПОСОБ ЗАПИСИ АЛГОРИТМОВ. УРОВЕНЬ ЯЗЫКА ПРОГРАММИРОВАНИЯ

При записи алгоритма в словесной форме, в виде блок-схемы или на псевдокоде допускается определенный произвол при изображении команд. Вместе с тем такая запись точна настолько, что позволяет человеку понять суть дела и исполнить алгоритм.

Однако на практике в качестве исполнителей алгоритмов используются специальные автоматы – компьютеры. Поэтому алгоритм, предназначенный для исполнения на компьютере, должен быть записан на понятном ему языке. И здесь на первый план выдвигается необходимость точной записи команд, не оставляющей места для произвольного толкования их исполнителем.

Следовательно, **язык для записи алгоритмов должен быть формализован.** Такой язык принято называть **языком программирования,** а запись алгоритма на этом языке – **программой для компьютера.**

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

Любой алгоритм, как мы знаем, есть последовательность предписаний, выполнив которые можно за конечное число шагов перейти от исходных данных к результату. В зависимости от степени детализации предписаний обычно

определяется уровень языка программирования – чем меньше детализация, тем выше уровень языка.

По этому критерию можно выделить следующие уровни языков программирования:

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

Машинные языки и машинно-ориентированные языки – это языки **низкого уровня**, требующие указания мелких деталей процесса обработки данных. Языки же **высокого уровня** имитируют естественные языки, используя некоторые слова разговорного языка и общепринятые математические символы. Эти языки более удобны для человека.

Языки высокого уровня делятся на:

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

Каждый компьютер имеет свой машинный язык, то есть свою совокупность машинных команд, которая отличается количеством адресов в команде, назначением информации, задаваемой в адресах, набором операций, которые может выполнить машина и др.

При программировании на машинном языке программист может держать под своим контролем каждую команду и каждую ячейку памяти, использовать все возможности имеющихся машинных операций.

Но процесс написания программы на машинном языке очень **трудоемкий** и **утомительный**. Программа получается **громоздкой, труднообозримой, ее трудно отлаживать, изменять и развивать.**

Поэтому в случае, когда нужно иметь эффективную программу, в максимальной степени учитывающую специфику конкретного компьютера, вместо машинных языков используют близкие к ним машинно-ориентированные языки (ассемблеры).

Язык ассемблера позволяет программисту пользоваться **текстовыми мнемоническими** (то есть легко запоминаемыми человеком) **кодами**, по своему усмотрению **присваивать символические имена регистрам компьютера и памяти, а также задавать удобные для себя способы адресации**. Кроме того, он позволяет использовать различные системы счисления (например, десятичную или шестнадцатеричную) для представления числовых констант, использовать в программе комментарии и др.

Программы, написанные на языке ассемблера, требуют значительно меньшего объема памяти и времени выполнения. Знание программистом языка ассемблера и машинного кода дает ему понимание архитектуры машины. Несмотря на то, что большинство специалистов в области программного обеспечения разрабатывают программы на языках высокого уровня, таких, как Visual Basic или C, наиболее мощное и эффективное программное обеспечение полностью или частично написано на языке ассемблера.

Языки высокого уровня были разработаны для того, чтобы освободить программиста от учета технических особенностей конкретных компьютеров, их архитектуры. В противоположность этому, язык ассемблера разработан с целью учесть конкретную специфику процессора. Следовательно, для того, чтобы написать программу на языке ассемблера для конкретного компьютера, важно знать его архитектуру.

Перевод программы с языка ассемблера на машинный язык осуществляется специальной программой, которая называется **ассемблером** и является, по сути, простейшим **транслятором.**

Основные преимущества алгоритмических языков перед машинными:

- **алфавит алгоритмического языка значительно шире алфавита машинного языка**, что существенно повышает наглядность текста программы;

- **набор операций,** допустимых для использования, **не зависит от набора машинных операций**, а выбирается из соображений удобства формулирования алгоритмов решения задач определенного класса;
- **формат предложений** достаточно **гибок и удобен** для использования, что позволяет с помощью одного предложения задать достаточно содержательный этап обработки данных;
- требуемые операции задаются с помощью **общепринятых математических обозначений**;
- **данным в алгоритмических языках присваиваются индивидуальные имена**, выбираемые программистом;
- в языке может быть предусмотрен значительно **более широкий набор типов данных** по сравнению с набором машинных типов данных.

Таким образом, алгоритмические языки в значительной мере являются **машинно-независимыми**. Они облегчают **работу программиста** и **повышают надежность** создаваемых **программ**.

1.2. ТИПЫ ДАННЫХ В ЯЗЫКЕ ПАСКАЛЬ. ОПЕРАЦИИ НАД НИМИ

Для обработки ЭВМ данные представляются в виде величин и их совокупностей. С понятием величины связаны такая важная характеристика, как ее тип.

Тип определяет:

- возможные значения переменных, констант, функций, выражений, принадлежащих к данному типу;
- внутреннюю форму представления данных в ЭВМ;
- операции и функции, которые могут выполняться над величинами, принадлежащими к данному типу.

В языке Паскаль тип величины задают **заранее**. Все переменные, используемые в программе, должны быть объявлены в разделе описания, начинающегося со служебного слова **var** с указанием их типа. Обязательное описание типа приводит к избыточности в тексте программ, но такая избыточность является важным вспомогательным средством разработки программ и рассматривается как необходимое свойство современных алгоритмических языков высокого уровня. Иерархия типов в языке Паскаль такая:

Простые:

Порядковые

- Целые
- Логические
- Символьные
- Перечисляемые
- Интервальные

Вещественные

Структуированные:

- Массивы
- Строки
- Множества
- Записи
- Файлы

Указатели

Простые типы данных

В таблице приведены простые типы данных в языке Паскаль, объем памяти, необходимый для хранения одной переменной указанного типа, множество допустимых значений и применимые операции.

Таблица 1

Идентификатор	*Длина (байт)*	*Диапазон значений*	*Операции*
Целые типы			
integer	2	-32768..32767	+, -, /, *, Div, Mod, >=, <=, =, <>, <, >
byte	1	0..255	+, -, /, *, Div, Mod, >=, <=, =, <>, <, >
word	2	0..65535	+, -, /, *, Div, Mod, >=, <=, =, <>, <, >
shortint	1	-128..127	+, -, /, *, Div, Mod, >=, <=, =, <>, <, >
longint	4	-2147483648..2147483647	+, -, /, *, Div, Mod, >=, <=, =, <>, <, >
Вещественные типы			
real	6	$2{,}9x10^{-39}$ – $1{,}7x10^{38}$	+, -, /, *, >=, <=, =, <>, <, >
single	4	$1{,}5x10^{-45}$ – $3{,}4x10^{38}$	+, -, /, *, >=, <=, =, <>, <, >
double	8	$5x10^{-324}$ – $1{,}7x10^{308}$	+, -, /, *, >=, <=, =, <>, <, >
extended	10	$3{,}4x10^{-4932}$ – $1{,}1x10^{4932}$	+, -, /, *, >=, <=, =, <>, <, >
Логический тип			
boolean	1	true, false	Not, And, Or, Xor, >=, <=, =, <>, <, >
Символьный тип			
char	1	все символы кода ASCII	+, >=, <=, =, <>, <, >

Перечисляемый и интервальный тип относятся к типам, определяемым пользователем и будут рассмотрены нами позже.

Дополнительные сведения о типах данных..

Порядковые типы, выделяемые из группы простых типов, характеризуются следующими свойствами:

- все возможные значения порядкового типа представляют собой ограниченное упорядоченное множество;
- к любому порядковому типу может быть применена стандартная функция Ord, которая в качестве результата возвращает порядковый номер конкретного значения в данном типе;
- к любому порядковому типу могут быть применены стандартные функции Pred и Succ, которые возвращают предыдущее и последующее значения соответственно;
- к любому порядковому типу могут быть применены стандартные функции Low и High, которые возвращают наименьшее и наибольшее значения величин данного типа.

В языке Паскаль введены понятия эквивалентности и совместимости типов. Два типа T1 и T2 являются эквивалентными (идентичными), если выполняется одно из двух условий:

- T1 и T2 представляют собой одно и то же имя типа;
- тип T2 описан с использованием типа T1 с помощью равенства или последовательности равенств.

Например:

- type
- T1 = Integer;
- T2 = T1;
- T3 = T2;

Менее строгие ограничения накладываются на совместимость типов. Так, типы являются совместимыми, если:

- они эквивалентны;
- являются оба либо целыми, либо действительными;
- один тип – интервальный, другой – его базовый;

- оба интервальные с общим базовым;
- один тип – строковый, другой – символьный.

В языке Паскаль ограничения на совместимость типов можно обойти с помощью приведения типов. Приведение типов позволяет рассматривать одну и ту же величину в памяти ЭВМ как принадлежащую разным типам. Для этого используется конструкция

Имя_Типа(переменная или значение)

Например, Integer('Z') представляет собой значение кода символа 'Z' в двухбайтном представлении целого числа, а Byte(534) даст значение 22, поскольку целое число 534 имеет тип Word и занимает два байта, а тип Byte занимает один байт, и в процессе приведения старший байт будет отброшен.

Переменной называют элемент программы, который предназначен для хранения, коррекции и передачи данных внутри программы. Все переменные программы в языке Паскаль должны быть объявлены в разделе описания переменных.

Наряду с переменными в пограммах используются и **константы**. Константа – это идентификатор, обозначающий некоторую неизменную величину определенного типа. Константы, как и переменные, должны объявляться в соответствующем разделе программы **const**.

В языке Паскаль применяется несколько стандартных видов констант:

- **Целочисленные** константы могут быть определены посредством чисел, записанных в десятичном или шестнадцатиричном формате данных. Это число не должно содержать десятичной точки.
- **Вещественные** константы могут быть определены числами, записанными в десятичном формате данных с использованием десятичной точки.
- **Символьные** константы могут быть определены посредством некоторого символа (заключенного в апострофы).
- **Строковые** константы могут быть определены последовательностью произвольных символов (заключенных в апострофы).
- **Типизированные** константы представляют собой инициализированные переменные, которые могут использоваться в программах наравне с обычными переменными. Каждой типизированной константе ставится в соответствие имя, тип и начальное значение.

Например:

- year: integer = 2001;
- symb: char = '?';
- money: real = 57.23;

Выражения

Выражение задает правило вычисления некоторого значения. Выражение состоит из констант, переменных, указателей функций, знаков операций и скобок.

Математические операции

В таблице приведены основные математические операции языка Паскаль.

Таблица 2

Символ операции	*Название операции*	*Пример*
*	умножение	2*3 (результат: 6)
/	деление	30/2 (результат: 1.5E+01)
+	сложение	2+3 (результат: 5)
-	вычитание	5-3 (результат: 2)
div	целочисленное деление	5 div 2 (результат: 2)
mod	остаток от деления	5 mod 2 (результат: 1)

Логические операции

Над логическими аргументами в языке Паскаль определены следующие операции:

- NOT – логическое отрицание ("НЕ")
- AND – логическое умножение ("И")
- OR – логическое сложение ("ИЛИ")
- XOR – логическое "Исключающее ИЛИ"

Результаты выполнения этих операций над переменными A и B логического типа приведены в таблице истинности.

Таблица 3

A	B	not A	A and B	A or B	A xor B
true	true	false	true	true	false
true	false		false	true	true
false	true	true	false	true	true
false	false		false	false	false

Операции отношения

К операциям отношения в языке Паскаль относятся такие операции, как:

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

В операциях отношения могут принимать участие не только числа, но и символы, строки, множества и указатели.

Приоритет операций

Порядок вычисления выражения определяется старшинством (приоритетом) содержащихся в нем операций. В языке Паскаль принят следующий приоритет операций:

- унарная операция not, унарный минус -, взятие адреса @
- операции типа умножения: * / div mod and
- операции типа сложения: + – or xor
- операции отношения: = <> < > <= >= in

Порядок выполнения операций переопределить можно с помощью скобок.

Например, 2*5+10 равно 20, но 2*(5+10) равно 30.

Основные математические функции

В этом разделе приведены основные математические функции, встроенные в системную библиотеку языка Паскаль.

Abs(X)

Возвращает абсолютное значение числа X.

Cos(X), Sin(X)

Возвращает косинус (синус) числа X, где X – угол в радианах.

Функций тангенс и котангенс в языке Паскаль нет. Для их вычисления используйте выражение sin(x)/cos(x) (или cos(x)/sin(x) для котангенса).

ArcTan(X)

Возвращает арктангенс числа X.

Exp(X)

Возвращает число, равное e в степени X.

Ln(x)

Возвращает число, равное натуральному логарифму от числа X.

Pi

Число Пи.

Sqr(X)

Возвращает число, равное квадрату числа X.

Функции возведения в произвольную степень в языке Паскаль нет. Используйте многократное умножение для возведения в целочисленную степень, либо функции Exp и Ln для возведения в вещественную степень.

a^b=exp(b*ln(a))

Sqrt(X)

Возвращает число, равное квадратному корню из числа X.

Trunc(X)

Возвращает число, равное целой части числа X. (Происходит отбрасывание дробной части числа X. Результат выполнения имеет тип Longint).

Frac(X)

Возвращает число, равное дробной части числа X.

Int(X)

Возвращает число, равное целой части числа X. Результат выполнения функции – real.

Round(X)

Функция округляет число X. Возвращаемое значение имеет тип Longint.

Random(X)

Возвращает случайное целое число в диапазоне 0..X. Если аргумент опущен (Random), то возвращается случайное вещественное число от 0 до 1.

Перед использованием random в программах рекомендуется сначала инициализировать генератор псевдослучайных чисел процедурой Randomize. В противном случае при каждом запуске программы будет генерироваться одна и та же последовательность случайных чисел.

Пример. Вывод на экран 5 случайных чисел в диапазоне -10..10.

```
var i: integer;
begin
  randomize;
  for i:=1 to 5 do writeln(random(21)-10);
end.
```

Inc(X,Y)

Увеличивает значение числа X на Y. Если число Y не указано, то увеличение происходит на 1.

Dec(X,Y)

Уменьшает значение числа X на Y. Если число Y не указано, то уменьшение происходит на 1.

Конечно, можно использовать и аналоги X:=X+Y; X:=X-Y, но они работают медленнее.

1.3. ОПЕРАТОРЫ ЯЗЫКА ПАСКАЛЬ

Ввод данных

Для ввода исходных данных чаще всего используется процедура ReadLn:

ReadLn(A1,A2,...AK);

Процедура производит чтение K значений исходных данных и присваивает эти значения переменным A1, A2, ..., AK.

При вводе исходных данных происходит преобразование из внешней формы представления во внутреннюю, определяемую типом переменных. Переменные, образующие список ввода, могут принадлежать либо к целому, либо к

действительному, либо к символьному типам. Чтение исходных данных логического типа в языке Паскаль недопустимо.

Значения исходных данных могут отделяться друг от друга пробелами и нажатием клавиш табуляции и Enter.

Не допускается разделение вводимых чисел запятыми!

Вывод данных

Для вывода результатов работы программы на экран используются процедуры:

```
Write(A1,A2,...AK);
WriteLn(A1,A2,...AK);
```

Первый из этих операторов производит вывод значений переменных A1, A2,...,AK в строку экрана. Второй оператор, в отличие от первого, не только производит вывод данных на экран, но и производит переход к началу следующей экранной строки. Если процедура writeln используется без параметров, то она просто производит пропуск строки и переход к началу следующей строки.

Переменные, составляющие список вывода, могут относиться к целому, действительному, символьному или булевскому типам. В качестве элемента списка вывода кроме имен переменных могут использоваться выражения и строки.

Форма представления значений в поле вывода соответствует типу переменных и выражений: величины целого типа выводятся как целые десятичные числа, действительного типа – как действительные десятичные числа с десятичным порядком, символьного типа и строки – в виде символов, логического типа – в виде логических констант TRUE и FALSE.

Оператор вывода позволяет задать ширину поля вывода для каждого элемента списка вывода. В этом случае элемент списка вывода имеет вид A:K, где A – выражение или строка, K – выражение либо константа целого типа. Если выводимое значение занимает в поле вывода меньше позиций, чем K, то перед этим значением располагаются пробелы. Если выводимое значение не помещается в ширину поля K, то для этого значения будет отведено необходимое количество позиций.

Для величин действительного типа элемент списка вывода может иметь вид A:K:M, где A – переменная или выражение действительного типа, K – ширина поля

вывода, M – число цифр дробной части выводимого значения. K и M – выражения или константы целого типа. В этом случае действительные значения выводятся в форме десятичного числа с фиксированной точкой.

Пример записи операторов вывода:

```
var rA, rB: Real;
   iP,iQ:Integer;
   bR, bS: Boolean;
   chT, chV, chU, chW: Char;
begin
  . . .
  WriteLn(rA, rB:10:2);
  WriteLn(iP, iQ:8);
  WriteLn(bR, bS:8);
  WriteLn(chT, chV, chU, chW);
end.
```

Структура программы на языке Паскаль

Программа на языке Паскаль состоит из заголовка, разделов описаний и раздела операторов. Заголовок программы содержит имя программы, например:

```
Program PRIM;
```

Описания могут включать в себя:

- раздел подключаемых библиотек (модулей);
- раздел описания меток;
- раздел описания констант;
- раздел описания типов;
- раздел описания переменных;
- раздел описания процедур и функций.

Раздел описания модулей определяется служебным словом USES и содержит имена подключаемых модулей (библиотек) как входящих в состав системы Pascal, так и написанных пользователем. Раздел описания модулей должен быть первым среди разделов описаний. Имена модулей отделяются друг от друга запятыми:

```
uses CRT, Graph;
```

Любой оператор в программе может быть помечен меткой. Имя метки задается по правилам образования идентификаторов Турбо Паскаль. В качестве метки также могут использоваться произвольные целые числа без знака, содержащие не более четырех цифр. Метка ставится перед оператором и отделяется от него двоеточием. Все метки, используемые в программе, должны быть перечислены в **разделе описания меток**, *например*: label 3, 471, 29, Quit;

Описание констант позволяет использовать имена как синонимы констант, их необходимо определить в **разделе описания констант**:

```
const K= 1024; MAX= 16384;
```

В **разделе описания переменных** необходимо указать все переменные, используемые в программе, и определить их тип:

```
var P,Q,R: Integer;
   A,B:   Char;
   F1,F2: Boolean;
```

Описание типов, процедур и функций будет рассмотрено позже. Отдельные разделы описаний могут отсутствовать, но следует помнить, что в Паскаль – программе должны быть обязательно описаны все компоненты программы.

Раздел операторов представляет собой составной оператор, который содержит между служебными словами

```
begin.....end
```

последовательность операторов. Операторы отделяются друг от друга символом ;. Текст программы заканчивается символом точка.

Кроме описаний и операторов Паскаль – программа может содержать комментарии, которые представляют собой произвольную последовательность символов, расположенную между открывающей скобкой комментариев { и закрывающей скобкой комментариев }.

Пример 1

```
program Primer; {вычисление суммы двух чисел}
var
  x,y,s: integer;
begin
 WriteLn('Введите через пробел два числа ');
 ReadLn(x,y);
 s := x + y;
 WriteLn('Сумма чисел равна ',s);
end.
```

Данная программа запрашивает с клавиатуры два числа, находит их сумму и выводит ответ. Теперь сделаем так, чтобы программа сначала очищала экран, выполняла свои действия, а в конце работы позволяла пользователю посмотреть результат, ожидая его нажатия клавиши.

Пример 2

```
program Primer; {вычисление суммы двух чисел}
uses Crt; {подключение модуля, необходимого для процедур
        очистки экрана и задержки}
var
 x,y,s: integer;
begin
 ClrScr; {очистка экрана}
 WriteLn('Введите через пробел два числа ');
 ReadLn(x,y);
 s := x + y;
 WriteLn('Сумма чисел равна ',s);
 ReadKey; {ожидание нажатия клавиши}
end.
```

В языке Паскаль используется два оператора для реализации условных переходов – IF и CASE, а также оператор безусловного перехода GOTO. Они позволяют нарушить последовательный порядок выполнения инструкций программы.

Оператор условного перехода

Оператор условного перехода в языке Паскаль имеет вид:

if *условие* then *оператор 1* else *оператор 2*;

условие – это логическое выражение, в зависимости от которого выбирается одна из двух альтернативных ветвей алгоритма. Если значение условия истинно (TRUE), то будет выполняться *оператор 1*, записанный после ключевого слова then. В противном случае будет выполнен *оператор 2*, следующий за словом else, при этом *оператор 1* пропускается. После выполнения указанных операторов программа переходит к выполеннию команды, стоящей непосредственно после оператора if.

Необходимо помнить, что перед ключевым словом else точка с запятой никогда не ставится!

else – часть в операторе if может отсутствовать:

if *условие* then *оператор 1*;

Тогда в случае невыполнения логического условия управление сразу передается оператору, стоящему в программе после конструкции if.

Следует помнить, что синтаксис языка допускает запись только одного оператора после ключевых слов then и else, поэтому группу инструкций обязательно надо объединять в составной оператор (окаймлять операторными скобками begin ... end). В противном случае возникает чаще всего логическая ошибка программы, когда компилятор языка ошибок не выдает, но программа тем не менее работает неправильно.

Примеры.

```
if x > 0 then modul := x else modul := -x;

if k > 0 then WriteLn('k – число положительное');

if min > max then begin
    t := min;
    min := max;
    max := t;
  end;
```

Оператор выбора

Часто возникают ситуации, когда приходится осуществлять выбор одного из нескольких альтернативных путей выполнения программы. Несмотря на то, что такой выбор можно организовать с помощью оператора if .. then, удобнее воспользоваться специальным оператором выбора. Его формат:

```
case выражение of
   вариант : оператор;
   ...
   вариант : оператор;
end;
```

или

```
case выражение of
   вариант : оператор;
   ...
   вариант : оператор;
   else оператор
end;
```

выражение, которое записывается после ключевого слова case, называется ***селектором***, оно может быть любого перечисляемого типа. *вариант* состоит из одной или большего количества констант или диапазонов, разделенных запятыми. Они должны принадлежать к тому же типу, что и селектор, причем недопустимо более одного упоминания *варианта* в записи инструкции case. Из перечисленного множества *операторов* будет выбран только тот, перед которым записан *вариант*, совпадающий со значением селектора. Если такого *варианта* нет, выполняется *оператор*, следующий за словом else (если он есть).

Пример

```
case ch of
   'A'..'Z', 'a'..'z' : WriteLn('Буква');
   '0'..'9'           : WriteLn('Цифра');
   '+', '-', '*', '/' : WriteLn('Оператор');
   else WriteLn('Специальный символ')
end;
```

Оператор безусловного перехода

Помимо операторов условного перехода существует также оператор безусловного перехода goto. Формат:

goto *метка*

Оператор goto переходит при выполнении программы к определенному оператору программы, перед которым находится *метка*. *Метка* должна быть описана в разделе описания меток той программы (процедуры или функции), в которой она используется. Нельзя перейти из одной процедуры или функции в другую.

Необходимо, чтобы в программе существовал оператор, отмеченный указанной меткой. Она записывается перед оператором и отделяется от него двоеточием.

Пример

```
label 1;
begin
  ...
  goto 1;
  ...
  1: WriteLn('Переход к метке 1');
end.
```

Учтите! Само понятие структурного программирования и общепринятый стиль программирования на структурных языках **НЕ ПРИВЕТСТВУЕТ** применение меток и операторов перехода в программах. Это затрудняет понимание программы как автором, так и потребителями, кроме того, применение меток отрицательно сказывается на эффективности генерируемого кода.

ЦИКЛЫ

В большинстве задач, встречающихся на практике, необходимо производить многократное выполнение некоторого действия. Такой многократно повторяющийся участок вычислительного процесса называется ***циклом***.

Если заранее известно количество необходимых повторений, то цикл называется ***арифметическим***. Если же количество повторений заранее неизвестно, то говорят об ***итерационном*** цикле.

В итерационных циклах производится проверка некоторого условия, и в зависимости от результата этой проверки происходит либо выход из цикла, либо повторение выполнения тела цикла. Если проверка условия производится перед выполнением блока операторов, то такой итерационный цикл называется циклом ***с предусловием*** (цикл "пока"), а если проверка производится после выполнения тела цикла, то это цикл ***с постусловием*** (цикл "до").

Особенность этих циклов заключается в том, что тело цикла с постусловием всегда выполняется хотя бы один раз, а тело цикла с предусловием может ни разу не выполниться. В зависимости от решаемой задачи необходимо использовать тот или иной вид итерационных циклов.

Арифметические циклы

Синтаксис:

for *переменная* := *значение 1* to *значение 2* do *оператор*

или

for *переменная* := *значение 1* downto *значение 2* do *оператор*

Оператор for вызывает *оператор*, находящийся после слова do, по одному разу для каждого значения в диапазоне от *значения 1* до *значения 2*.

Переменная цикла, начальное и конечное значения должны иметь порядковый тип. Со словом to, значение переменной цикла **увеличивается** на 1 при каждой итерации цикла. Со словом downto, значение переменной цикла **уменьшается** на 1 при каждой итерации цикла. Не следует самостоятельно изменять значение управляющей переменной внутри цикла.

Как и в случае использования оператора условного прехода, следует помнить, что синтаксис языка допускает запись только одного оператора после ключевого слова do, поэтому, если вы хотите в цикле выполнить группу операторов, обязательно надо объединить их в составной оператор (окаймить операторными скобками begin ... end). В противном случае будет сделана логическая ошибка программы.

Пример 1. Квадраты чисел от 2-х до 10-и.

```
for x:=2 to 10 do WriteLn(x*x);
```

Пример 2. Латинский алфавит.

```
for ch:='A' to 'Z' do Writeln(ch);
```

Пример 3. Использование цикла с downto.

```
for i:=10 downto 1 do WriteLn(i);
```

Пример 4. Использование составного оператора.

```
for x:=1 to 10 do begin
    y:=2*x+3;
    WriteLn('f(',x,')=',y);
  end;
```

Итерационные циклы с предусловием

Синтаксис:

while *выражение* do *оператор*

Оператор после do будет выполняться до тех пор, пока логическое *выражение* принимает истинное значение (True). Логическое *выражение* является условием возобновления цикла. Его истинность проверяется каждый раз перед очередным повторением *оператора* цикла, который будет выполняться лишь до тех пор, пока логическое *выражение* истинно. Как только логическое *выражение* принимает значение ложь (False), осуществляется переход к оператору, следующему за while.

Выражение оценивается до выполнения *оператора*, так что если оно с самого начала было ложным (False), то *оператор* не будет выполнен ни разу.

Здесь также следует помнить, что позволяется использовать только один оператор после ключевого слова do. Если необходимо выполнить группу операторов, то стоит использовать составной оператор.

Пример.

```
eps:=0.001;
while x > eps do x:=x/2;
```

Итерационные циклы с постусловием

Синтаксис:

```
repeat
  оператор;
  оператор;
  ...
  оператор
until выражение
```

Операторы между словами repeat и until повторяются, пока логическое *выражение* является ложным (False). Как только логическое *выражение* становится истинным (True), происходит выход из цикла.

Так как *выражение* оценивается после выполнения *операторов*, то в любом случае *операторы* выполнятся хотя бы один раз.

Пример.

```
repeat
  WriteLn('Введите положительное число');
  ReadLn(x);
until x>0;
```

Операторы завершения цикла

Для всех операторов цикла выход из цикла осуществляется как вследствие естественного окончания оператора цикла, так и с помощью операторов перехода и выхода.

В версии Турбо Паскаль 7.0 определены стандартные процедуры:

```
Break
Continue
```

Процедура Break выполняет безусловный выход из цикла. Процедура Continue обеспечивает переход к началу новой итерации цикла.

Заметим, что хотя и существует возможность выхода из цикла с помощью оператора безусловного перехода goto, делать этого не желательно. Во всех случаях можно воспользоваться специально предназначенными для этого процедурами Break и Continue.

1.4. СТРУКТУРИРОВАННЫЕ ТИПЫ ДАННЫХ

Любой из структурированных типов данных характеризуется множественностью образующих этот тип элементов. Переменная или константа структурированного типа всегда имеет несколько компонент. Каждая из этих компонент, в свою очередь, может принадлежать структурированному типу, что позволяет говорить о возможной вложенности типов.

В языке Паскале пять структурированных типов:

- массивы;
- строки;
- множества;
- записи;
- файлы.

Однако, прежде чем приступить к их изучению, нам надо рассмотреть еще два типа данных – **перечисляемый** и **интервальный**, которые относятся к порядковым типам, нами ранее не рассматривались, но потребуются при изучении нового материала.

Перечисляемый тип данных

Перечисляемый тип представляет собой ограниченную упорядоченную последовательность скалярных констант, составляющих данный тип. Значение каждой константы задается ее именем. Имена отдельных констант отделяются друг от друга запятыми, а вся совокупность констант, составляющих данный перечисляемый тип, заключается в круглые скобки.

Программист объединяет в одну группу в соответствии с каким-либо признаком всю совокупность значений, составляющих перечисляемый тип. Например, перечисляемый тип Rainbow (РАДУГА) объединяет скалярные значения RED, ORANGE, YELLOW, GREEN, LIGHT_BLUE, BLUE, VIOLET (КРАСНЫЙ, ОРАНЖЕВЫЙ, ЖЕЛТЫЙ, ЗЕЛЕНЫЙ, ГОЛУБОЙ, СИНИЙ, ФИОЛЕТОВЫЙ).

Перечисляемый тип Traffic_Light (СВЕТОФОР) объединяет скалярные значения RED, YELLOW, GREEN (КРАСНЫЙ, ЖЕЛТЫЙ, ЗЕЛЕНЫЙ).

Перечисляемый тип описывается в разделе описания типов, например:

```
type  Rainbow = (RED, ORANGE, YELLOW,
          GREEN, LIGHT_BLUE, BLUE, VIOLET);
```

Каждое значение является константой своего типа и может принадлежать только одному из перечисляемых типов, заданных в программе. Например, перечисляемый тип Traffic_Light не может быть определен в одной программе с типом Rainbow, так как оба типа содержат одинаковые константы.

Описание переменных, которые объявлены в разделе описания типов, производится с помощью имен типов. Например:

```
type  Traffic_Light= (RED, YELLOW, GREEN);
var   Section: Traffic_Light;
```

Это означает, что переменная Section может принимать значения RED, YELLOW или GREEN.

Переменные перечисляемого типа могут быть описаны в разделе описания переменных, например:

```
var Section: (RED, YELLOW, GREEN);
```

При этом имена типов отсутствуют, а переменные определяются совокупностью значений, составляющих данный перечисляемый тип.

К переменным перечисляемого типа может быть применим оператор присваивания:

```
Section:= YELLOW;
```

Упорядоченная последовательность значений, составляющих перечисляемый тип, автоматически нумеруется, начиная с нуля и далее через единицу. Отсюда следует, что к перечисляемым переменным и константам могут быть применены операции отношения и стандартные функции Pred (предыдущий), Succ (последующий), Ord (номер).

Интервальный тип данных

Отрезок (диапазон значений) любого порядкового типа может быть определен как интервальный (ограниченный) тип. Отрезок задается диапазоном от минимального до максимального значения констант, разделенных двумя точками. В качестве констант могут быть использованы константы, принадлежащие к целому,

символьному, логическому или перечисляемому типам. Скалярный тип, на котором строится отрезок, называется базовым типом. Примеры отрезков:

```
1..10
-15..25
'a'..'z'
```

Минимальное и максимальное значения констант называются нижней и верхней границами отрезка, определяющего интервальный тип. Нижняя граница должна быть меньше верхней.

Над переменными, относящимися к интервальному типу, могут выполняться все операции и применяться все стандартные функции, которые допустимы для соответствующего базового типа.

Массивы

Массивы – это совокупности однотипных элементов. Характеризуются они следующим:

1. каждый компонент массива может быть явно обозначен и к нему имеется прямой доступ;
2. число компонент массива определяется при его описании и в дальнейшем не меняется.

Для обозначения компонент массива используется имя переменной-массива и так называемые индексы, которые обычно указывают желаемый элемент. Тип индекса может быть только порядковым (кроме longint). Чаще всего используется интервальный тип (диапазон).

Описание типа массива задается следующим образом:

type
имя типа = array[*список индексов*] of *тип*

Здесь *имя типа* – правильный идентификатор; *список индексов* – список одного или нескольких индексных типов, разделенных запятыми; *тип* – любой тип данных.

Вводить и выводить массивы можно только поэлементно.

Пример 1. Ввод и вывод одномерного массива.

```
const
  n = 5;
type
  mas = array[1..n] of integer;
```

```
var
  a: mas;
  i: byte;
begin
  writeln('введите элементы массива');
  for i:=1 to n do readln(a[i]);
  writeln('вывод элементов массива:');
  for i:=1 to n do write(a[i]:5);
end.
```

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

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

Если массивы a и b описаны как:

```
var
  a = array[1..5] of integer;
  b = array[1..5] of integer;
```

то переменные a и b считаются разных типов. Для обеспечения совместимости применяйте описание переменных через предварительное описание типа.

Если типы массивов идентичны, то в программе один массив может быть присвоен другому. В этом случае значения всех переменных одного массива будет присвоены соответствующим элементам второго массива.

Вместе с тем, над массивами не определены операции отношения. Сравнивать два массива можно только поэлементно.

Так как *тип*, идущий за ключевым словом of в описании массива, – любой тип языка Паскаль, то он может быть и другим массивом. Например:

```
type
  mas = array[1..5] of array[1..10] of integer;
```

Такую запись можно заменить более компактной:

```
type
  mas = array[1..5, 1..10] of integer;
```

Таким образом возникает понятие **многомерного** массива. Глубина вложенности массивов произвольная, поэтому количество элементов в списке индексных типов (размерность массива) не ограничена, однако не может быть более 65520 байт.

Работа с многомерными массивами почти всегда связана с организацией вложенных циклов. Так, чтобы заполнить двумерный массив (матрицу) случайными числами, используют конструкцию вида:

```
for i:=1 to m do
for j:=1 to n do a[i,j]:=random(10);
```

Для "красивого" вывода матрицы на экран используйте такой цикл:

```
for i:=1 to m do begin
  for j:=1 to n do write(a[i,j]:5);
  writeln;
end;
```

СТРОКИ

Для обработки строковой информации в Турбо Паскаль введен строковый тип данных. Строкой в Паскале называется последовательность из определенного количества символов. Количество символов последовательности называется длиной строки. Синтаксис:

```
var s: string[n];
var s: string;
```

n – максимально возможная длина строки – целое число в диапазоне 1..255. Если этот параметр опущен, то по умолчанию он принимается равным 255.

Строковые константы записываются как последовательности символов, ограниченные апострофами. Допускается формирование строк с использованием записи символов по десятичному коду (в виде комбинации # и кода символа) и управляющих символов (комбинации ^ и некоторых заглавных латинских букв).

Пример:

```
'Текстовая строка'
#54#32#61
'abcde'^A^M
```

Пустой символ обозначается двумя подряд стоящими апострофами. Если апостроф входит в строку как литера, то при записи он удваивается.

Переменные, описанные как строковые с разными максимальными длинами, можно присваивать друг другу, хотя при попытке присвоить короткой переменной длинную лишние символы будут отброшены.

Выражения типа char можно присваивать любым строковым переменным.

В языке Паскаль имеется простой доступ к отдельным символам строковой переменной: i-й символ переменной st записывается как st[i]. Например, если st – это 'Строка', то st[1] – это 'С', st[2] – это 'т', st[3] – 'р' и так далее.

Над строковыми данными определена операция слияния (конкантенации), обозначаемая знаком +. Например:

a := 'Turbo';
b := 'Pascal';
c := a + b;
В этом примере переменная с приобретет значение 'TurboPascal'.

Кроме слияния над строками определены операции сравнения <,>,=,<>,<=,>=. Две строки сравниваются посимвольно, слева направо, по кодам символов. Если одна строка меньше другой по длине, недостающие символы короткой строки заменяются символом с кодом 0.

В системе Turbo Pascal имеется несколько полезных стандартных процедур и функций, ориентированных на работу со строками. Ниже приводится список этих процедур и функций с краткими пояснениями.

Length(s:string):integer
Функция возвращает в качестве результата значение текущей длины строки-параметра.

Пример.

n := length('Pascal'); {n будет равно 6}
Concat(s1,[s2,...,sn]:string):string
Функция выполняет слияние строк-параметров, которых может быть произвольное количество. Каждый параметр является выражением строкового типа. Если длина строки-результата превышает 255 символов, то она усекается до 255 символов. Данная функция эквивалентна операции конкатенации "+" и работает немного менее эффективно, чем эта операция.

Copy(s:string; index:integer; count:integer):string

Функция возвращает подстроку, выделенную из исходной строки s, длиной count символов, начиная с символа под номером index.

Пример.

```
s := 'Система Turbo Pascal';
s2 := copy(s, 1, 7); {s2 будет равно 'Система'}
s3 := copy(s, 9, 5); {s3 будет равно 'Turbo'}
s4 := copy(s, 15, 6); {s4 будет равно 'Pascal'}
```

Delete(var s:string; index,count:integer)

Процедура удаляет из строки-параметра s подстроку длиной count символов, начиная с символа под номером index.

Пример.

```
s := 'Система Turbo Pascal';
delete(s,8,6); {s будет равно 'Система Pascal'}
```

Insert(source:string; var s:string;index:integer)

Процедура предназначена для вставки строки source в строку s, начиная с символа index этой строки.

Пример.

```
s := 'Система Pascal';
insert('Turbo ',s,9); {s будет равно 'Система Turbo Pascal'}
```

Pos(substr,s:string):byte

Функция производит поиск в строке s подстроки substr. Результатом функции является номер первой позиции подстроки в исходной строке. Если подстрока не найдена, то функция возвращает 0.

Пример.

```
s := 'Система Turbo Pascal';
x1 := pos('Pascal', s); {x1 будет равно 15}
x2 := pos('Basic', s); {x2 будет равно 0}
```

Str(X: *арифметическое выражение*; var st: string)

Процедура преобразует численное выражение X в его строковое представление и помещает результат в st.

Val(st: string; x: *числовая переменная*; var code: integer)

Процедура преобразует строковую запись числа, содержащуюся в st, в числовое представление, помещая результат в x. x – может быть как целой, так и действительной переменной. Если в st встречается недопустимый (с точки зрения правил записи чисел) символ, то преобразование не происходит, а в code записывается позиция первого недопустимого символа. Выполнение программы при этом не прерывается, диагностика не выдается. Если после выполнения процедуры code равно 0, то это свидетельствует об успешно произошедшем преобразовании.

В дополнение приведем некоторые функции, связанные с типом char, но которые тем не менее часто используются при работе со строками.

Chr(n: byte): char

Функция возвращает символ по коду, равному значению выражения n. Если n можно представить как числовую константу, то можно также пользоваться записью #n.

Ord(ch: char): byte;

В данном случае функция возвращает код символа ch.

UpCase(c: char): char;

Если c – строчная латинская буква, то функция возвращает соответствующую прописную латинскую букву, в противном случае символ c возвращается без изменения.

МНОЖЕСТВА

Понятие множества в языке Паскаль основывается на математическом представлении о конечных множествах: это ограниченная совокупность различных элементов. Для построения конкретного множественного типа используется перечисляемый или интервальный тип данных. Тип элементов, составляющих множество, называется базовым типом.

Множественный тип описывается с помощью служебных слов Set of, например:

type M = Set of B;

Здесь M – множественный тип, B – базовый тип.

Пример описания переменной множественного типа:

```
type
  M = Set of 'A'..'D';
var
  MS: M;
```

Принадлежность переменных к множественному типу может быть определена прямо в разделе описания переменных:

var C: Set of 0..7;

Константы множественного типа записываются в виде заключенной в квадратные скобки последовательности элементов или интервалов базового типа, разделенных запятыми, например:

['A', 'C'] [0, 2, 7] [3, 7, 11..14]

Символ вида [ ] означает пустое подмножество. Количество базовых элементов не должно превышать 256.

Множество включает в себя набор элементов базового типа, все подмножества данного множества, а также пустое подмножество. Так, переменная T множественного типа

var T: Set of 1..3;
может принимать восемь различных значений:

[ ] [1] [2] [3] [1,2] [1,3] [2,3] [1,2,3]

К переменным и константам множественного типа применимы операции присваивания(:=), объединения(+), пересечения(*) и вычитания(-):

['A','B'] + ['A','D'] даст ['A','B','D']
['A','D'] * ['A','B','C'] даст ['A']
['A','B','C'] – ['A','B'] даст ['C'].

Результат выполнения этих операций есть величина множественного типа.

К множественным величинам применимы операции: тождественность (=), нетождественность (<>), содержится в (<=), содержит (>=). Результат выполнения этих операций имеет логический тип, например:

['A','B'] = ['A','C'] даст FALSE
['A','B'] <> ['A','C'] даст TRUE
['B'] <= ['B','C'] даст TRUE
['C','D'] >= ['A'] даст FALSE.

Кроме этих операций для работы с величинами множественного типа в языке ПАСКАЛЬ используется операция in, проверяющая принадлежность элемента базового типа, стоящего слева от знака операции, множеству, стоящему справа от

знака операции. Результат выполнения этой операции – булевский. Операция проверки принадлежности элемента множеству часто используется вместо операций отношения, например:

'A' in ['A', 'B'] даст TRUE,
2 in [1, 3, 6] даст FALSE.

ЗАПИСИ

Запись представляет собой совокупность ограниченного числа логически связанных компонент, принадлежащих к разным типам. Компоненты записи называются полями, каждое из которых определяется именем. Поле записи содержит имя поля, вслед за которым через двоеточие указывается тип этого поля. Поля записи могут относиться к любому типу, допустимому в языке Паскаль, за исключением файлового типа.

Описание записи в языке Паскаль осуществляется с помощью служебного слова record, вслед за которым описываются компоненты записи. Завершается описание записи служебным словом end.

Например, телефонный справочник содержит фамилии и номера телефонов, поэтому отдельную строку в таком справочнике удобно представить в виде следующей записи:

```
type TRec = Record
      FIO: String[20];
      TEL: String[7]
    end;
var rec: TRec;
```

Описание записей возможно и без использования имени типа, например:

```
var rec: Record
         FIO: String[20];
         TEL: String[7]
       end;
```

Обращение к записи в целом допускается только в операторах присваивания, где слева и справа от знака присваивания используются имена записей одинакового типа. Во всех остальных случаях оперируют отдельными полями записей. Чтобы обратиться к отдельной компоненте записи, необходимо задать имя записи и через точку указать имя нужного поля, например:

rec.FIO, rec.TEL

Такое имя называется **составным**. Компонентой записи может быть также запись, в таком случае составное имя будет содержать не два, а большее количество имен.

Обращение к компонентам записей можно упростить, если воспользоваться оператором присоединения with.

Он позволяет заменить составные имена, характеризующие каждое поле, просто на имена полей, а имя записи определить в операторе присоединения:

with rec do *оператор*;

Здесь rec – имя записи, *оператор* – оператор, простой или составной. *Оператор* представляет собой область действия оператора присоединения, в пределах которой можно не использовать составные имена. Например для нашего случая:

```
with rec do
  begin
    FIO:='Иванов А.А.';
    TEL:='2223322';
  end;
```

Такая алгоритмическая конструкция полностью идентична следующей:

```
rec.FIO:='Иванов А.А.';
rec.TEL:='2223322';
```

Особой разновидностью записей являются записи с вариантами, которые объявляются с использованием зарезервированного слова case. С помощью записей с вариантами вы можете одновременно сохранять различные структуры данных, которые имеют большую общую часть, одинаковую во все структурах, и некоторые небольшие отличающиеся части.

Например, сконструируем запись, в которой мы будем хранить данные о некоторой геометрической фигуре (отрезок, треугольник, окружность).

```
type
TFigure = record
  type_of_figure: string[10];
  color_of_figure: byte;
  case integer of
    1: (x1,y1,x2,y2: integer);
    2: (a1,a2,b1,b2,c1,c2: integer);
    3: (x,y: integer; radius: word);
```

```
end;
var figure: TFigure;
```

Таким образом, в переменной figure мы можем хранить данные как об отрезке, так и о треугольнике или окружности. Надо лишь в зависимости от типа фигуры обращаться к соответствующим полям записи.

Заметим, что индивидуальные поля для каждого из типов фигур занимают тем не менее одно адресное пространство памяти, а это означает, что одновременное их использование невозможно.

В любой записи может быть только одна вариантная часть. После окончания вариантной части в записи не могут появляться никакие другие поля. Имена полей должны быть уникальными в пределах той записи, где они объявлены.

1.5. ФАЙЛОВЫЕ ТИПЫ ДАННЫХ

Файлы

Введение файлового типа в язык Паскаль вызвано необходимостью обеспечить возможность работы с периферийными (внешними) устройствами ЭВМ, предназначенными для ввода, вывода и хранения данных.

Файловый тип данных или файл определяет упорядоченную совокупность произвольного числа однотипных компонент.

Понятие файла достаточно широко. Это может быть обычный файл на диске, коммуникационный порт ЭВМ, устройство печати, клавиатура или другие устройства.

При работе с файлами выполняются операции ввода – вывода. Операция ввода означает перепись данных с внешнего устройства (из входного файла) в основную память ЭВМ, операция вывода – это пересылка данных из основной памяти на внешнее устройство (в выходной файл).

Файлы на внешних устройствах часто называют физическими файлами. Их имена определяются операционной системой. В программах на языке Паскаль имена файлов задаются с помощью строк. Например, имя файла на диске может иметь вид:

```
'LAB1.DAT'
```

'c:\ABC150\pr.txt'
'my_files'

Типы файлов Турбо Паскаль

Турбо Паскаль поддерживает три файловых типа:

- текстовые файлы;
- типизированные файлы;
- нетипизированные файлы.

Доступ к файлу в программе происходит с помощью переменных файлового типа. Переменную файлового типа описывают одним из трех способов:

file of *тип* – типизированный файл (указан тип компоненты);

text – текстовый файл;

file – нетипизированный файл.

Примеры описания файловых переменных:

```
var
  f1: file of char;
  f2: file of integer;
  f3: file;
  t: text;
```

Стандартные процедуры и функции

Любые дисковые файлы становятся доступными программе после связывания их с файловой переменной, объявленной в программе. Все операции в программе производятся только с помощью связанной с ним файловой переменной.

Assign(f, FileName)
связывает файловую переменную f с физическим файлом, полное имя которого задано в строке FileName. Установленная связь будет действовать до конца работы программы, или до тех пор, пока не будет сделано переназначение.

После связи файловой переменной с дисковым именем файла в программе нужно указать направление передачи данных (открыть файл). В зависимости от этого направления говорят о чтении из файла или записи в файл.

Reset(f)

открывает для чтения/добавления файл, с которым связана файловая переменная f. После успешного выполнения процедуры Reset файл готов к чтению из него первого элемента. Процедура завершается с сообщением об ошибке, если указанный файл не найден.

Если f – типизированный файл, то процедурой reset он открывается для чтения и записи одновременно.

Rewrite(f)

открывает для записи файл, с которым связана файловая переменная f. После успешного выполнения этой процедуры файл готов к записи в него первого элемента. Если указанный файл уже существовал, то все данные из него уничтожаются.

Close(f)

закрывает открытый до этого файл с файловой переменной f. Вызов процедуры Close необходим при завершении работы с файлом. Если по какой-то причине процедура Close не будет выполнена, файл все-же будет создан на внешнем устройстве, но содержимое последнего буфера в него не будет перенесено.

EOF(f): boolean

возвращает значение TRUE, когда при чтении достигнут конец файла. Это означает, что уже прочитан последний элемент в файле или файл после открытия оказался пуст.

Rename(f, NewName)

позволяет переименовать физический файл на диске, связанный с файловой переменной f. Переименование возможно после закрытия файла.

Erase(f)

уничтожает физический файл на диске, который был связан с файловой переменной f. Файл к моменту вызова процедуры Erase должен быть закрыт.

IOResult

возвращает целое число, соответствующее коду последней ошибки ввода – вывода. При нормальном завершении операции функция вернет значение 0. Значение функции IOResult необходимо присваивать какой-либо переменной, так как при каждом вызове функция обнуляет свое значение. Функция IOResult работает только при выключенном режиме проверок ошибок ввода – вывода или с ключом компиляции {$I-}.

Работа с типизированными файлами

Типизированный файл – это последовательность компонент любого заданного типа (кроме типа "файл"). Доступ к компонентам файла осуществляется по их порядковым номерам. Компоненты нумеруются, начиная с 0. После открытия файла указатель (номер текущей компоненты) стоит в его начале на нулевом компоненте. После каждого чтения или записи указатель сдвигается к следующему компоненту.

Запись в файл:

Write(f, *список переменных*);

Процедура записывает в файл f всю информацию из списка переменных.

Чтение из файла:

Read(f, *список переменных*);

Процедура читает из файла f компоненты в указанные переменные. Тип файловых компонент и переменных должны совпадать. Если будет сделана попытка чтения несуществующих компонент, то произойдет ошибочное завершение программы. Необходимо либо точно рассчитывать количество компонент, либо перед каждым чтением данных делать проверку их существования (функция eof, см. выше)

Смещение указателя файла:

Seek(f, n);

Процедура смещает указатель файла f на n-ную позицию. Нумерация в файле начинается с 0.

Определение количества компонент:

FileSize(f): longint;

Функция возвращает количество компонент в файле f.

Определение позиции указателя:

FilePos(f): longint;

Функция возвращает порядковый номер текущего компонента файла f.

Отсечение последних компонент файла:

Truncate(f);

Процедура отсекает конец файла, начиная с текущей позиции включительно.

Пример 1. Программа считывает 10 целых чисел, введенных пользователем, в массив, сохраняет их в файл, а потом считывает из файла и выводит на экран.

```
program my_file1;
uses crt;
var
  f1:file of integer;
  A:array[1..10] of integer;
  i,k:integer;
  j:longint;
begin
  {Очищается экран и вводится десять элементов массива}
  clrscr;
  for i:=1 to 10 do
   begin
     write('a[',i,']=');
     readln(a[i])
   end;
  {файловая переменная f1 связывается с файлом tet.vvv}
  Assign(f1,'c:\tet.vvv');
  {на локальном диске C физически создается пустой файл tet.vvv, если он
существовал, то существующий очищается}
  Rewrite(f1);
```

```
{в цикле последовательно элементы массива записываются в файл, связанный
с переменной f1}
  for i:=1 to 10 do
    write(f1,a[i]);
{работа с файлом завершилась, его нужно закрыть}
  Close(f1);
{открывается существующий файл, с которым связана файловая переменная
f1 и указатель текущей компоненты файла настраивается на начало файла}
  Reset(f1);
{определяем количество компонент файла}
  j:=Filesize(f1);
  for i:=1 to j do
    begin
{читается содержимое очередной компоненты файла в переменную k}
      Read(f1,k);
      writeln(k);
    end;
{закрываем файл}
Close(f1);
  readln;

  end.
```

Пример 2. Программа выводит содержимое заданного файла на экран, контролируя ошибочный ввод.

```
  program my_file2;
  uses crt;
  var
    f2:file of integer;
    s:string;
    k:longint;
    i,j:integer;
    error:byte;
  begin
    clrscr;
    repeat
      writeln('Enter path file');
      readln(s);
      {$I-}
```

```
    assign(f2,s);
    reset(f2);
    {$I+}
    error:=IOResult;
    if error<>0 then writeln('Error type file')
  until error=0;
  k:=filesize(f2);
  for i:=1 to k do
   begin
     read(f2,j);
     writeln(j);
   end;
close(f2);
  repeat
  until keypressed;
end.
```

Пример 3. Программа позволяет открыть файл для добавления данных.

```
program my_file3;
uses crt;
var
  f2:file of integer;
  s:string;
  k:longint;
  i,j:integer;
  error:byte;
begin
  clrscr;
  repeat
    writeln('Enter path file');
    readln(s);
    {$I-}
    assign(f2,s);
    reset(f2);
    {$I+}
    error:=IOResult;
    if error<>0 then writeln('Error type file')
  until error=0;
  k:=filesize(f2);
  Seek(f2,k-1);
```

```
  for i:=1 to 10 do
   begin
     read(j);
     write(f2,j)
   end;
  repeat
  until keypressed;
end.
```

Работа с текстовыми файлами

Текстовый файл – это совокупность строк, разделенных метками конца строки. Сам файл заканчивается меткой конца файла. Доступ к каждой строке возможен лишь последовательно, начиная с первой. Одновременная запись и чтение запрещены.

Чтение из текстового файла:

Read(f, *список переменных*);
ReadLn(f, *список переменных*);

Процедуры читают информацию из файла f в переменные. Способ чтения зависит от типа переменных, стоящих в списке. В переменную char помещаются символы из файла. В числовую переменную: пропускаются символы-разделители, начальные пробелы и считывается значение числа до появления следующего разделителя. В переменную типа string помещается количество символов, равное длине строки, но только в том случае, если раньше не встретились символы конца строки или конца файла. Отличие ReadLn от Read в том, что в нем после прочтения данных пропускаются все оставшиеся символы в данной строке, включая метку конца строки. Если список переменных отсутствует, то процедура ReadLn(f) пропускает строку при чтении текстового файла.

Запись в текстовый файл:

Write(f, *список переменных*);
WriteLn(f, *список переменных*);

Процедуры записывают информацию в текстовый файл. Способ записи зависит от типа переменных в списке (как и при выводе на экран). Учитывается формат вывода. WriteLn от Write отличается тем, что после записи всех значений из

переменных записывает еще и метку конца строки (формируется законченная строка файла).

Добавление информации к концу файла:

Append(f)

Процедура открывает текстовый файл для добавления информации к его концу. Используйте эту процедуру **вместо** Rewrite.

Пример 4. Программа позволяет ввести информацию в текстовый файл и выводит суммарную введенную информацию на экран

```
program my_file4;
uses crt;
var
  f3:text;
  s:string;
  i:integer;
  k:longint;
begin
  clrscr;
 {должен быть уже создан файл r.rrr}
  assign(f3,'c:\r.rrr');
  append(f3);
  for i:=1 to 4 do
    begin
      readln(s);
      writeln(f3,s)
    end;
  close(f3);
  reset(f3);
  repeat
      readln(f3,s);
      writeln(s);
  until eof(f3);
  close(f3);
   readln;
  end.
```

Работа с нетипизированными файлами

Нетипизированные файлы – это последовательность компонент произвольного типа.

Открытие нетипизированного файла:

Reset(f, BufSize)
Rewrite(f, BufSize)

Параметр BufSize задает число байтов, считываемых из файла или записываемых в него за одно обращение. Минимальное значение BufSize – 1 байт, максимальное – 64 K байт. Если BufSize не указан, то по умолчанию он принимается равным 128.

Чтение данных из нетипизированного файла:

BlockRead(f, X, Count, QuantBlock);

Эта процедура осуществляет за одно обращение чтение в переменную X количества блоков, заданное параметром Count, при этом длина блока равна длине буфера. Значение Count не может быть меньше 1. За одно обращение нельзя прочесть больше, чем 64 K байтов.

Необязательный параметр QuantBlock возвращает число блоков, прочитанных текущей операцией BlockRead. В случае успешного завершения операции чтения QuantBlock = Count, в случае аварийной ситуации параметр QuantBlock будет содержать число удачно прочитанных блоков. Отсюда следует, что с помощью параметра QuantBlock можно контролировать правильность выполнения операции чтения.

Запись данных в нетипизированный файл:

BlockWrite(f, X, Count, QuantBlock);

Эта процедура осуществляет за одно обращение запись из переменной X количества блоков, заданное параметром Count, при этом длина блока равна длине буфера.

Необязательный параметр QuantBlock возвращает число блоков, записанных успешно текущей операцией BlockWrite.

Для нетипизированных файлов можно использовать процедуры Seek, FIlePos и FileSize, аналогично соответствующим процедурам типизированных файлов.

1.6. РЕКУРСИВНЫЕ АЛГОРИТМЫ

Язык Паскаль допускает, чтобы подпрограмма вызывала саму себя, то есть допускает рекурсивные обращения. Рассмотрим использование рекурсивных функций.

Функция является рекурсивной, если в ее определении содержится вызов этой функции.

На рекурсии основана декомпозиция задачи на подзадачи, решение которых, насколько это возможно, пытаются свести к уже решенным или к решаемой в настоящий момент задаче. Рекурсия близка к аппарату математической индукции, что определяет удобство реализации индуктивных определений.

Как правило, рекурсивная функция реализует разбор случаев, некоторые из них влекут продолжение процесса вычисления через вызов рекурсивной функции, другие не прибегают к рекурсии, они называются **базис рекурсии**. Феномен рекурсии довольно сложен. И иногда он существенно помогает в решении задач.

Для примера рассмотрим задачу о ханойской башне.

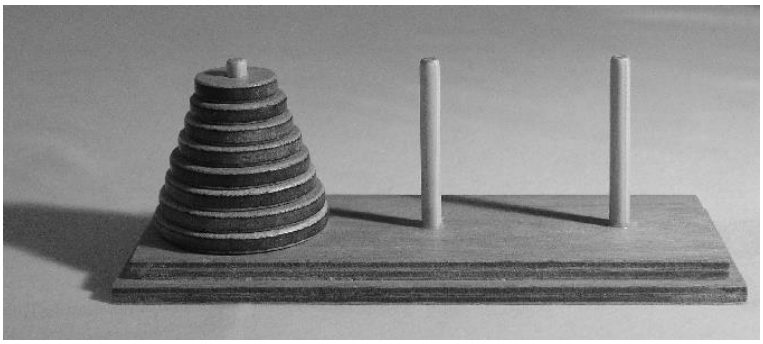


Рисунок 1

Перенести пирамиду из восьми колец. За один раз разрешается переносить только одно кольцо, причём нельзя класть большее кольцо на меньшее.

Предположим, что мы уже научились перекладывать **n-1** колец, тогда алгоритм перекладывания **n** колец такой:

- перекладываем на средний колышек **n-1** колец
- перекладываем на крайний правый колышек самое большое кольцо
- перекладываем на крайний правый колышек **n-1** колец

Последовательно уменьшая **n** мы приходим к одному кольцу, алгоритм перекладывания которого очевиден. После чего легко восстанавливается весь алгоритм.

Следующая задача – вычисление факториала. В математике эту функцию задают следующим образом:

$$factorial(n) = \begin{cases} 1, n = 0,1 \\ n \cdot factorial(n-1) \end{cases}$$

В листинге программы вычисление факториала реализовано двумя способами: рекурсивно и итерационно. До настоящего момента мы использовали всегда итерационные определения.

```
program fact;
var
 n:integer;
function fact1(n:integer):integer;
begin
  if (n=0) or (n=1) then fact1:=1
  else fact1:=n*fact1(n-1)
end;

function fact2(n:integer):integer;
var
 a,i:integer;
begin
  a:=1;
  for i:=1 to n do
    a:=i*a;
  fact2:=a
end;

begin
  write('n=');
  readln(n);
  writeln(fact1(n),fact2(n):5);
  readln
end
```

Как видно, рекурсивное определение факториала не отличается от математического определения. Такое определение является более лаконичным и прозрачным. Для реализации итерации нам потребовалось ввести две дополнительные переменные, организовать цикл.

При вычислении по рекурсии используется так называемая **подстановочная модель:**

(fact1 6)
(* 6 (fact1 5))
(* 6 (* 5 (fact1 4)))
(* 6 (* 5 (* 4 (fact1 3))))
(* 6 (* 5 (* 4 (* 3 (fact1 2)))))
(* 6 (* 5 (* 4 (* 3 (* 2 (fact1 1))))))
(* 6 (* 5 (* 4 (* 3 (* 2 1)))))
(* 6 (* 5 (* 4 (* 3 2))))
(* 6 (* 5 (* 4 6)))
(* 6 (* 5 24))
(* 6 120)
720

При каждом новом обращении к функции параметры, которые она использует, заносятся в стек. Таким образом, при рекурсии память используется более интенсивно. При итерации мы в явном виде вводим в программу две переменные, организуем цикл, то есть мы подробно указываем программе, что нужно делать.

Рассмотрим еще пример – вычисление чисел Фибоначчи. Существует легенда, что Фибоначчи придумал эти числа, наблюдая за размножением кроликов – каждый месяц от взрослой пары рождается по паре кроликов. Рожденный кролик становится взрослым через два месяца. Первоначально имеется одна пара взрослых кроликов. Сколько пар взрослых кроликов будет через год?

1 месяц – 1 пара
2 месяц – 1 пара
3 месяц – 2 пары
4 месяц – 3 пары
5 месяц – 5 пар
6 месяц – 8 пар

Тогда каждое следующее поколение кроликов (n-й год) состоит из суммы уже живущих кроликов (n-1-й год) и вновь родившихся и ставших взрослыми (а родилось их столько, сколько было зрелых пар в n-2-й год). Полученный ряд чисел 1, 1, 2, 3, 5, 8, … и получил название чисел Фибоначчи.

В последствии оказалось, что листья у всех растений вырастают не случайным образом, – листья на ветке располагаются вдоль винтовой линии, находясь друг от друга на строго определенном расстоянии и вращаясь вдоль ветки под строго определенным углом. Число шагов между листьями,

находящимися в точности на одной прямой (то есть, отстоящими друг от друга на целое количество витков спирали), описывается числами Фибоначчи. *Например, дробь 1/2 свойственна злакам, березе, винограду; 1/3 – осоке, тюльпану, ольхе; 2/5 – груше, смородине, сливе; 3/8 – капусте, редьке, льну; 5/13 – ели, жасмину и т.д.*

Искомая формула выглядит так:

$$fib(n) = \begin{cases} 1, n = 1,2 \\ fib(n-1) + fib(n-2) \end{cases}$$

Опять составим программу, в которой реализован как рекурсивный, так и итерационный алгоритмы.

```
program fibonach;
var
 n:integer;
function fib1(n:integer):integer;
begin
 if n=1 then fib1:=1
 else if n=2 then fib1:=1
      else
       fib1:=fib1(n-1)+fib1(n-2)
end;
function fib2(n:integer):integer;
var
  i,a,b,c:integer;
begin
  a:=1;
  b:=1;
  for i:=3 to n do
    begin
      c:=b;
      b:=a+b;
      a:=c;
    end;
  fib2:=b
end;
begin
  write('n=');
  readln(n);
```

```
  writeln(fib1(n));
  writeln(fib2(n));
  readln
end.
```

В приведенном алгоритме мы видим, что рекурсивная функция не усложнилась, оно содержит все тоже математическое определение. Но итерационный алгоритм не выглядит таким прозрачным. В очень многих случаях рекурсивное выполнение подпрограммы может быть более компактным и эффективным, но не следует забывать об опасности переполнения стека.

Чтобы сочинить итеративный алгоритм для чисел Фибоначчи, нужно совсем немного смекалки. Теперь для контраста рассмотрим следующую задачу: сколькими способами можно разменять сумму в 1 доллар, если имеются монеты по 50, 25, 10, 5 и 1 цент?

В более общем случае, можно ли написать процедуру подсчета способов размена для произвольной суммы денег?

У этой задачи есть простое решение в виде рекурсивной процедуры. Предположим, мы как-то упорядочили типы монет, которые у нас есть. В таком случае верно будет следующее уравнение:

Число способов разменять сумму **a** с помощью **n** типов монет равняется

- числу способов разменять сумму **a** с помощью всех типов монет, кроме первого, плюс
- число способов разменять сумму **a** − **d** с использованием всех **n** типов монет, где **d** – достоинство монет первого типа.

Чтобы увидеть, что это именно так, заметим, что способы размена могут быть поделены на две группы: те, которые не используют первый тип монеты, и те, которые его используют. Следовательно, общее число способов размена какой-либо суммы равно числу способов разменять эту сумму без привлечения монет первого типа плюс число способов размена в предположении, что мы этот тип используем. Но последнее число равно числу способов размена для суммы, которая остается после того, как мы один раз употребили первый тип монеты.

Таким образом, мы можем рекурсивно свести задачу размена данной суммы к задаче размена меньших сумм с помощью меньшего количества типов монет. Укажем следующие вырожденные случаи (базы рекурсии):

1. если **a** в точности равно 0, мы считаем, что имеем 1 способ размена.
2. если **a** меньше 0, мы считаем, что имеем 0 способов размена.
3. если **n** равно 0, мы считаем, что имеем 0 способов размена.

Это описание легко перевести в рекурсивную процедуру:

```
program mon;
function first_denomination(n:integer):integer;
begin
  case n of
  1: first_denomination:=1;
  2: first_denomination:=5;
  3: first_denomination:=10;
  4: first_denomination:=25;
  5: first_denomination:=50
  end;
end;
function change(a,n:integer):integer;
begin
  if a=0 then change:=1
  else
    if (a<0) or (n=0) then change:=0
    else
      begin
        change:=change(a,(n-1))+change(a-first_denomination(n),n)
      end;
end;
begin
  writeln(change(100,5));
  readln;
end.
```

Функция first_denomination принимает в качестве входа число доступных типов монет и возвращает достоинство первого типа. Здесь мы упорядочили монеты от самой крупной к более мелким, но годился бы и любой другой порядок. Теперь мы можем ответить на исходный вопрос о размене доллара: 292.

Фукция change порождает древовидно-рекурсивный процесс с избыточностью, похожей на ту, которая возникает в нашей первой реализации fib. (На то, чтобы получить ответ 292, уйдет заметное время.) С другой стороны, неочевидно, как построить более эффективный алгоритм для получения этого результата.

1.7. ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ

Рассматриваемые до сих пор параметры программы обладали тем свойством, что под них выделяется вполне определенный размер памяти и между отдельными объектами устанавливаются связи еще на этапе компиляции и компоновки. Во время работы программы вносить изменения в выделенный размер памяти или установленные связи не удается. Порой это бывает неудобно. Например, программа работает с различным количеством целых чисел. Естественно разместить их в каком-то массиве. Размер массива должен быть определен заранее, и, если программа должна быть универсальной, при определении массива необходимо учитывать случай максимального количества таких чисел. Однако это приведет к неэффективному использованию оперативной памяти.

В языке Паскаль есть возможность по ходу выполнения программы выделять и освобождать необходимую память для размещения в ней различных данных. Таким образом, можно организовывать динамические, т. е. изменяющие размеры, структуры данных. Оперативная память при этом используется наиболее эффективным образом. Такая возможность связана с наличием в языке особых типов данных – указателей. Область оперативной памяти, где можно выделять отдельные участки для размещения данных и освобождать их, будем называть динамической областью памяти или динамической памятью.

Указатель в Turbo Pascal дает адрес объекта определенного типа, называемого базовым типом.

При определении типа-указателя используется этот базовый тип, перед которым ставится признак указателя ^.

Пример.

```
type
Mas = array[l..10] of Real, {базовый тип)
Point = ^Mas;    [тип-указатель на массив из 10 вещественных
чисел}
```

Переменная типа-указателя, которая в дальнейшем будет называться просто указателем, является физическим носителем адреса величины базового типа. Она занимает 4 байта памяти (2 слова). Первое слово дает смещение адреса, второе – адрес сегмента. Значение этой переменной можно задать процедурой New или GetMem. Переменной типа-указателя можно также присвоить значение nil, означающее отсутствие ссылки на какой-либо объект (фактически в этом случае переменной присваивается значение 0).

Пример.

```
type
Complex = record {базовый тип)
Re, Im: Real
end;
Point = ^Complex; {тип-указатель}
AdrInt = ^Integer; {тип-указатель)
var
   X: Complex;
   PI, P2, P3, P4: Point;
   Adl: AdrInt;
 …
 New(PI); {выделение нового элемента типа Complex)
 P2 :- @X; {определение адреса переменной X)
 P3 := P1; [присвоение значения другого указателя)
 P4 := nil; [присвоение значения nil)
 New(Adl); {выделение нового элемента типа Integer)
```

Следует иметь в виду, что указатели, ссылающиеся на объекты разных типов, сами являются объектами разных типов и для них недопустима операция присваивания значений друг другу. Указатели одного и того же типа можно сравнивать с помощью операций = и <>.

Чтобы получить значение элемента, с которым связан указатель, следует взять имя указателя и после него поставить знак ^.

Пример.

```
X := P1^; {переменной X присваивается значение
элемента, на который указывает P1)
P3^ := X; {элементу, на который указывает P3,
присваивается значение переменной x}
```

В Turbo Pascal существует операция получения адреса объекта – @, например X-=@Y; {X – адрес параметра Y)

Использование указателей позволяет осуществлять динамическое распределение памяти.

Процедура New(P), где P – указатель, позволяет выделить область памяти такого размера, в котором можно разместить величину базового типа. Указатель принимает значение адреса выделенной области.

Процедура Dispose (P), где P – указатель, позволяет освободить область памяти, на которую указывает указатель P, для последующего использования. После выполнения процедуры значение указателя P становится неопределнным.

Пример. Ввести в память машины 500 вещественных чисел, а затем вывести их в обратном порядке.

```
program EXAMPLE14;
type
  Mas = array[1..500] of Real;
  Point = ^Mas;
var
  P: Point; {указатель}
  i: Word;
begin
  New(P); {выделение области для 500 чисел}
  WriteLn('Введите 500 чисел'),
  for i := 1 to 500 do Read(P^[i]);
  for i := 500 downto 1  do WriteLn(P^[i]);
  Dispose(P) {освобождение области}
end.
```

Существует и другая возможность работать с динамической памятью – использовать процедуры GetMem и FreeMem,

Процедура GetMem (P,Size), где P – переменная типа-указатель, а Size – выделяемая область памяти в байтах, позволяет выделить в динамической памяти область необходимого размера, при этом адрес выделенной области присваивается переменной P.

Процедура FreeMem (P,Size) – здесь параметры те же, что и в процедуре GetMem, – освобождает занятую область памяти с адресом, задаваемым переменной P и размером Size байтов. Эта область становится свободной для повторного использования, а указатель P становится неопределенным.

1.8. СТРАТЕГИИ И МЕТОДЫ ТЕСТИРОВАНИЯ И ОТЛАДКИ

Целью тестирования является обнаружение ошибок в программе.

Тестирование программного обеспечения охватывает целый ряд видов деятельности, аналогичных последовательности процессов разработки программного обеспечения. В него входят:

- постановка задачи для теста,
- проектирование теста,
- написание тестов,

- тестирование тестов,
- выполнение тестов,
- изучение результатов тестирования.

Решающую роль играет проектирование тестов. Возможен целый ряд подходов к стратегии проектирования тестов. Чтобы ориентироваться в них, рассмотрим два крайних подхода. Первый состоит в том, что тесты проектируются на основе внешних спецификаций программ и модулей, либо спецификаций сопряжения программы или модуля. Программа при этом рассматривается как черный ящик (стратегия 'черного ящика'). Существо такого подхода – проверить соответствует ли программа внешним спецификациям. При этом логика модуля совершенно не принимается во внимание.

Второй подход основан на анализе логики программы (стратегия 'белого ящика'). Существо подхода – в проверке каждого пути, каждой ветви алгоритма. При этом внешняя спецификация во внимание не принимается.

Альтернатива 'черный ящик' – 'белый ящик' является достаточно общей, что подтверждается ее применением не только при тестировании, но, например, в альтернативных направлениях исследований в области искусственного интеллекта.

В этой области сторонники одной точки зрения ('черный ящик') убеждены, что важно совпадение поведения искусственно созданных и естественных интеллектуальных систем, а внутренние механизмы формирования поведения разработчик искусственного интеллекта (ИИ) вовсе не обязан копировать. Это направление ИИ называют машинным интеллектом.

Другая точка зрения ('белый ящик') – изучение механизмов естественного мышления и анализ данных о способах формирования разумного поведения человека является основой построения ИИ. Это направление получило название искусственного разума.

Ярким представителем первого направления является 'Deep Blue', которую 'научили' игре в шахматы на уровне (а может быть и выше) мировых гроссмейстеров. Представителями второго направления являются нейрокомпьютеры.

Ни один из этих подходов не является оптимальным. Из анализа существа первого подхода ясно, что его реализация сводится к проверке всех возможных комбинаций значений на входе программы. Рассмотрим в качестве примера задачу тестирования тривиальной программы, получающей на входе три числа и вычисляющей их среднее арифметическое. Тестирование этой программы для всех значений входных данных невозможно, так как их бесконечное множество. Как правило, исчерпывающее тестирование для всех входных данных программы неосуществимо, поэтому ограничиваются меньшим. При этом **исходят из**

максимальной отдачи теста по сравнению с затратами на его создание. Она измеряется вероятностью того, что тест выявит ошибки, если они имеются в программе. Затраты измеряются временем и стоимостью подготовки, выполнения и проверки результатов теста.

Проанализируем теперь второй подход к тестированию. На рисунке изображены возможные пути небольшого программного модуля. Квадратами представлены последовательные сегменты, а стрелками – передачи управления (с помощью развилок или циклов). Число путей в модуле имеет очень большой порядок, можно утверждать, что модуль удовлетворительно нельзя протестировать.

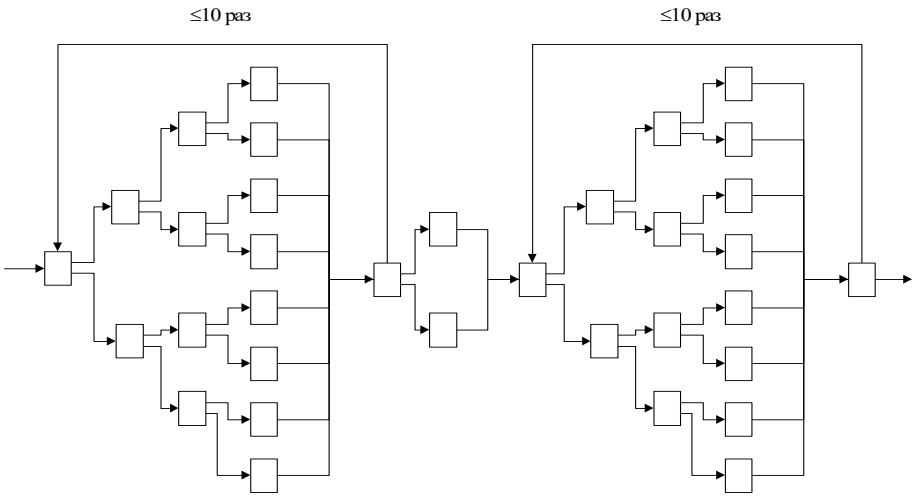


Рисунок 2

Очевидное основание этого утверждения состоит в том, что выполнение даже всех путей не гарантирует соответствия программы ее спецификациям. Допустим, если требовалось написать программу для вычисления кубического корня, а программа фактически вычисляет корень квадратный, то программа будет совершенно неправильной, даже если проверить все пути. Вторая проблема – отсутствующие пути. Если программа реализует спецификации не полностью (например, отсутствует такая специализированная функция, как проверка на отрицательное значение входных данных программы вычисления квадратного корня), никакое тестирование существующих путей не выявит такой ошибки. И, наконец, проблема зависимости результатов тестирования от входных данных. Путь может правильно выполняться для одних данных и неправильно для других. Например, если для определения равенства 3 чисел программируется выражение вида:

IF (A+B+C)/3=A,

то оно будет верным не для всех значений A, B и C (ошибка возникает в том случае, когда из двух значений B или C одно больше, а другое на столько же

меньше А). Если концентрировать внимание только на тестировании путей, нет гарантии, что эта ошибка будет выявлена.

Таким образом, полное тестирование программы невозможно. Тест для любой программы будет обязательно неполным, то есть тестирование не гарантирует отсутствие всех ошибок. Стратегия проектирования тестов заключается в том, чтобы попытаться уменьшить эту неполноту насколько это возможно. При этом ключевым вопросом является следующий: какое подмножество всех возможных тестов имеет наивысшую вероятность обнаружения ошибок при ограниченных времени, трудовых затратах, стоимости, машинном времени и т. п.

Наихудшей из всех методологий является случайный набор тестов, так как он имеет малую вероятность быть оптимальным.

Рекомендуется следующая процедура разработки тестов:

- разрабатывать тесты используя методы стратегии “черного ящика”;
- дополнительное тестирование, используя методы стратегии “белого ящика”.

Методы стратегии ‘белого ящика’

Тестирование по принципу белого ящика характеризуется степенью, какой тесты выполняют или покрывают логику (исходный текст программы).

Метод покрытия операторов

Если отказаться полностью от тестирования всех путей, можно показать, что критерием покрытия является выполнение каждого оператора программы хотя бы один раз.

Это необходимое, но недостаточное условие для приемлемого тестирования по принципу белого ящика.

Пример:

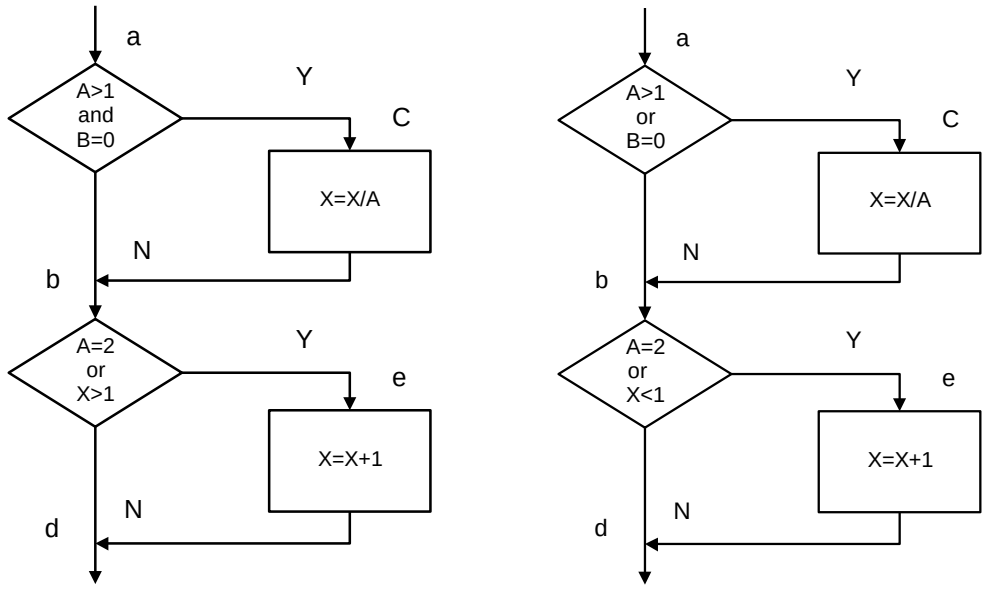


Рисунок 3

В этой программе можно выполнить каждый оператор, записав один единственный тест, который реализовал бы путь *ace.* Т.е., если бы на входе было: A=2, B=0, X=3, каждый оператор выполнился бы один раз. Но этот критерий на самом деле хуже, чем он кажется на первый взгляд. Пусть в первом условии вместо "*and"* стоит *«or»* и во втором вместо *"x>1" – "x<1"* (блок-схема правильной программы приведена первой, а неправильной – второй). Результаты тестирования приведены в таблице 1: **ожидаемый результат определяется по первому алгоритму, а фактический – по второму алгоритму, поскольку определяется чувствительность метода тестирования к ошибкам программирования.** Как видно из этой таблицы 4, ни одна из этих ошибок не будет обнаружена.

Таблица 4

Результат тестирования методом покрытия операторов

Тест	*Ожидаемый результат*	*Фактический результат*	*Результат тестирования*
A=2, B=0, X=3	X=2,5	X=2,5	неуспешно

Метод покрытия решений (покрытия переходов)

Более сильный метод тестирования известен как покрытие решений (покрытие переходов).

Согласно данному методу должно быть написано достаточное число тестов, такое, что каждое направление перехода должно быть реализовано по крайней мере один раз.

Покрытие решений обычно удовлетворяет критерию покрытия операторов. Поскольку каждый оператор лежит на некотором пути, исходящем либо из

оператора перехода, либо из точки входа программы, при выполнении каждого направления перехода каждый оператор должен быть выполнен.

Для программы приведенной на рисунке 3 покрытие решений может быть выполнено двумя тестами, покрывающими пути {*ace, abd*}, либо {*acd,abe*}. Пути {*acd,abe*}покроем, выбрав следующие исходные данные: {A=3, B=0, X=3} и {A=2, B=1, X=1} (результаты тестирования – в таблице 5).

Таблица 5

Результат тестирования методом покрытия решений

Тест	*Ожидаемый результат*	*Фактический результат*	*Результат тестирования*
A=3, B=0, X=3	X=1	X=1	неуспешно
A=2, B=1, X=1	X=2	X=1,5	успешно

Метод покрытия условий

Лучшие результаты по сравнению с предыдущими может дать метод покрытия условий. В этом случае записывается число тестов, достаточное для того, чтобы все возможные результаты каждого условия в решении выполнялись по крайней мере один раз.

В предыдущем примере имеем четыре условия: {A>1, B=0}, {A=2, X>1}. Следовательно, требуется достаточное число тестов, такое, чтобы реализовать ситуации, где A>1, A<=1, B=0 и B<>0 в точке *a* и A=2, A<>2, X>1 и X<=1 в точке B. Тесты, удовлетворяющие критерию покрытия условий и соответствующие им пути:

а) A=2, B=0, X=4 *ace*

б) A=1, B=1, X=0 *abd*

Таблица 6

Результаты тестирования методом покрытия условий

Тест	*Ожидаемый результат*	*Фактический результат*	*Результат тестирования*
A=2, B=0, X=4	X=3	X=3	неуспешно
A=1, B=1, X=0	X=0	X=1	успешно

Метод комбинаторного покрытия условий

Даныый критерий требует создания такого числа тестов, чтобы все возможные комбинации результатов условия в каждом решении выполнялись по крайней мере один раз. Набор тестов, удовлетворяющих критерию комбинаторного покрытия условий, удовлетворяет также и критериям покрытия решений, покрытия

условий. По этому критерию в рассматриваемом примере должны быть покрыты тестами следующие восемь комбинаций:

1. A>1, B=0;
2. A>1, B<>0;
3. A<=1, B=0;
4. A<=1, B<>0;
5. A=2, X>1;
6. A=2, X<=1;
7. A<>2, X>1;
8. A<>2, X<>1

Для того чтобы протестировать эти комбинации, необязательно использовать все 8 тестов. Фактически они могут быть покрыты четырьмя тестами:

A=2, B=0, X=4

A=2, B=1, X=1

A=0,5, B=0, X=2

A=0.5, B=1, X=1

Таблица 7

Результаты тестирования методом комбинаторного покрытия условий

Тест	*Ожидаемый результат*	*Фактический результат*	*Результат тестирования*
A=2, B=0, X=4	X=3	X=3	неуспешно
A=2, B=1, X=1	X=2	X=1,5	успешно
A=0,5 B=0, X=2	X=3	X=4	успешно
A=0.5, B=1, X=1	X=1	X=1	неуспешно

Приведенные таблицы показывают, что из рассмотренных выше методов тестирования программ, наибольшей способностью обнаруживать ошибки программы, обладает метод комбинаторного покрытия условий.

Рассмотренные выше методы покрытия операторов, покрытия решений (покрытия переходов), покрытия условий и комбинаторного покрытия условий применяются при тестировании логики программного модуля. Как следует из их описания, для применения этих методов на практике структура программы должна быть известной. Естественно, что эти методы должен в первую очередь применять программист, разрабатывающий программу. Если программист ставит своей целью создание надежных программ, он будет для своей программы разрабатывать ‘каверзные тесты’, которые с наибольшей вероятностью могут обнаруживать ошибки программы.

1.9. ОТЛАДКА

Отладкой называется процесс выявления природы ошибки программы и исправления ошибок, после того, как ошибки были обнаружены в процессе тестирования.

Из всех этапов проектирования логики программных модулей этап отладки является наименее формализованным. В нем выделяют две задачи:

1. определение природы ошибки;
2. исправление ошибки.

Решение первой из этих задач занимает около 95 % времени, затрачиваемых на отладку. Поэтому любые средства ускорения процесса определения местоположения ошибки в программ имеют важное значение.

Наиболее распространенными и наименее эффективными для отладки являются так называемые методы ‘грубой силы’. К ним относят:

1. отладку с использованием дампа памяти;
2. отладку с использованием операторов печати по всей программе;
3. отладку с использованием автоматических средств.

Термин *дамп* используется в отношении технической выдачи, не входящей в техническое задание (ТЗ). В частности, многие операционные системы делают *дамп памяти* для программ, работа которых завершается аварийно – создаётся файл, в котором сохраняется область памяти, в которой произошла ошибка, состояние регистров процессораи стека.

Дампы памяти практически невозможно использовать при программировании на языке высокого уровня.

Расстановка печатей в программе бессистемно вряд ли может ускорить отладку. Для системной расстановки нужны четкие представления о структуре алгоритма (схема работы программы или блок-схема). Серьезным недостатком этого метода отладки является тот факт, что печати, вставляемые в программу, изменяют ее, вследствие чего могут послужить дополнительным источником ошибок.

Использование автоматических средств отладки из этой группы методов наиболее предпочтительно.

Общей характеристикой методов ‘грубой силы’ является то, что они не требуют значительных умственных затрат и могут продолжаться бесконечно долго, если наряду с ними не применять более гибкие методы, к которым относятся:

1. метод индукции;
2. метод дедукции.

Название методов напоминают о криминалистике и не напрасно, ибо есть аналогия между этими методами и расследованием преступления.

Метод индукции включает:

1. определение данных тестирования, имеющих отношение к ошибке;
2. анализ от частного к общему позволит выявить закономерности в данных пункта 1;
3. в результате анализа (п.2) выдвигается гипотеза о причине ошибки;
4. для подтверждения гипотезы 3 разрабатывается один или больше тестов, которые должны либо подтвердить, либо опровергнуть гипотезу;
5. если дополнительные тесты подтверждают гипотезу, можно приступать к исправлению ошибки, а вот если не подтверждают, то требуется в лучшем случае возврат к п.3, а в худшем – к п.2.

Альтернативный *метод дедукции* заключается в:

1. перечисление возможных причин или гипотез:
2. использование данных тестирования для исключения некоторых возможных причин;
3. уточнение выбранной наиболее вероятной гипотезы, возможно с использованием дополнительных тестов:
4. доказательство выбранной гипотезы совпадает с п.4 и п.5 метода индукции.

ГЛАВА 2. ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Объектно-ориентированное программирование (ООП) представляет собой новый этап развития современных концепций построения языков программирования. Здесь получили дальнейшее развитие принципы структурного программирования – структуризация программ и данных, модульность и т. д.

В основе ООП лежит понятие **объекта** (object), сочетающего в себе данные и действия над ними. Объект в некотором роде похож на стандартный тип-запись (record), но включает в себя не только поля данных, но также и подпрограммы для обработки этих данных, называемые **методами**. Таким образом, в объекте сосредоточены его свойства и поведение.

ООП характеризуется тремя основными свойствами: **инкапсуляция** (encapsulation), **наследование** (inheritance) и **полиморфизм** (polymorphism).

Инкапсуляция означает упоминавшееся выше объединение в одном объекте данных и действий над ними. Примером может служить перемещаемый по экрану отрезок, определяемый координатами своих концов (данные), и процедурой, обеспечивающей это перемещение (метод).

Наследование позволяет создавать иерархию объектов, начиная с некоторого простого первоначального (предка) и кончая более сложными, но включающими (наследующими) свойства предшествующих элементов (потомки). Эта иерархия в общем случае может иметь довольно сложную древовидную структуру. Каждый потомок несет в себе характеристики своего предка (содержит те же данные и методы), а также обладает собственными характеристиками. При этом наследуемые данные и методы описывать у потомка нет необходимости. В качестве такой иерархии можно рассмотреть точку на экране дисплея, задаваемую своими координатами (предок), отрезок, задаваемый координатами двух точек – его концов (потомок точки), перемещаемый отрезок, задаваемый координатами своих концов и процедурой, обеспечивающей его перемещение (потомок неперемещаемого отрезка) и т. д.

Полиморфизм означает что для различных родственных объектов можно задать единый класс действий (например, перемещение по экрану любой геометрической фигуры). Затем для каждого конкретного объекта составляется своя подпрограмма, выполняющая это действие непосредственно для данного объекта (естественно, что перемещение по экрану точки отличается от перемещения отрезка, а перемещение отрезка, в свою очередь, отличается от перемещения многоугольника и т. д.), причем все эти подпрограммы могут иметь одно и то же имя. Когда потребуется перемещать конкретную фигуру, будет выбрана из всего класса соответствующая подпрограмма.

Может показаться, что сочетание в одном объекте параметров и действий над ними является искусственным объединением. Однако окружающие нас объекты как раз и обладают таким свойством. Взять, например, компьютер. Он состоит из отдельных частей (процессор, монитор, клавиатура и т. д.) и характеризуется рядом параметров (емкость памяти, разрешающая способность дисплея, емкость жесткого диска и т. д.). Все это представляет собой данные рассматриваемого объекта. Кроме этого компьютер может выполнять или над ним можно совершать определенные действия (вставить дискету, поместить точку на экран и т. д.). Так что, действительно, объект – компьютер представляет собой сочетание параметров и действий над ними.

Таким образом, задаваемый объект позволяет локализовать в одном месте его свойства и сделать его в некотором смысле замкнутым по отношению к другим объектам и элементам программы, что, конечно, может в ряде случаев упростить его программирование.

ООП обладает рядом **преимуществ** при создании больших программ. В частности, к ним можно отнести:

- использование более естественных с точки зрения повседневной практики понятий, простота введения новых понятий;
- некоторое сокращение размера программ за счет того, что повторяющиеся (наследуемые) свойства и действия можно не описывать многократно, как это делается при использовании подпрограмм; кроме того, использование динамических объектов позволяет более эффективно использовать оперативную память;
- возможность создания библиотеки объектов;
- сравнительно простая возможность внесения изменений в программу без изменения уже написанных частей, а в ряде случаев и без перекомпиляции этих написанных и уже скомпилированных частей, используя свойства наследования и полиморфизма;
- возможность написания подпрограмм с различными наборами формальных параметров, но имеющих одно и то же имя, используя свойство полиморфизма;
- более четкая локализация свойств и поведения объекта в одном месте (используется свойство инкапсуляции), позволяющая проще разбираться со структурой программы, отлаживать ее, находить ошибки;
- возможность разделения доступа к различным объектам программы и т. д.

Однако следует иметь в виду, что ООП обладает и рядом недостатков и эффективно не во всех случаях. Как правило, использование ООП приводит к уменьшению быстродействия программы, особенно в тех случаях, когда используются виртуальные методы. Неэффективно ООП применительно к

небольшим программам, поэтому его можно рекомендовать при создании больших программ, а лучше даже класса программ. Можно, по-видимому, даже сказать, что ООП скорее не упрощает саму программу, а упрощает технологию ее создания.

2.1. ИНТЕГРИРОВАННАЯ СРЕДА РАЗРАБОТКИ VISUAL STUDIO

Visual Studio (VS) представляет собой интегрированную среду разработки (Integrated Development Environment, IDE). IDE – это набор инструментов разработчика ПО, собранный в составе единого приложения и облегчающий труд программиста при написании приложений. Без IDE (в данном случае – без VS) для написания программы требуется текстовый редактор, с помощью которого программист вводит весь исходный код своей будущей программы. Затем, когда исходный код написан, необходимо запустить из командной строки компилятор, чтобы создать исполняемый файл приложения. Основная проблема, связанная с использованием текстового редактора и компилятора, запускаемого из командной строки, заключается в том, что вы выполняете большое количество ручной работы и теряете при этом много времени.

Автоматически генерируемый код

В состав VS входит целый набор типовых проектов, из которых каждый разработчик может подобрать именно то, что ему в данный момент требуется. Каждый раз, когда вы создаете новый проект, VS автоматически создаст "скелет" будущего приложения, причем этот код можно немедленно скомпилировать и запустить на исполнение.

В составе каждого типового проекта имеются элементы, которые по желанию добавлять в ваш проект. Любой проект, в любом случае, содержит автоматически сгенерированный код, который представляет собой основу будущей программы.

VS предлагает множество готовых к использованию элементов управления, включая и код, необходимый для их создания. Это экономит время разработчиков, избавляя их от необходимости каждый раз заново создавать типовой программный код для решения часто встречающихся задач. Многие из более сложных элементов управления содержат так называемые "программы-мастера" (Wizards), которые помогают настроить поведение элементов управления, автоматически генерируя код в зависимости от выбранных вами опций.

Опыт быстрого кодирования

Редактор VS оптимизирует работу программиста по кодированию. Существенная часть синтаксических элементов новой программы выделяется при помощи системы цветовых обозначений. В ходе того, как вы будете вводить новый код, на экране будут появляться всплывающие подсказки. Наконец, для ускорения выполнения многих задач вам будет предоставлено большое количество клавиатурных комбинаций (keyboard shortcuts).

Все необходимое – всегда под рукой

Вам действительно необходимо научиться ориентироваться в среде VS, чтобы разобраться во всем множестве новых возможностей, призванных упростить нелегкую задачу быстрой разработки высококачественных приложений. Так, окно **Toolbox** просто "до отказа" забито различными элементами управления, окно **Server Explorer** предназначено для работы с сервисами и базами данных операционной системы, а окно **Solution Explorer** дает возможность работать с вашими проектами, тестировать утилиты, и предоставляет средства визуального проектирования. Кроме того, предоставляются и средства работы с компиляторами.

Настраиваемость и расширяемость

Многие из элементов, образующих среду VS, являются настраиваемыми. Это значит, что вы можете менять цвета отображения элементов кода, опции редактора, а также общее оформление. Набор опций настолько обширен, что и в самом деле необходимо потратить некоторое время, чтобы ознакомиться с ними и привыкнуть к тому, где и какую из них следует искать.

При первом запуске вам потребуется выполнить еще один важный конфигурационный шаг – выбрать среду, которую вы хотите открывать по умолчанию.

Выбор настроек среды по умолчанию во многом зависит от того, на каком языке и в какой среде вы в основном будете писать свои программы. Выбор настроек определяет расположение и набор окон, отображаемых по умолчанию, а также стандартные настройки, которые по умолчанию будут использоваться средой VS IDE.

2.2. НАВИГАЦИЯ В СРЕДЕ VISUAL STUDIO

Меню

В верхнем левом углу экрана, показанного на рис. 4, вы увидите начало строки меню (menu bar), начинающейся с пунктов **File**, **Edit**, **View**, **Tools** и т. д. Строка меню представляет собой стандартный элемент интерфейса любого Windows –

приложения. Помимо стандартного управления файлами, меню **File** представляет собой стандартную "начальную точку", откуда вы начинаете работать с любым проектом. Кроме того, меню **File** предоставляет возможности быстрого доступа к недавно открывавшимся файлам и проектам.

Меню **Edit** предоставляет стандартные возможности редактирования: функции вырезки в буфер (cut), копирования (copy) и вставки (paste). Кроме того, оно позволяет установить закладку (bookmark), чтобы обеспечить быстрый доступ к нужному фрагменту кода и быструю навигацию по большим файлам.

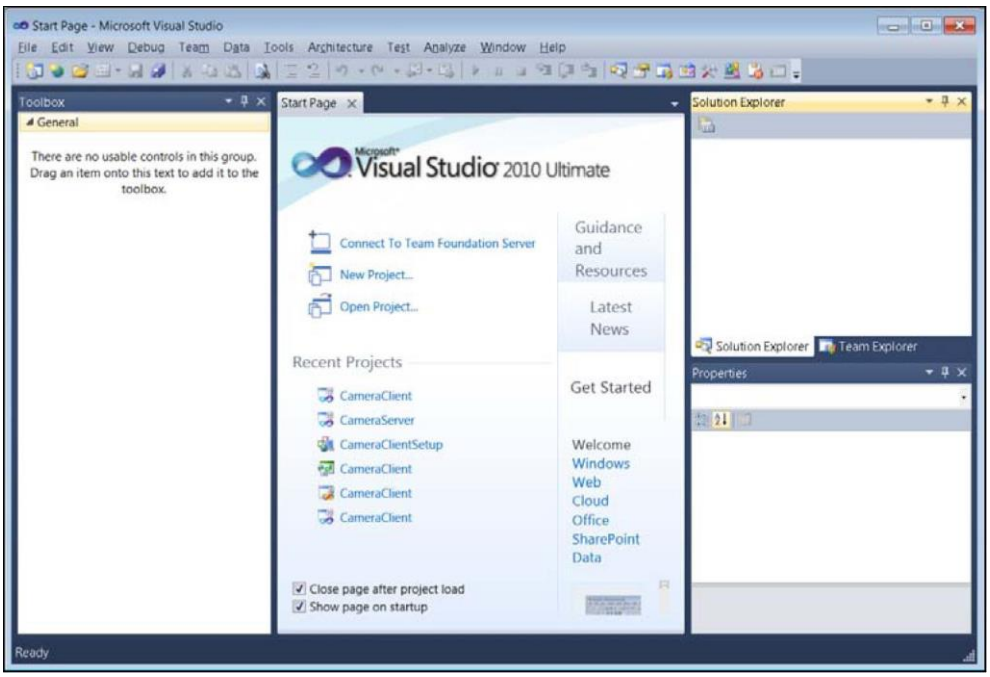


Рисунок 4. Закрепленное окно Toolbox

Не стоит жалеть времени на ознакомление с опциями, доступными через меню **View**, хотя бы для того, чтобы посмотреть, какие функции есть в вашем распоряжении. Но если вы только приступаете к изучению Visual Studio и являетесь начинающим программистом, то не стоит "зацикливаться" на этом сейчас – в дальнейшем мы обратим на эти функции особое внимание, обсудим предназначение каждой из них и объясним, в какой ситуации конкретное представление (view) будет особенно полезным.

Меню **View** позволяет быстро получить доступ ко всем инструментальным окнам, имеющимся в VS. Кроме того, меню **View** содержит пункт **Other Windows**, куда включены дополнительные окна приложений, которые могут оказаться удобными при написании новых программ.

Меню **Tools** предоставляет множество различных функциональных возможностей: например, с его помощью вы можете подключить отладчик, чтобы

просмотреть работу ваших программ пошагово, проходя одну строку за другой, подключиться к базе данных, установить добавочные модули, макросы и выполнить множество других действий. Одна из важнейших опций меню **Tools** так и называется – **Options**, так как она предоставляет доступ к сотням и сотням настроек VS, позволяющих индивидуально настроить среду разработки.

Меню **Test** можно использовать, чтобы выполнить модульное тестирование вашей новой программы, по одному модулю единовременно. Именно в этом меню различные редакции VS предоставляют доступ к различным наборам инструментов тестирования.

Пункты меню **Analyze**, **Architecture** и **Team** предоставляют расширенные наборы инструментов, предназначенных для повышения производительности приложений, работы с их архитектурными компонентами, а также для их интеграции с Microsoft Team Foundation Server.

Меню **Windows** и **Help** имеют предназначение, стандартное для большинства приложений Windows. Так, меню **Windows** позволяет манипулировать различными окнами VS, а с помощью меню **Help** можно читать техническую документацию по VS.

Инструментальная панель (Toolbar)

Под строкой меню на этой иллюстрации находится инструментальная панель. На инструментальной панели расположены кнопки, предоставляющие быстрый доступ к наиболее часто используемым функциям. Набор кнопок инструментальной панели представляет собой подмножество команд, которые доступны через меню. Инструментальные панели контекстно-чувствительны, и набор доступных через них опций меняется в зависимости от той работы, которую вы выполняете через VS в данный момент. Любую инструментальную панель можно отобразить или скрыть. Делается это с помощью команд меню **View | Toolbars**.

Кроме того, инструментальные панели можно настраивать по вашему выбору. Для этого выполните щелчок правой кнопкой мыши по нужной панели, прокрутите появившееся контекстное меню и выберите из него опцию **Customize**. После этого на экране появится окно настройки инструментальной панели, где вы сможете добавить на нее кнопки быстрого вызова функций, которыми часто пользуетесь именно вы, но которые не были включены в набор функций предлагаемых для данной инструментальной панели по умолчанию.

Рабочая область (Work Area)

В центре окна, представленного на иллюстрации, находится стартовая страница (Start page). Это – та самая область, которой вы можете пользоваться для введения кода ваших программ и работы с визуальными редакторами. Стартовая страница разделена на две области: область управления проектами

(project management) и информационную зону (information). Расположенная слева область управления проектами предоставляет возможность быстро начать работу над новым проектом или выбрать из списка один из проектов, с которыми вы работали недавно. Расположенная справа информационная область содержит ресурсы, помогающие новичку начать работу с VS, например, ссылки на Web -сайт Microsoft, пошаговые инструкции, помогающие разобраться с новыми возможностями, а также постоянно обновляемую вкладку, на которой можно прочесть последние новости сообщества разработчиков Microsoft.

Инструментальный набор (Toolbox)

В крайней правой части окна располагается вертикальная вкладка, озаглавленная **Toolbox**, которая содержит контекстно-чувствительный список элементов управления (controls). Эти элементы управления можно перетаскивать мышью в текущую рабочую область, чтобы включить их в состав разрабатываемой программы.

Термин "контекстно-чувствительный" означает, что некоторые элементы списка будут отображены или скрыты, в зависимости от того, где в последний раз был выполнен щелчок мышью, или от контекста, в котором вы в данный момент работаете. Например, вы можете вводить программный код или создавать/редактировать новую Web-страницу. Если вы еще не начали ни над чем работать, то страница **Toolbox** будет пустой.

Окно Solution Explorer

Окно **Solution Explorer**, находящееся на правой границе страницы **Start** отображает все ваши решения (solutions), проекты (projects) и элементы этих проектов. Именно здесь можно найти и требуемым образом организовать все файлы и настройки, принадлежащие проекту. Окно **Solution Explorer** пусто, потому что еще не открыто ни одного решения и не создано ни одного проекта. Если вы закроете это окно, а затем вам потребуется вновь его открыть, то это можно сделать через меню **View**, которое уже обсуждалось чуть ранее.

Строка состояния

В самой нижней части экрана находится строка состояния (status bar), которая сообщает вам обо всем, что происходит в среде VS в настоящий момент. На рис. 1. в строке состояния выведено сообщение **Ready**, которое говорит о том, что среда VS готова к работе. В ходе вашей работы с VS строка состояния будет изменяться в зависимости от контекста, отображая информацию, относящуюся к задаче, выполняемой на текущий момент. Например, если вы работаете с редактором кода, то в строке состояния будет отображаться текущая строка и

позиция, в которой находится курсор, а также другая информация о статусе редактора.

Управление окнами VS

Внимательно взгляните на экран VS, показанный на рис. 1. Обратите внимание, что все окна в рабочей области – **Toolbox**, **Start** и **Solution Explorer** – снабжены строками заголовка (title bars). В строке заголовка окна присутствуют три значка: **Window Position** (треугольная стрелка, направленная вниз), **Maximize/Restore Down** и **Close.** В окне **Solution Explorer**, в строке заголовка эти три кнопки находятся на правой границе. Значок **Window Position** позволяет изменить расположение и состояние окна, предоставляя следующие опции: **Dock**, **Float**, **Dock As Tabbed Document**, **Auto Hide** и **Hide**. Вы можете развернуть окно на всю рабочую область или изменить его размеры и позволить ему свободно перемещаться ("плавать") по всей рабочей области с помощью кнопки **Maximize/Restore Down**.

Если окно пристыковано (docked), то кнопка **Maximize/Restore Down** изменяет свой вид на значок с изображением кнопки, которая позволяет развернуть и свернуть окно. Кнопка **Close** позволяет закрыть окно.

Распахивание и сворачивание окон

Если задержать курсор мыши над вкладкой **Toolbox**, то она развернется и отобразит список из трех значков, доступных и на инструментальной панели окна **Toolbox**: **Window Position** (треугольная стрелка, направленная вниз), **Hide** (кнопка) и **Close** (косой крест). Изображение кнопки **Hide** находится чуть в стороне, и на левой границе по-прежнему присутствует вертикальная вкладка.

Если вы переместите курсор с поля окна **Toolbox**, оно снова свернется и примет вид вкладки, расположенной на левой границе экрана.

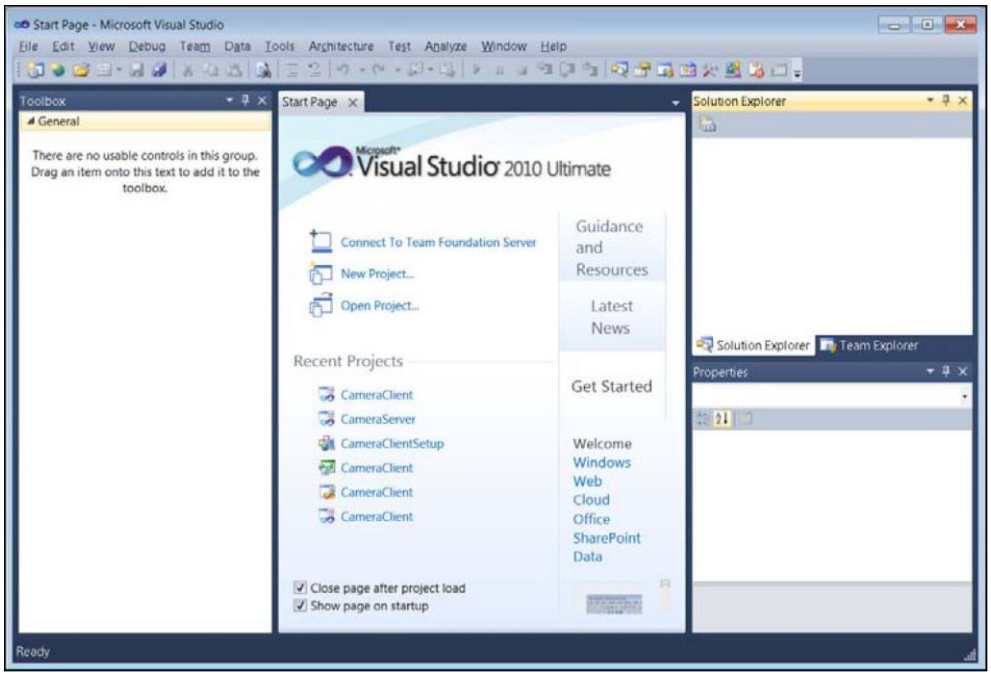


Рисунок 5. Закрепленное окно Toolbox

Любое свернутое окно, например, окно **Toolbox**, можно развернуть, а затем щелкнуть мышью по значку с изображением канцелярской кнопки (**Hide**), в результате чего вид окна станет подобен виду окна **Solution Explorer**. На рис. 2 показано "закрепленное" окно; кнопка на значке **Hide** (над всплывающей подсказкой **Auto Hide**) примет вертикальное положение, а вкладка **Toolbox** на левой границе экрана исчезнет.

Щелчок мышью по значку **Hide** в любом развернутом окне свернет это окно и отобразит вкладку, аналогичную вкладке **Toolbox**. Еще один способ сворачивания окна состоит в том, чтобы выбрать опцию **Auto Hide** из меню **Window Position** (треугольная стрелка, направленная вниз).

Модификация настройки среды после установки

Причины, по которым вам может потребоваться вернуть все настройки среды в тот первозданный вид, который они имеют сразу же после завершения установки, могут быть разными. Например, это может быть простое желание вернуть для всех настроек значения по умолчанию, импортировать общие настройки, созданные другим разработчиком, или же просто переключать настройки, переходя от проекта к проекту.

Запустите VS и выберите из меню команды Tools | Import And Export Settings. На экране появится окно мастера импорта и экспорта настроек (Import and Export Settings Wizard). В окне Import and Export Settings Wizard имеются опции **Export selected environment settings**, **Import selected environment settings** и **Reset all settings**.

Экспорт выбранных параметров настройки среды

Начнем обсуждение с операции экспорта, которая позволит вам предоставить ваши настройки среды в распоряжение других разработчиков. Это также может оказаться полезным, если вы планируете внести в свои настройки существенные изменения и хотите сохранить резервную копию своих настроек, чтобы впоследствии иметь возможность при необходимости быстро к ним вернуться. Чтобы выполнить операцию экспорта, выберите опцию **Export selected environment settings**, а затем нажмите кнопку **Next**. На экране появится окно **Choose Settings to Export**, показанное на рис. 6, в котором вы сможете выбрать экспортируемые настройки.

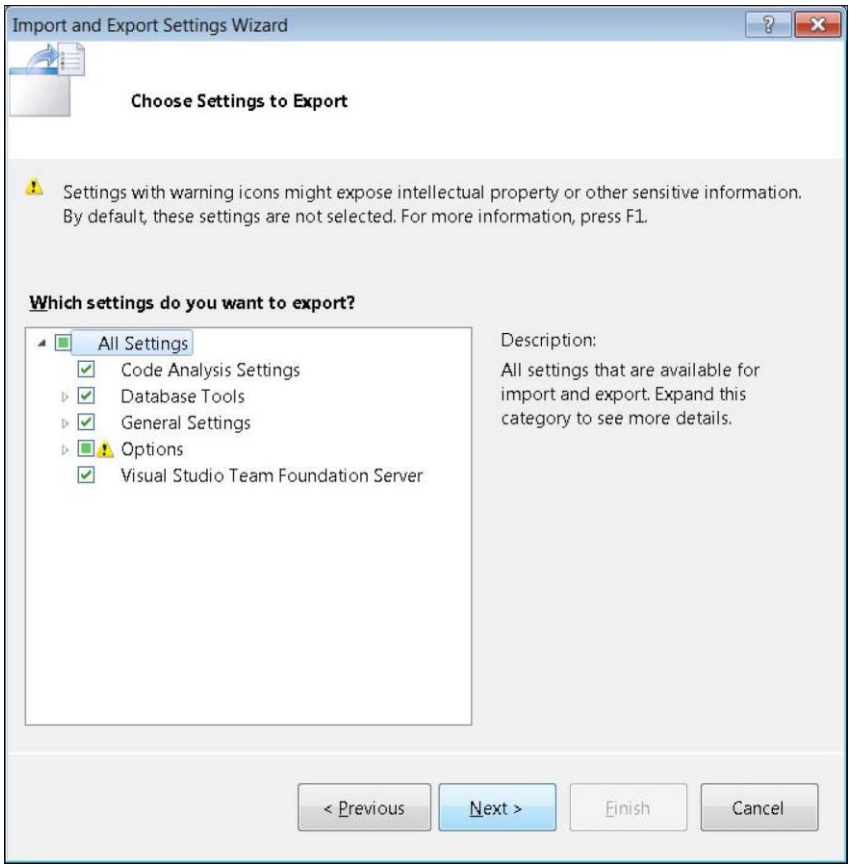


Рисунок 6. Окно Choose Settings to Export

Вы увидите в этом окне целое "дерево" настроек, из которого можно выбрать именно те настройки, которые вам требуется экспортировать.

Значком предупреждения в виде восклицательного знака на фоне желтого треугольника будут помечены настройки, которые не рекомендуются к экспорту по личным мотивам и по соображениям безопасности. Как правило, настройки, помеченные таким значком, имеют отношение к полным путям к системным файлам или таким объектам за пределами среды VS, которые обычно не предоставляются в общий доступ.

После того как вы выберете все подлежащие экспорту настройки, нажмите кнопку Next, и на экране появится окно Name Your Settings File.

Два текстовых поля в окне предназначены для ввода имени файла с экспортированными настройками и пути к каталогу, в котором этот файл должен быть сохранен. Обратите внимание, что стандартное имя файла, предлагаемое по умолчанию, включает текущую дату. Это может оказаться полезным, если вы часто экспортируете настройки – тогда при импорте вы сможете быстро найти нужный файл. Нажмите кнопку Finish, и операция экспорта будет завершена. Об этом вам сигнализирует окно Export Complete.

Щелкните мышью по кнопке Close, чтобы закрыть это окно. Теперь, когда в вашем распоряжении есть экспортированный файл с настройками IDE, вы в любой момент сможете вернуться к сохраненным настройкам, просто импортировав этот файл. Кроме того, данный файл можно предоставить в распоряжение других разработчиков.

Импорт сохраненных настроек среды разработчика

Импорт сохраненных параметров настройки выполняется, когда требуется быстро вернуться к конкретным ранее сохраненным настройкам среды разработчика, импортировать настройки, сохраненные другим разработчиком, или же изменить некоторые параметры для проекта, над которым вы работаете в данный момент.

Чтобы выполнить операцию импорта, откройте среду VS, затем выберите из меню команды Tools | Import and Export Settings. Когда на экране появится окно Import and Export Settings Wizard выберите опцию Import selected environment settings и нажмите кнопку Next. На экране появится окно Save Current Settings.

Диалоговое окно Save Current Settings позволяет предварительно выполнить экспорт (что равносильно созданию резервной копии) ваших текущих настроек, прежде чем приступать и импорту новых. Если вы выполните резервное копирование, то впоследствии вы в любой момент сможете вернуться к заранее сохраненному набору настроек, если что-то в новом наборе вас не устроит.

Вы можете импортировать некоторые из предопределенных настроек, являющихся частью VS. Эти настройки можно найти в составе ветви **Default Settings**. Кроме того, вы можете импортировать и индивидуальные настройки, которые находятся в составе ветви **My Settings**. Индивидуальные настройки включают в свой состав текущие настройки, плюс любые другие настройки, которые вы сохранили в каталоге по умолчанию.

При желании, по вашему выбору вы можете нажать кнопку **Browse** и перейти в каталог, где расположен экспортированный файл с настройками.

Сброс настроек к стандартным значениям

Сброс значений параметров настройки VS к стандартным значениям, предлагаемым по умолчанию, может потребоваться, если вам нужно быстро вернуть среду VS в исходное состояние или если вам приходится переключаться между стандартными настройками VS. Чтобы выполнить переключение к набору стандартных параметров по умолчанию, выберите из меню команды Tools | Import And Export Settings.

Чтобы выполнить переход к стандартным параметрам по умолчанию, выберите в этом окне опцию Reset All Settings и нажмите кнопку Next. На экране появится окно Save Current Settings. Выберите нужную вам опцию сохранения и нажмите кнопку Next. На экране появится окно Choose a Default Collection of Settings.

Знакомство с типами проектов Visual Studio

Visual Studio поддерживает множество типов проектов, что позволяет разработчикам создавать приложения на основе типовых шаблонов.

Чтобы просмотреть, какие типы проектов вы можете создавать, выберите из меню команды **File** | **New** | **Project**.

Вы можете создать не только новый проект, но и новый Web-сайт, открыть файл для редактирования или запустить программу-мастер (wizard), которая создаст новый проект из уже имеющихся файлов. Выберите из меню команды **File** | **New** | **Project**, и на экране появится окно **New Project**, показанное на рис. 4.

Окно **New Project** демонстрирует, что создать можно множество различных проектов, в том числе: **Windows**, **Web**, **Office**, **SharePoint**, **Cloud**, **Reporting**, **Silverlight**, **Test**, **WCF** и **Workflow**. Некоторые из этих типов проектов не перечислены в окне **New Project**, но если вы прокрутите список шаблонов (**Installed Templates**), то вы увидите, что в списке они присутствуют.

Проекты Windows

Если вы выберете опцию **Windows Projects**, то вы увидите длинный список типов проектов приложений Windows, которые вы можете создавать с помощью среды VS. К числу их относятся: настольные приложения (Desktop Applications), включая **Windows Presentation Foundation (WPF)**, **Windows Forms**, и консольные приложения (**Console Applications**). Опция **Console Application** предназначена для построения так называемых консольных приложений, которые запускаются из командной строки и не имеют графического пользовательского интерфейса (Graphical User Interface, GUI). Как правило, эта опция должна выбираться, если вы хотите написать утилиту для администраторов, которую можно использовать при написании административных скриптов, запускаемых из командной строки. Кроме того, данная опция пригодна и для быстрого тестирования вашей программы.

Windows Forms представляет собой старую технологию разработки приложений с графическим пользовательским интерфейсом. Более современная технология разработки приложений с графическим пользовательским интерфейсом основана на платформе .NET и называется WPF (Windows Presentation Foundation).

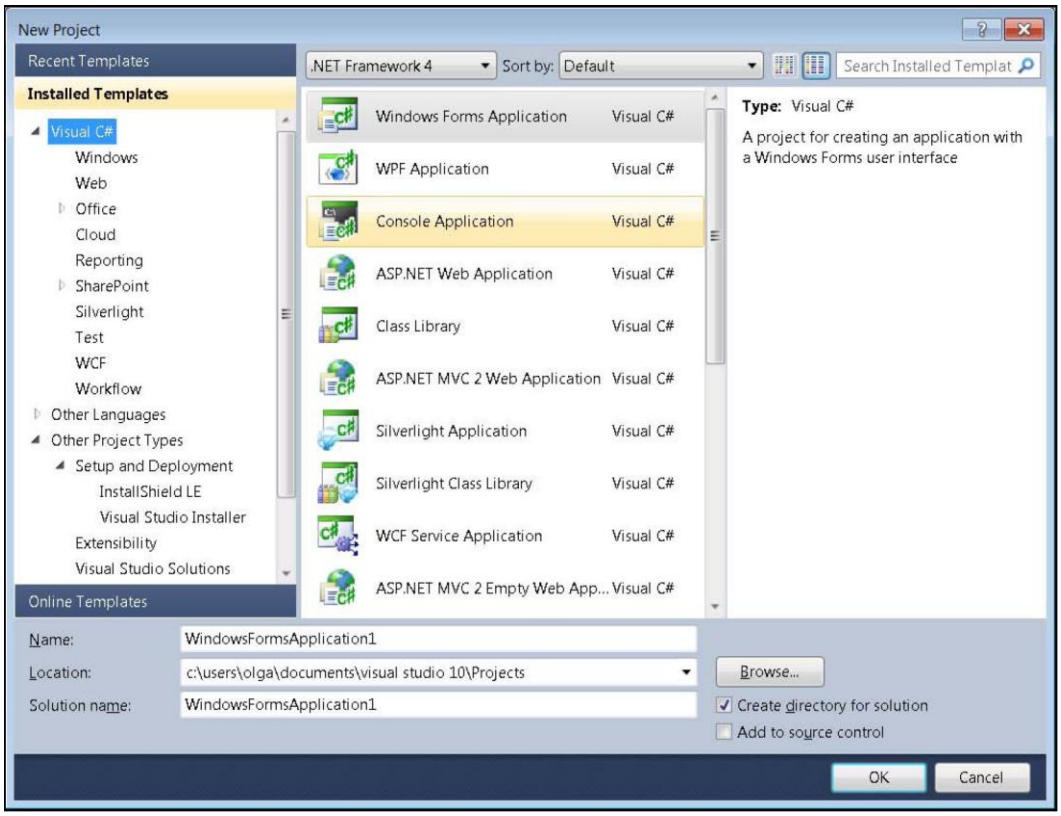


Рисунок 7. Диалоговое окно New Project

К числу остальных проектов Windows относится проект Windows Services. Службы или сервисы Windows (Windows Services) – это программы, которые постоянно работают в фоновом режиме и не нуждаются в графическом пользовательском интерфейсе (GUI). Далее, в числе типов проектов можно обнаружить опцию **Class Libraries**. Библиотеки классов (Class Libraries) предназначены для хранения многократно используемого кода, который часто называют промежуточным (middleware). Наконец, библиотеки элементов управления (Control Libraries) представляют собой библиотеки, которые содержат графические элементы управления. Графические элементы управления можно перетаскивать мышью из окна **Toolbox** в окно визуального редактора графического интерфейса в VS.

Web-проекты

К числу Web-проектов относятся такие, как **ASP.NET**, **Server Controls**, **Web Services** и **Dynamic Data**. Проект ASP.NET позволяет создать приложение, хостинг для которого предоставляется Web-сервером, например, таким, как Internet

Information Server (IIS), и которое работает в Web-браузере. Проекты типа **Server Controls** позволяют создавать библиотеки элементов управления GUI, которые можно перетаскивать в рабочую область визуального проектирования Web-страницы в VS. Компоненты типа **Web Services** можно использовать многократно, вызывая их через Интернет. Важной функцией компонентов типа Web Services является то, что они используют универсальные (ubiquitous) протоколы, которые могут вызываться кодом, работающим на любой платформе, что упрощает интеграцию между разнородными компьютерными системами. Проекты типа **Dynamic Data** предоставляют возможность быстрого создания работающих Web-сайтов, основываясь на существующей схеме базы данных.

Проекты Office

В течение многих лет разработчики пользовались языком Visual Basic for Applications (VBA) для написания программ, автоматизирующих работу с Microsoft Office. Проекты Office позволяют вам автоматизировать приложения Office с помощью платформы .NET, используя для этого такие языки, как VB и C#. К числу поддерживаемых приложений Microsoft Office принадлежат Excel, Word, Project, PowerPoint, Outlook, Visio и InfoPath.

Проекты SharePoint

SharePoint представляет собой технологию, предназначенную для разработки Web-приложений в стиле порталов. Она тесно ассоциируется с приложениями Office, а также координацией и управлением рабочими группами. Чтобы создавать и запускать проекты SharePoint, необходимо, чтобы компьютер, который вы используете для работы с VS, работал под управлением одной из серверных платформ Microsoft, например, Windows Server 2008. SharePoint не работает на таких платформах, как Windows 7, Windows Vista или Windows XP.

Проекты по работе с базами данных (Database Projects)

Проекты для работы с базами данных (Database projects) включают в свой состав проекты SQL Server. Проекты данного типа предлагают возможности тесной интеграции с SQL Server с целью построения такого кода .NET, который будет работать как часть SQL Server. Например, вы можете писать хранимые процедуры (stored procedures) и функции, используя для этого либо C#, либо VB, и использовать в своем коде преимущества инфраструктуры .NET. VS упрощает развертывание кода на SQL Server, фактически сводя его к единственному щелчку мышью.

Вы познакомились с преимуществами среды VS и получили обзорную информацию о том, как эта интегрированная среда разработчика может помочь

вам повысить производительность своей работы, ознакомились с такими ее возможностями, как автоматическая генерация кода, средства быстрой разработки приложений, средства визуального проектирования и расширяемость.. Познакомившись со всеми основными функциями IDE, теперь вы можете запустить VS и закрепить полученные знания и навыки на практике.

Платформа .NET поддерживает несколько различных языков программирования. Так как все эти языки работают на одной и той же платформе и используют одни и те же библиотеки классов, выбор языка, на котором будет вестись разработка, превращается в вопрос личных предпочтений программиста.

Visual Studio (VS) поставляется с поддержкой четырех языков программирования: C#, C++, F# и Visual Basic.NET (VB). Наиболее популярными языками платформы .NET являются C# и VB, и, соответственно, в VS для них и обеспечивается самый высокий уровень поддержки.

Самая серьезная проблема в проектировании больших программ – это *сложность*, *запутанность текстов.* Из-за запутанности программ имеются ошибки, нестыковки и проч. Как следствие – страдает производительность процесса создания программ и их сопровождение. Решение этой проблемы состоит в структуризации программ. Появление объектно-ориентированного программирования связано в большой степени со структуризацией программирования. Мероприятия для обеспечения большей структуризации – это проектирование программы как иерархической структуры, отдельные процедуры, входящие в программу, не должны быть слишком длинными, налагается запрет на использование операторов перехода goto и проч. Современные системы программирования разрешают в названиях переменных использовать *русские буквы.* Между тем современные программисты, как правило, *не используют* данную возможность, хотя когда вдруг в среде англоязычного текста появляются русские слова, это *вносит большую выразительность* в текст программы. Программный код начинает от этого лучше читаться, восприниматься человеком (транслятору, компилятору – все равно).

Глава 3. Язык программирования VISUAL BASIC

3.1 ПРОСТЕЙШИЕ ПРОГРАММЫ С ЭКРАННОЙ ФОРМОЙ И ЭЛЕМЕНТАМИ УПРАВЛЕНИЯ

Пример 1. Форма, кнопка, метка и диалоговое окно

Чтобы запрограммировать какую-либо задачу, необходимо в пункте меню **File** выполнить команду **New Project.** В появившемся окне **New Project** в левой колонке находится список инсталлированных шаблонов **(Installed Templates).** Среди них – шаблоны языков программирования, встроенных в Visual Studio, в том числе Visual Basic, Visual C#, Visual C++, Visual F# и др. Нам нужен язык Visual Basic. В средней колонке выберем шаблон (Templates) **Windows Forms Application** и щелкнем на кнопке **OK.**

В окне изображена *экранная форма* – **Forml,** в которой программисты располагают различные компоненты графического интерфейса пользователя или, как их иначе называют, элементы управления. Это поля для ввода текста **TextBox,** командные кнопки **Button,** строчки текста в форме – метки **Label,** которые не могут быть отредактированы пользователем, и прочие элементы управления.

Причем здесь используется самое современное так называемое *визуальное программирование,* предполагающее простое перетаскивание мышью из панели элементов **Toolbox,** где расположены всевозможные элементы управления, в форму. Таким образом, стараются свести к минимуму непосредственное написание программного кода.

Ваша первая программа будет отображать такую экранную форму, в которой будет что-либо написано, также в форме будет расположена командная кнопка с надписью "Нажми меня". При нажатии кнопки появится диалоговое окно с сообщением "Всем привет!".

Написать такую программку – вопрос 2-3 минут. Но вначале я хотел бы буквально двумя словами объяснить современный объектно-ориентированный подход к программированию. Подход заключается в том, что в программе все, что может быть названо именем существительным, называют ***объектом***. Так в нашей программе мы имеем четыре объекта: форму **Form,** надпись на форме **Label,** кнопку **Button** и диалоговое окно MessageBox с текстом "Всем привет!" (окно с приветом). Итак, добавьте метку и кнопку на форму.

Любой такой объект можно создавать самому, а можно пользоваться готовыми объектами. В данной задаче мы пользуемся готовыми визуальными объектами, которые можно перетаскивать мышью из панели элементов управления **Toolbox.** В

этой задаче нам нужно знать, что каждый объект имеет свойства (properties). Например, свойствами кнопки являются: имя кнопки (Name) – Button1, надпись на кнопке (Text), расположение кнопки (Location) в системе координат формы x, y, размер кнопки size и т. д. Свойств много, их можно увидеть, если щелкнуть правой кнопкой мыши в пределах формы и выбрать в контекстном меню команду Properties.

Указывая мышью на все другие элементы управления в форме, можно просмотреть их свойства: формы Form1 и надписи в форме – метки Label1.

Вернемся к нашей задаче. Для объекта Label1 выберем свойство Text и напишем напротив этого поля "Microsoft Visual Basic 2010" (вместо текста Label1). Для объекта Button1 также в свойстве Text напишем "Нажми меня".

Кроме того, что объекты имеют свойства, следует знать, что объекты обрабатываются событиями. Событием, например, является щелчок на кнопке, щелчок в пределах формы, загрузка (Load) формы в оперативную память при старте программы и проч. В нашей задаче единственным событием, которым мы управляем, является щелчок по командной кнопке. Напомню, что после щелчка на кнопке должно появиться диалоговое окно, в котором написано: "Всем привет!".

Чтобы обработать это событие, необходимо написать всего одну строчку программного кода. Перейдем на вкладку для написания кода: щелчок правой кнопкой мыши в пределах формы, затем выбор команды View Code. Слева вверху мы видим раскрывающийся список, где перечислены объекты, которые присутствуют в данном проекте. Здесь мы выберем Button1 (командную кнопку). Справа вверху находится раскрывающийся список, в котором перечислены все события для кнопки; выбираем событие Click.

При этом управляющая среда VB2010 генерирует две строчки программного кода.

```
Public Class Form1

    Private Sub Button1_Click(ByVal sender As Object,
     ByVal e As System.EventArgs) Handles Button1.Click

    End Sub
End Class
```

Таким образом, система VB2010 написала начало процедуры Sub обработки события Buttoni_Click и конец процедуры End Sub. Эти две строчки называют *пустым обработчиком события.* Заполним этот обработчик. Для этого между этими строчками пишем:

MessageBox.Show("Всем привет!")

Здесь вызывается метод (программа) Show объекта MessageBox с текстом ’’Всем привет!”. Таким образом, объекты кроме свойств имеют также и *методы*, т. е. программы, которые обрабатывают объекты.

В этих трех строчках мы написали *процедуру обработки события* нажатия кнопки (click) Button1. Теперь нажмем клавишу <F5> и проверим работоспособность программы.

Пример 2. Событие MouseHover

Немного усложним предыдущую задачу. Добавим еще одну обработку события MouseHover мыши для объекта Label1. Событие MouseHover наступает тогда, когда пользователь указателем мыши "зависает" над каким-либо объектом, причем именно "зависает", а не просто перемещает мышь над объектом (от англ, *hover* – реять, парить). Есть еще событие MouseEnter (Войти), когда указатель мыши *входит в пределы* области элемента управления (в данном случае метки Labell).

Переключимся на вкладку программного кода **Forml.vb.** Вы видите, что у нас две вкладки: **Forml.vb** и **Forml.vb [Design],** т. е. вкладка программного кода и вкладка визуального проекта программы. Переключаться между ними можно мышью или нажатием комбинации клавиш <Ctrl>+<Tab>.

Итак, запрограммируем событие MouseHover объекта Labell. Для этого в окне редактора кода (где мы сейчас находимся) в левом верхнем раскрывающемся списке выбираем **объект Labell,** а в правом **– событие MouseHover.** При этом среда VB2010 генерирует две строки программного кода (пустой обработчик):

Private Sub Labell_MouseHover *{Параметры процедуры…}*

End Sub

Между этими двумя строчками вставляем вызов диалогового окна:

MessageBox.Show("Событие Hover!")

Теперь проверим возможности программы: нажимаем клавишу <F5>, "зависаем" указателем мыши над Labell, щелкаем на кнопке Button1. Все работает!

Необходимо сказать про *наглядность*, *оперативность*, *технологичность* работы программиста. Посмотрите на свойства каждого объекта в окне **Properties.** Вы видите, как много строчек. Если вы меняете какое-либо свойство, то оно будет выделено жирным шрифтом. Удобно! Но все-таки еще более удобно свойства объектов *назначать* (устанавливать) *в программном коде.* Почему?

Каждый программист имеет в своем арсенале множество уже отлаженных фрагментов, которые он использует в своей очередной новой программе.

Программисту стоит лишь вспомнить, где он программировал ту или иную ситуацию. Программа, которую написал программист, имеет свойство быстро забываться. Если вы посмотрите на строчки кода, которые писали три месяца назад, то будете ощущать, что многое забыли; если прошел год, то вы смотрите на написанную вами программу, как на чужую. Поэтому при написании программ на первое место выходит *понятность*, *ясность*, *очевидность* написанного программного кода. Для этого каждая система программирования имеет какие-либо средства. Кроме того, сам программист придерживается некоторых правил, помогающих ему работать *производительно* и *эффективно.*

Назначать свойства объектов в программном коде удобно при обработке события Form1_Load, т. е. события загрузки формы в оперативную память при старте программы. На вкладке программного кода слева вверху выбираем (Form1 Events), а справа – событие Load. А можно еще быстрей, просто сделать двойной щелчок в пределах формы на вкладке Forml.vb [Design]. При этом управляющая среда генерирует две строчки:

```
Private Sub Forml_Load()

End Sub
```

Между этими двумя строчками обычно вставляют свойства различных объектов и даже часто пишут много строчек программного кода. Здесь мы назначим свойству Text объекта Label1 значение "Microsoft Visual Basic 2010":

```
Label1.Text = "Microsoft Visual Basic 2010"
```

Аналогично для объекта Button1:

```
Buttonl.Text = "Нажми меня!"
```

Совершенно необязательно писать каждую букву приведенных команд. Например, для первой строчки достаточно написать "la" (даже с маленькой буквы), уже это вызовет выпадающее меню, где вы сможете выбрать нужные для данного контекста ключевые слова. Это очень мощное и полезное современное средство редактирования программного кода! Если вы от VB2010 перешли в другую систему программирования, в которой отсутствует данная функция, то будете ощущать сильный дискомфорт.

Вы написали название объекта Label1, поставили точку. Теперь вы видите выпадающее меню, где можете выбрать либо нужное свойство объекта, либо метод (т. е. подпрограмму). В данном случае вы выберите свойство Text.

Как видите, не следует пугаться слишком длинных ключевых слов, длинных названий объектов, свойств, методов, имен переменных. Система подсказок современных систем программирования значительно облегчает всю нетворческую работу. Вот почему в современных программах можно наблюдать такие длинные имена ключевых слов, имен переменных и проч. Потому что на первое место

выходит *ясность, прозрачность программирования*, а громоздкость программ компенсируется системой подсказок.

Далее хотелось бы, чтобы слева вверху формы на синем фоне (в так называемой строке заголовка) была не надпись **Form1**, а что-либо осмысленное. Например, слово "Приветствие". Для этого ниже присваиваем эту строку свойству Text формы. Поскольку мы изменяем свойство объекта Form1 внутри подпрограммы обработки события, связанного с формой, следует к форме обращаться через ссылку Me:

Me.Text = "Приветствие"

ИЛИ

MyBase.Text = "Приветствие".

После написания последней строчки кода мы должны увидеть на экране программный код, показанный в листинге l.l.

Листинг 1.1. Программирование событий

```
Public Class Form1
   Private Sub Form1_Load(ByVal sender As Object,
       ByVal e As System.EventArgs) Handles Me.Load

       Label1.Text = "Microsoft Visual Basic 2010"
       Button1.Text = "Нажми меня!"
       ' Здесь объект Form1 ссылается на себя - Me:
       Me.Text = "Приветствие"
   End Sub

   Private Sub Button1_Click(ByVal sender As Object,
       ByVal e As System.EventArgs) Handles Button1.Click

       MessageBox.Show("Всем привет !")
   End Sub

   Private Sub Label1_MouseHover(ByVal sender As Object,
       ByVal e As System.EventArgs) Handles Label1.MouseHover

       ' Событие Hover(реять, парить) - когда указатель мыши
       ' завис над территорией метки Label1
       MessageBox.Show("Событие Hover !")
   End Sub
End Class
```

Комментарии, поясняющие работу программы, в окне редактора кода будут выделены зелёным цветом, чтобы в тексте выразительно отделять его от прочих элементов программы. В VB комментарий пишут после одиночной кавычки (') или после ключевого слова REM (от англ, *remark* – примечание). Даже если вам кажется весьма очевидным то, что вы пишете в программном коде, *напишите комментарий.* Как показывает опыт, даже очень очевидный замысел программиста забывается удивительно быстро. Человеческая память отметает все, что по оценкам организма считается ненужным.

Посмотрим еще раз на текст программы. Обратите внимание на то, что для улучшения читаемости программы второй параметр процедуры обработки события загрузки формы Form1_Load был перенесен на другую строку. При этом использован символ продолжения (_) для обозначения того, что остаток строки *переносится на следующую строку.* Замечу, что для обозначения переноса строки символ продолжения можно не использовать, если при переносе не разделять параметр процедуры, как это сделано, например, в процедуре обработки события щелчок на кнопке. Здесь запятая после первого параметра означает, что завершение строки нужно искать на следующей строке. В VB2010 существуют также ситуации, когда строку программного кода *можно разрывать*, а символ переноса (_) ставить не обязательно.

Обычно в редакторах программного кода используется моноширинный шрифт, поскольку все символы такого шрифта имеют одинаковую ширину, в том числе точка и прописная русская буква «Ш». По умолчанию в редакторе программного кода VB2010 задан шрифт Consolas. Однако если пользователь привык к шрифту Courier New, то настройку шрифта можно изменить, выбрав меню **Tools | Options | Environment | Fonts and Colors.**

Теперь закроем проект **(File | Close Project).** Система предложит нам сохранить проект, сохраним его под именем **Hover.**

Пример 3. Ввод и вывод в консольном приложении

Иногда, например для научных расчетов, требуется организовать какой-нибудь самый простой ввод данных, выполнить, может быть, весьма сложную математическую обработку введенных данных и оперативно вывести на экран результат вычислений. Такая же ситуация возникает тогда, когда большая программа отлаживается по частям. И для отладки вычислительной части совершенно не важен сервис при вводе данных.

Можно по-разному организовать такую программу, в том числе программируя так называемое *консольное приложение* (от англ. *console* – пульт управления). Под консолью обычно подразумевают экран компьютера и клавиатуру.

Для примера напишем консольное приложение, которое приглашает пользователя ввести два числа, складывает их и выводит результат вычислений **в диалоговое окно.**

Для этого запускаем VB2010, далее создаем новый проект **(New Project),** выбираем шаблон **Console Application.** После двойного щелчка на этом шаблоне попадаем сразу на вкладку программного кода.

Как видите, здесь управляющая среда VB2010 приготовила четыре строки кода. Между Sub Main() и End Sub мы можем вставлять собственный программный код. Sub Main() – это стартовая точка, с которой начинается выполнение программы.

В программе будут участвовать три переменные: X, Y, Z. X и Y вводятся пользователем, далее Z = X + Y и Z выводится в диалоговое окно со словами "Сумма =". Эти переменные следует объявить как single. Тип данных single применяется тогда, когда число, записанное в переменную, может иметь целую и дробную части. Переменная типа single занимает 4 байта.

Объявление этих переменных на VB2010 может иметь такой вид:

```
Dim X, Y, Z As Single
```

При этом в отличие от Visual Basic версии 6 (VB6), все три переменные будут иметь тип Single.

Далее используем объект Console для ввода X:

```
Console.WriteLine("Введите первое слагаемое:")
X = Console.ReadLine()
```

Аналогично для Y:

```
Console.WriteLine("Введите второе слагаемое:")
Y = Console.ReadLine()
```

Далее вычисление Z:

```
Z = X + Y
```

Заметьте, каждую команду (оператор) традиционно пишут с новой строчки. Если вы видите целесообразность написать несколько операторов в одной строке, то после каждого оператора надо ввести двоеточие (:).

После вычисления суммы необходимо вывести Z из оперативной памяти на экран. Для этого воспользуемся форматированным выводом в фигурных скобках метода WriteLine объекта Console:

```
Console.WriteLine("{0} + {1} = {2}", X, Y, Z)
```

Программа написана. Нажмите клавишу <F5>, чтобы увидеть результат.

Вы видите, что все окна появляются на фоне черного окна. Вообще говоря, консольное приложение задумывалось, чтобы программировать в этом черном окне. Вы можете попробовать, если вам это любопытно. Здесь ввод данных организуют через функцию Console.WriteLine (листинг 1.2).

Листинг 1.2. Ввод и вывод данных в консольном приложении

```
Module Modulel
    Sub Main ()
    Dim X, Y, Z As Single

            Console.Title = "Складываю два числа:"
            Console.WriteLine("Введите первое слагаемое:")
            X = Console.ReadLine()
            Console.WriteLine("Введите второе слагаемое:")
            Y = Console.ReadLine()

            Z = X + Y

            Console.WriteLine("{0} + {1} = {2}", X, Y, Z)

            Console.Read()
        ' Чтобы остановить и увидеть все, что на консоли
    End Sub
End Module
```

Такое программирование напоминает период конца 1980-х годов, когда появились первые персональные компьютеры с очень слабой производительностью и небольшой памятью. Поэтому рекомендую пользоваться в консольном приложении не объектом Console, а функциями InputBox и MsgBox.

Для вывода данных Visual Basic, начиная с версии VB.NET, имеет удобное средство MessageBox. Show, однако в консольном приложении его вызвать нельзя. Альтернативой MessageBox. Show может быть функция MsgBox, хорошо знакомая лицам, программировавшим на VB6:

```
MsgBox("Сумма = " & Z)
```

Функция MsgBox осталась в VB2010 и может применяться наряду с MessageBox.Show. Допустимо использовать MsgBox и в консольном приложении. Заменив в консольном приложении все методы объекта Console функциями InputBox и MsgBox, получим такой программный код, который очень напоминает VB6 (листинг 1.3).

Листинг 1.3. Ввод и вывод данных в стиле VB6

```
Module Modulel
    Sub Main ()
        Dim X, Y, Z As Single
        X = InputBox("Введите первое слагаемое:")
        Y = InputBox("Введите второе слагаемое:")
        Z = X + Y
        MsgBox("Сумма = " & Z)
    End Sub
End Module
```

Кстати, в консольном приложении вместо MsgBox можно все-таки использовать MessageBox.Show. Для этого в пункте меню **Project** выбираем команду **Add Reference**, на вкладке **.NET** выбираем строку **System.Windows.Forms**, а затем в программном коде перед Module вставляем строку Imports System.windows.Forms. Ключевое слово Imports используется для импортирования пространства имен, которое содержит класс MessageBox.

При организации научных расчетов или в ситуации, когда необходимо отладить расчетную часть большой программы, когда сервис при вводе данных вообще не имеет значения, можно просто присваивать значения переменным при их объявлении. Очень технологичным является вариант, когда данные записываются в текстовый файл с помощью, например, Блокнота (notepad.exe), а в программе предусмотрено чтение текстового файла в оперативную память.

Пример 4. Проверка типа данных: функция IsNumeric

В предыдущей программе при вводе данных пользователь может ошибочно вводить символы вместо чисел, поэтому целесообразно в этом случае предлагать пользователю ввести дайные еще раз. При этом можно воспользоваться проверкой типа введенных данных с помощью функции IsNumeric (X) и вечным циклом Do *<тело цикла>* Loop. В программе это выглядит таким образом:

```
Do
'Вечный цикл, пока пользователь не введет именно число
X = InputBox("Введите первое слагаемое", "Суммирование")
If IsNumeric(X) = True Then Exit Do
Loop
```

В этом фрагменте операторы, заключенные между Do и Loop, будут выполняться до тех пор, пока функция IsNumeric (X) не вернет значение True (Истина), т. е. введенное значение X является числом. Только в этом случае произойдет выход из цикла Exit Do. Кроме того, обе вводимые переменные следует объявить как String, т. е. как строковые переменные:

```
Dim X, Y As String
```

Заметим, что при введении чисел, имеющих целую и дробную части в качестве разделителя, следует использовать *запятую*, а не точку. Иначе функция IsNumeric будет воспринимать введенные символы как строку, а не число.

Немножко изменим приведенный выше вечный цикл Do...Loop:

```
Do
X = InputBox("Введите первое слагаемое", "Суммирование")
If IsNumeric(X) Then Exit Do
Loop
```

Здесь проверку на выход из цикла мы написали более компактно, словами эту проверку можно описать так: "Если x является числом, то выйти из цикла".

Кроме того, обратите внимание на InputBox. Здесь у этой функции появился еще один параметр после запятой – "Суммирование". Этот текст будет в заголовке окна ввода. Аналогично организуем ввод Y. Законченный программный код представлен в листинге 1.4.

Листинг 1.4. Программный код с проверкой типа

```
Module Module1
  ' Эта программа проверяет числовые ли данные ввел пользователь,
  ' а затем складывает два введенных числа.
  Sub Main()

    Dim X, Y As String
```

```
    Do ' Вечный цикл, пока пользователь не введет именно число:
       X = InputBox("Введите первое слагаемое", "Суммирование")
       If IsNumeric(X) = True Then Exit Do ' - проверка типа
    Loop

    Do
       Y = InputBox("Введите второе слагаемое", "Суммирование")
       If IsNumeric(Y) = True Then Exit Do
    Loop

    Dim Z As Single = Val(X) + Val(Y)
    Z = Convert.ToSingle(X) + Convert.ToSingle(Y)

    ' Возможности конвертирования в VB .NET и выше,
    ' в том числе VB2010:
    ' Z = CType(X, Single) + CType(Y, Single)
    ' Z = Single.Parse(X) + Single.Parse(Y)

    ' Конвертирование в VB6: Z = CSng(X) + CSng(Y)

    MsgBox("Сумма = " & Z, , "Результат:")
  End Sub
End Module
```

В тексте программы после ввода двух чисел в строковые переменные X и Y объявляем переменную Z как Single, в переменную Z будет копироваться сумма введенных чисел. Однако чтобы сложить эти два числа, их необходимо привести (преобразовать) также к типу Single. Для преобразования строковых переменных в VB6 имелись удобные названия функций преобразования: CDbi(X), CSng(X), CInt(X). Они *конвертируют* (от англ. *convert* – преобразовать) строковую переменную X соответственно в переменную типа Double, Single и Integer. В VB6 имелась также функция Val (X), однако она корректно конвертирует строковую переменную в числовой тип данных, если в качестве разделителя целой и дробной частей применяют десятичную точку (не запятую).

Начиная с VB.NET, кроме этих функций в Visual Basic имеется класс Convert, который включает в себя функции преобразования типов. Поэтому используем в нашей программе наиболее современную возможность преобразования Convert.ToSingie(X). Конвертировать переменные одного типа в переменные другого типа можно также и другими функциями, такими как CType, Parse, конкретное

использование этих функций приведено в тексте программы в комментарии (см. листинг 1.4).

Для вывода результата сложения на экран используем функцию MsgBox, здесь она получила еще один параметр для более привлекательного дизайна. Попробуем запустить эту программу, нажав клавишу <F5>.

Если появились ошибки, то работу программы следует проверить отладчиком – клавиши <F8> или <F11>. В этом случае управление останавливается на каждом операторе, и вы можете проверить значение каждой переменной, наводя указатель мыши на переменные. Можно выполнить программу до определенной программистом точки *(точки останова*), используя, например, клавишу <F9> или оператор Stop, и в этой точке проверить значения необходимых переменных.

Пример 5. Ввод данных через текстовое поле TextBox

При работе с формой чаще всего ввод данных организуют через элемент управления текстовое поле TextBox. Напишем типичную программу, которая вводит через текстовое поле число, при нажатии командной кнопки извлекает из него квадратный корень и выводит результат на метку Label. В случае ввода не числа сообщает пользователю об этом, очищая текстовое поле. Есть еще одна кнопка **Очистка** для обнуления текстового поля и метки.

Для этого запускаем VB2010, выбираем пункт меню **File | New Project**, затем – шаблон **Windows Forms Application** и щелкаем на кнопке **OK**. Далее из панели элементов управления **Toolbox** в форму указателем мыши перетаскиваем текстовое поле TextBox1, метку Label1 и две командные кнопки Button1 и Button2. Таким образом, в форме будут находиться четыре элемента управления.

Двойной щелчок в пределах проектируемой формы, и мы попадаем на вкладку программного кода в обработку события Form1_ Load – события загрузки формы. Здесь задаем свойствам формы (к форме обращаемся посредством ссылки Me), кнопкам Button1 и Button2, текстовому полю TextBox1, метке Label1 следующие значения:

```
Me.Text = "Извлечение квадратного корня"
Button1.Text = "Извлечь корень" – Button2.Text = "Очистка"
TextBox1.Clear()' Очистка текстового поля
Label1.Text = ""
Label1.TextAlign = ContentAlignment.MiddleCenter
```

Последняя строка означает выравнивание текста, записанного в Label1.Text, по центру и на середине метки.

Нажмите клавишу <F5> для выявления возможных опечаток, т. е. синтаксических ошибок и предварительного просмотра дизайна будущей программы.

Далее программируем событие Button1_Ciick – щелчок мышью на кнопке **Извлечь корень**. Создать пустой обработчик этого события удобно, дважды щелкнув мышью на этой кнопке. Между двумя появившимися строчками программируем диагностику правильности вводимых данных, конвертирование строковой переменной в переменную типа Single и непосредственное извлечение корня (листинг 1.5).

Листинг 1.5. Извлечение корня

```
' Программа вводит через текстовое поле число, при нажатии на
' командную кнопку извлекает из него квадратный корень и выводит
' результат на метку Label1. В случае ввода не числа сообщает
' пользователю об этом, очищая текстовое поле. Есть еще одна кнопка
' Очистка для обнуления текстового поля и метки.

Public Class Form1
   Private Sub Form1_Load(ByVal sender As System.Object,
                  ByVal e As System.EventArgs) Handles MyBase.Load
      Me.Text = "Извлечение квад. корня"
      Button1.Text = "Извлечь корень"
      Button2.Text = "Очистка"
      TextBox1.Clear() ' – очистка текстового поля
      Label1.Text = ""
      Label1.TextAlign = ContentAlignment.MiddleCenter
   End Sub

   Private Sub Button1_Click(ByVal sender As System.Object,
               ByVal e As System.EventArgs) Handles Button1.Click

      ' Обработка события щелчок на кнопке Извлечь корень:
      If Not IsNumeric(TextBox1.Text) Then
         MessageBox.Show("Следует вводить числа", "Ошибка",
                  MessageBoxButtons.OK, MessageBoxIcon.Error)
         TextBox1.Clear() ' – очистка текстового поля
         TextBox1.Focus() ' - установить фокус на текстовом поле
         Exit Sub
      End If
```

```
    Dim X, Y As Single
    ' Преобразование из строковой переменной в Single:
    X = Convert.ToSingle(TextBox1.Text)
    Y = Math.Sqrt(X)
    Label1.Text = "Корень из " + X.ToString + " равен " + Y.ToString

  End Sub

  Private Sub Button2_Click(ByVal sender As System.Object,
            ByVal e As System.EventArgs) Handles Button2.Click
    ' Обработка события щелчок на кнопке Очистка:
    Label1.Text = ""
    TextBox1.Clear() ' – очистка текстового поля
    TextBox1.Focus()
  End Sub

End Class
```

Здесь при обработке события "щелчок мышью на кнопке **Извлечь корень** проводится проверка, введено ли число в текстовом поле. Проверка осуществляется с помощью функции IsNumeric: если введено не число (например, введены буквы), то выводится диалоговое окно с текстом "Следует вводить числа".

Далее, поскольку ввод неправильный, текстовое поле очищается – TextBoxl. clear(), а фокус передается опять на текстовое поле для ввода числа (т. е. курсор будет находиться в текстовом поле).

Оператор Exit Sub означает выход из программы обработки события Buttonl_Click.

Если пользователь ввел число, то управление не пойдет на ветку If...Then, а будет выполняться следующий оператор Dim – объявление переменных X и Y. Обычно все объявления делают в начале подпрограммы, но удобно объявить переменную там, где она впервые используется.

Далее функция Convert.ToSingle конвертирует строковую переменную TextBoxl.Text в число X, из которого уже можно извлекать квадратный корень Math.Sqrt(X). Математические функции VB2010 являются методами класса Math. Их можно увидеть, набрав Math и поставив точку (.). В выпадающем списке вы увидите множество математических функций: Abs, Sin, Cos, Min и т. д. и два

свойства – две константы E = 2,71... (основание натуральных логарифмов) и PI = 3,14... (число диаметров, уложенных вдоль окружности).

Последней строчкой обработки события Button1_Ciick является присваивание переменной Label1.Text длинного текста. Здесь символ + (можно также & – амперсанд) означает "сцепить" переменные в одну строку.

Нажав клавишу <F5>, проверяем, как работает программа. Заметьте, что при вводе в текстовое поле числа, имеющего целую и дробную части, в качестве разделителя необходимо *ставить запятую*, а не точку. Иначе, введенное в текстовое поле не будет восприниматься как число.

Аналогично обработаем событие Button2_Ciick – очистка текстового поля и метки, а также передача фокуса опять на текстовое поле.

При необходимости используйте отладчик (клавиша <F11>) для *пошагового выполнения программы* и выяснения всех промежуточных значений переменных путем "зависания" указателя мыши над переменными.

Пример 6. Ввод пароля в текстовое поле и изменение шрифта

Это очень маленькая программа для ввода пароля в текстовое поле, причем при вводе вместо вводимых символов некто, "находящийся за спиной пользователя", увидит только звездочки. Программа состоит из формы Form1, текстового поля TextBox1, метки Label1, куда для демонстрации возможностей мы будет копировать пароль (паспорт, т. е. секретные слова), и командной кнопки Button1 – Покажи паспорт.

Перемещаем в форму все названные элементы управления. Текст программы приведен в листинге 1.6.

Листинг 1.6. Ввод пароля

```
' Программа для ввода пароля в текстовое поле, причем при вводе вместо
' вводимых символов, некто, "находящийся за спиной пользователя",
' увидит только звездочки

Public Class Form1

    Private Sub Form1_Load(ByVal sender As System.Object,
                ByVal e As System.EventArgs) Handles MyBase.Load

        Me.Text = "Введи пароль"
```

```
        TextBox1.Text = Nothing
        TextBox1.PasswordChar = "*"
        TextBox1.Font = New System.Drawing.Font("Courier New", 9.0!)
        Label1.Text = ""
        Label1.Font = New System.Drawing.Font("Courier New", 9.0!)
        Button1.Text = "Покажи паспорт"
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object,
                ByVal e As System.EventArgs) Handles Button1.Click

        ' Обработка события щелчок на кнопке:
        Label1.Text = TextBox1.Text
    End Sub
End Class
```

Обрабатываем два события. Первое событие – загрузка формы Form1_Load. Здесь очищаем текстовое поле и делаем его "защищенным от посторонних глаз" с помощью свойства TextBoxl.PasswordChar, каждый введенный пользователем символ маскируется символом звездочки (*). Далее мы хотели бы для большей выразительности и читабельности программы, чтобы вводимые звездочки и результирующий текст имели одинаковую длину. Все символы шрифта Courier New имеют одинаковую ширину, поэтому его называют моноширинным шрифтом. Кстати, используя именно этот шрифт, удобно программировать таблицу благодаря одинаковой ширине букв этого шрифта. Еще одним широко используемым моноширинным шрифтом является шрифт Consola. Задаем шрифт, используя свойство Font обоих объектов: TextBoxl и Label1. Число 9.0 означает размер шрифта.

Осталось обработать событие Button1_Ciick – щелчок на кнопке. Здесь – присваивание текста из поля тексту метки. Программа написана, нажимаем клавишу <F5>.

Пример 7. Управление стилем шрифта с помощью элемента управления CheckBox

Кнопка **CheckBox** (Флажок) также находится на панели элементов управления **Toolbox.** Флажок может быть либо установлен (содержит "галочку"), либо сброшен (пустой). Напишем программу, которая управляет стилем шрифта текста, выведенного на метку **Label.** Управлять стилем будем посредством флажка **CheckBox.**

Используя панель инструментов, в форму поместим метку Label1 и флажок CheckBox1. В листинге 1.7 приведен текст программы управления этими объектами.

Листинг 1.7. Управление стилем шрифта

```
' Программа управляет стилем шрифта текста выведенного на метку
' Label посредством флажка CheckBox

Public Class Form1

   Private Sub Form1_Load(ByVal sender As System.Object,
               ByVal e As System.EventArgs) Handles MyBase.Load

      Me.Text = "Флажок CheckBox"
      CheckBox1.Text = "Полужирный"
      CheckBox1.Focus()
      Label1.Text = "Выбери стиль шрифта"
      Label1.TextAlign = ContentAlignment.MiddleCenter
      Label1.Font = New System.Drawing.
                        Font("Courier New", 14.0!)
   End Sub

   Private Sub CheckBox1_CheckedChanged(ByVal sender _
         As System.Object, ByVal e As System.EventArgs) _
                           Handles CheckBox1.CheckedChanged
      ' Изменение состояния флажка на противоположный :

      If CheckBox1.Checked = True Then Label1.Font =
           New Drawing.Font("Courier New", 14.0!, FontStyle.Bold)

      If CheckBox1.Checked = False Then Label1.Font =
          New Drawing.Font("Courier New", 14.0!, FontStyle.Regular)
   End Sub
End Class
```

При обработке события Form1_Load задаем начальные значения некоторых свойств объектов Form1, Label1 и CheckBoxl. Удобно получить пустой обработчик этого события, дважды щелкнув в пределах формы. Между строчками Private Sub Form1_Load и End Sub присваиваем начальные значения некоторым свойствам и

запускаем методы объектов. Так, тексту флажка, выводимого с правой стороны, присваиваем значение ''Полужирный".

Кроме того, при старте программы фокус должен находиться на флажке (CheckBoxl.Focus()), в этом случае пользователь может изменять установку флажка даже клавишей <Пробел>.

Текст метки – ''Выбери стиль шрифта", выравнивание метки TextAlign задаем посередине и по центру (MiddleCenter) относительно всего того места, что предназначено для метки. Задаем шрифт метки Courier New (в этом шрифте все буквы имеют одинаковую ширину) размером 14 пунктов.

Изменение состояния флажка соответствует событию CheckedChanged. Чтобы получить пустой обработчик события CheckedChanged, следует дважды щелкнуть на элементе CheckBoxl вкладки **Forml.vb [Design].** Между соответствующими строчками следует записать (см. текст программы): если флажок установлен (т. е. содержит ''галочку") Checked = True, то для метки Labell устанавливается тот же шрифт Courier New, 14 пунктов, но Bold, т. е. полужирный. Заметим, что в операторе условия после оператора присваивания (=) символ продолжения строки (_) необязателен.

Далее – следующая строчка кода: если флажок не установлен, т. е. CheckBoxl.Checked=False, то шрифт устанавливается Regular, т.е. обычный. Очень часто эту ситуацию программируют, используя ключевое слово Else (Иначе), однако это выражение будет выглядеть более выразительно и понятно так, как написали мы.

Программа написана, нажмите клавишу <F5>. Проверьте работоспособность программы.

Пример 8. Побитовый оператор Xor

Несколько изменим предыдущую программу в части обработки события checkedchanged (Изменение состояния флажка). Вместо двух условий if.. .Then напишем один оператор:

```
Labell.Font = New System.Drawing.Font("Courier New", 14.0!,
                                      Label1.Font.Style Xor FontStyle.Bold)
```

Здесь каждый раз при изменении состояния флажка значение параметра Labell.Font.Style сравнивается с одним и тем же значением FontStyle.Bold. Поскольку между ними стоит побитовый оператор Xor (Исключающее ИЛИ), он будет назначать Bold, если текущее состояние Labell.Font.Style "не Bold". А если Labell.Font.Style пребывает в состоянии "Bold", то Xor будет назначать состояние "не Bold".

Таблица истинности оператора Xor такова:

A Xor B = C

0 Xor 0 = 0

1 Xor 0 = 1

0 Xor 1 = 1

1 Xor 1 = 0

В нашем случае мы имеем всегда B = l (FontStyie.Bold), а A (Label1.Font.Style) попеременно то Bold, то Regular (т. е. "не Bold"). Таким образом, оператор Xor всегда будет назначать противоположное тому, что записано в Labell. Font.Style.

Как видно, применение побитового оператора привело к существенному уменьшению количества программного кода. Использование побитовых операторов может значительно упростить написание программ со сложной логикой.

Посмотрите, как работает программа, нажав клавишу <F5>.

Теперь добавим в форму еще один элемент управления CheckBox. Мы собираемся управлять стилем шрифта FontStyle двумя флажками. Один, как и прежде, задает полужирный стиль Bold или обычный Regular, а второй задает наклонный italic или возвращает в Regular. Текст новой программы приведен в листинге 1.8.

Листинг 1.8. Усовершенствованный программный код

```
Public Class Form1

    Private Sub Form1_Load(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles MyBase.Load
      Me.Text = "Флажок CheckBox"
      CheckBox1.Text = "Полужирный"
      CheckBox2.Text = "Наклонный"
      Label1.Text = "Выбери стиль шрифта"
      Label1.TextAlign = ContentAlignment.MiddleCenter
      Label1.Font = New System.Drawing.Font("Courier New", 14.0!)
    End Sub

    Private Sub CheckBox1_CheckedChanged(ByVal sender As  _
          System.Object, ByVal e As System.EventArgs) _
                      Handles CheckBox1.CheckedChanged
      Label1.Font = New System.Drawing.Font(
        "Courier New", 14.0!, Label1.Font.Style Xor FontStyle.Bold)
    End Sub
```

```
    Private Sub CheckBox2_CheckedChanged(ByVal sender As  _
            System.Object, ByVal e As System.EventArgs) _
                        Handles CheckBox2.CheckedChanged
        Label1.Font = New System.Drawing.Font( _
            "Courier New", 14.0!, Label1.Font.Style Xor FontStyle.Italic)
    End Sub

End Class
```

Как видно, здесь принципиально ничего нового нет, только лишь добавлена обработка события изменения состояния флажка CheckedChanged для CheckBox2.

Пример 9. Вкладки TabControl и переключатели RadioButton

Вкладки используются для организации управления и оптимального расположения экранного пространства. Выразительным примером использования вкладок является диалоговое окно **Параметры** Microsoft Word. То есть если требуется отобразить большое количество управляемой информации, то весьма уместно использовать вкладки **TabControl**.

Поставим задачу написать программу, позволяющую выбрать текст из двух вариантов, задать цвет и размер шрифта этого текста на трех вкладках **TabControl** с использованием переключателей **RadioButton**.

Программируя поставленную задачу, создадим новый проект **Windows Forms Application,** получим стандартную форму. Затем, используя панель **Toolbox,** в форму перетащим мышью элемент управления **TabControl.** Планируем три вкладки. Для этого в свойствах (окно **Properties)** элемента управления TabControl1 выбираем свойство TabPages, в результате попадаем в диалоговое окно **TabPage Collection Edit,** где добавляем (кнопка **Add)** третью вкладку (первые две присутствуют по умолчанию). Эти вкладки нумеруются от нуля, т. е. третья вкладка будет распознаваться как TabPages(2). Название каждой вкладки укажем при обработке события формы.

Далее для каждой вкладки выбираем из панели **Toolbox** по два переключателя **RadioButton,** а в форму перетаскиваем метку **Label.** В свойстве AutoSize метки Label1 указываем False. Теперь через щелчок правой кнопкой мыши в пределах формы переключаемся на редактирование программного кода.

Текст программы приведен в листинге 1.9.

Листинг 1.9. Использование вкладок и переключателей

```
' Программа, позволяющая выбрать текст из двух вариантов, выбрать
' цвет и размер шрифта для этого текста на трех вкладках TabControl
' с использованием переключателей RadioButton.

Public Class Form1

    Private Sub Form1_Load(ByVal sender As System.Object,
                ByVal e As System.EventArgs) Handles MyBase.Load

        ' Выбирайте улыбки, которые вам ближе
        MyBase.Text = "Какая улыбка Вам ближе"
        TabControl1.TabPages(0).Text = "Текст"
        TabControl1.TabPages(1).Text = "Цвет"
        TabControl1.TabPages(2).Text = "Размер"

        RadioButton1.Text = "Восхищенная, сочувственная, " &
                                    "скромно-смущенная"
        RadioButton2.Text = "Нежная улыбка, ехидная, " &
                    "бесстыжая, подленькая, снисходительная"
        RadioButton3.Text = "Красный"
        RadioButton4.Text = "Синий"
        RadioButton5.Text = "11 пунктов"
        RadioButton6.Text = "13 пунктов"

        Label1.Text = RadioButton1.Text
    End Sub

' Ниже обработки событий изменения состояния шести переключателей:

    Private Sub RadioButton1_CheckedChanged(ByVal sender _
            As System.Object, ByVal e As System.EventArgs) _
                    Handles RadioButton1.CheckedChanged

        Label1.Text = RadioButton1.Text
    End Sub

    Private Sub RadioButton2_CheckedChanged(ByVal sender _
            As System.Object, ByVal e As System.EventArgs) _
```

```
                    Handles RadioButton2.CheckedChanged

        Label1.Text = RadioButton2.Text
    End Sub

    Private Sub RadioButton3_CheckedChanged(ByVal sender _
            As System.Object, ByVal e As System.EventArgs) _
                    Handles RadioButton3.CheckedChanged

        Label1.ForeColor = Color.Red
    End Sub

    Private Sub RadioButton4_CheckedChanged(ByVal sender _
                As System.Object, ByVal e As System.EventArgs) _
                    Handles RadioButton4.CheckedChanged

        Label1.ForeColor = Color.Blue
    End Sub

    Private Sub RadioButton5_CheckedChanged(ByVal sender _
                As System.Object, ByVal e As System.EventArgs) _
                    Handles RadioButton5.CheckedChanged

        Label1.Font = New Font(Label1.Font.Name, 11)
    End Sub

    Private Sub RadioButton6_CheckedChanged(ByVal sender _
                As System.Object, ByVal e As System.EventArgs) _
                    Handles RadioButton6.CheckedChanged

        Label1.Font = New Font(Label1.Font.Name, 13)
    End Sub
End Class
```

Как видно из текста программы, при обработке события загрузки формы Form1_Load указываем название каждой из вкладок и тексты, связанные с каждым переключателем RadioButton. Далее при обработке события изменения состояния переключателей CheckedChanged задаем текст метки Labell, цвет текста метки и размер шрифта.

Пример 10. Свойство Visible и всплывающая подсказка ToolTip

Продемонстрируем возможности свойства visible (Видимый). Программа пишет в метку Label1 некоторый текст, а пользователь с помощью командной кнопки делает этот текст невидимым, а затем опять видимым и т. д. При зависании мыши над кнопкой появляется подсказка ’’Нажми меня”.

Запускаем VB2010, далее выбираем пункты меню **File | New Project | Windows Forms Application** и нажимаем кнопку **OK**. Затем из панели элементов управления **Toolbox** в форму перетаскиваем метку **Label**, кнопку **Button** и всплывающую подсказку **ToolTip**. Только в этом случае каждый элемент управления в форме (включая форму) получает свойство ToolTip on Tip. Убедитесь в этом, посмотрев свойства (окно Properties) элементов управления.

Для кнопки Button1 напротив свойства ToolTip on Tip мы могли бы написать ’’Нажми меня”. Однако я предлагаю написать это непосредственно в программном коде. В этом случае программист не будет долго искать соответствующее свойство, когда станет применять данный фрагмент в своей новой программе!

Далее – щелчок правой кнопкой мыши в пределах формы и выбор команды **View Code**. Окончательный текст программы представлен в листинге 1.10.

Листинг 1.10. Свойство Visible и всплывающая подсказка *ToolTip*

```
' Программа пишет в метку Label некоторый текст, а пользователь с
' помощью командной кнопки делает этот текст либо видимым, либо
' невидимым. Здесь использовано свойство Visible. При зависании мышью
' над кнопкой появляется подсказка "Нажми меня" (Свойство ToolTip).
Public Class Form1

  Private Sub Form1_Load(ByVal sender As System.Object,
                ByVal e As System.EventArgs) Handles MyBase.Load
    Me.Text = "Житейская мудрость"

    Label1.Text =
    "Сколько ребенка не учи хорошим манерам," & vbCrLf &
    "он будет поступать так, как папа с мамой"

    Label1.TextAlign = ContentAlignment.MiddleCenter
    Button1.Text = "Кнопка"
    ToolTip1.SetToolTip(Button1, "Нажми меня")
  End Sub
  ' Обработка события щелчок на кнопке:
  Private Sub Button1_Click(ByVal sender As System.Object,
      ByVal e As System.EventArgs) Handles Button1.Click
```

```
        'If Label1.Visible = True Then
        'Label1.Visible = False
        '    Else
        'Label1.Visible = True
        'End If

        '  Label1.Visible = Label1.Visible Xor True
        Label1.Visible = Not Label1.Visible
    End Sub
End Class
```

При обработке события Form1_Load свойству метки Text присваиваем некоторый текст, ''склеивая" его с помощью амперсанда (&) из отдельных фрагментов. Системная константа vbCrLf начинает текст с новой строки (это так называемый перевод каретки). Эту константу удобно также вызывать из перечисления ControlChars. В этом перечислении можно выбрать и другие управляющие символы. Свойство метки TextAlign располагает текст метки по центру и посередине (MiddleCenter). Выражение, содержащее ToolTipl, устанавливает (Set) текст всплывающей подсказки для кнопки Buttonl при ''зависании" над ней указателя мыши.

Еще одно событие – Button1_Ciick, щелчок мышью на кнопке. Здесь, как видно, закомментированы пять строчек, в которых записана логика включения видимости метки или ее выключение. Логика абсолютно понятна: если свойство видимости (Visible) *включено* (True), то его следует *выключить* (False); иначе (Else) – включить.

Несколько путано, но разобраться можно. И все работает. Проверьте! Кнопку можно нажимать мышью, клавишей <Enter> и клавишей <Пробел>.

Однако можно пойти другим путем. Именно поэтому пять строчек этой сложной логики переведены в комментарий. Мы уже встречались с побитовым оператором Xor (Исключающее ИЛИ). Напоминаю, что этот оператор, говоря кратко, выбирает "да" (True), сравнивая "нет" и "да", и выбирает "нет" (False), сравнивая "да" и "да". Однако можно еще более упростить написание программного кода:

```
Labell.Visible = Not Labell.Visible
```

То есть при очередной передаче управления на эту строчку свойство Labell.visible будет принимать противоположное значение. Вы убедились, что можно по-разному программировать подобные ситуации.

Пример 11. Калькулятор на основе использования комбинированного списка ComboBox

Элемент управления **ComboBox** используется для отображения вариантов выбора в выпадающем списке. Продемонстрируем работу этого элемента управления на примере программы, реализующей функции калькулятора. Здесь для отображения вариантов выбора арифметических операций используется комбинированный список **ComboBox.**

После запуска VB2010 и выбора шаблона **Windows Forms Application** из панели **Toolbox** перетащим в форму два текстовых поля **TextBox,** метку **Label** и комбинированный список **ComboBox.**

Текст программы представлен в листинге 1.11.

Листинг 1.11. Суперкалькулятор

```
' Программа, реализующая функции калькулятора.
' Здесь для отображения вариантов выбора арифметических действий
' используется комбинированный список ComboBox.

Public Class Form1
   Private Sub Form1_Load(ByVal sender As System.Object,
                  ByVal e As System.EventArgs) Handles MyBase.Load

      ComboBox1.Text = "Выбери операцию"
      ComboBox1.Items.AddRange(New Object() {"Прибавить",
         "Отнять", "Умножить", "Разделить", "Очистить"})
      ComboBox1.TabIndex = 2
      TextBox1.Clear() : TextBox2.Clear()
      TextBox1.TabIndex = 0 : TextBox2.TabIndex = 1
      Me.Text = "Супер калькулятор"
      Label1.Text = "Равно: "

   End Sub
   ' Обработка события изменения индекса выбранного элемента:
  Private Sub ComboBox1_SelectedIndexChanged(ByVal sender As System.
           Object, ByVal e As System.EventArgs) Handles ComboBox1.
                                    SelectedIndexChanged
      Label1.Text = "Равно: "
```

```
        If Not IsNumeric(TextBox1.Text) Or
                            Not IsNumeric(TextBox2.Text) Then
            MessageBox.Show("Следует вводить числа!", "Ошибка",
                        MessageBoxButtons.OK, MessageBoxIcon.Error)
            Exit Sub
        End If

        Dim result As Single
        Select Case ComboBox1.SelectedIndex ' Выбор арифмет. операции:

            Case 0 ' - выбрали "Прибавить":
                result = CType(TextBox1.Text, Double) +
                                CType(TextBox2.Text, Double)
                'result = CDbl(TextBox1.Text) + CDbl(TextBox2.Text)
            Case 1 ' - выбрали "Отнять":
                result = CType(TextBox1.Text, Double) -
                                CType(TextBox2.Text, Double)
                'result = CDbl(TextBox1.Text) - CDbl(TextBox2.Text)
            Case 2 ' - выбрали "Умножить":
                result = CType(TextBox1.Text, Double) *
                                CType(TextBox2.Text, Double)
                'result = CDbl(TextBox1.Text) * CDbl(TextBox2.Text)
            Case 3 ' - выбрали "Разделить":
                result = CType(TextBox1.Text, Double) /
                                CType(TextBox2.Text, Double)
                'result = CDbl(TextBox1.Text) / CDbl(TextBox2.Text)
            Case 4 ' - выбрали "Очистить":
                TextBox1.Clear() : TextBox2.Clear()
                Label1.Text = "Равно: " : Exit Sub

        End Select
        Label1.Text = Label1.Text + result.ToString

    End Sub
End Class
```

В этой программе обрабатываем два события: загрузка формы Form1_Load и изменение индекса выбранного элемента ComboBoxl_SelectedIndexChanged.

При загрузке формы присваиваем начальные значения некоторым свойствам, в том числе задаем коллекцию элементов комбинированного списка: "Прибавить", "Отнять" и т. д. Здесь также задаем табличные индексы для текстовых

полей и комбинированного списка TabIndex. Табличный индекс определяет *порядок обхода элементов.* Так, при старте программы фокус будет находиться в первом текстовом поле, поскольку мы назначили TextBox1. TabIndex = 0. Далее при использовании пользователем клавиши <Tab> будет происходить переход от элемента к элементу соответственно табличным индексам.

При обработке события изменение индекса выбранного элемента ComboBoxl_SelectedIndexChanged с помощью функции IsNumeric проверяем, можно ли текстовые поля преобразовать в число. Если хотя бы одно поле невозможно преобразовать в число, то программируем сообщение "Следует вводить числа!" и выход из процедуры обработки события с помощью Exit Sub. Далее оператор select Case осуществляет множественный выбор арифметической операции в зависимости от индекса выбранного элемента списка SeiectedIndex. Для преобразования значений строковых переменных в числовой тип Double автором использована функция CType. В комментарии показана возможность использования функции CDbl для этой же цели.

Заметьте, в этом примере даже не пришлось программировать событие деления на ноль. Система Visual Basic сделала это за нас.

Пример 12. Ссылка на другие ресурсы LinkLabel

Элемент управления **LinkLabel** позволяет *создавать в форме ссылки* на Web-страницы, подобно гиперссылкам в HTML – документах, ссылки на открытие файлов какими-либо программами, ссылки на просмотр содержания логических дисков, папок и проч.

Напишем программу, которая с помощью элемента управления **LinkLabel** обеспечит ссылку для посещения почтового сервера **www.mail.ru,** ссылку для просмотра папки C:\Windows\ и ссылку для запуска текстового редактора Блокнот.

Для программирования этой задачи после запуска VB2010 выберем шаблон **Windows Forms Application,** затем из панели **Toolbox** перетащим в форму три элемента управления **LinkLabel.** Равномерно разместим их в форме. Далее не задаем никаких свойств этим элементам в окне **Properties.** Все начальные значения свойств укажем в программном коде при обработке события загрузки формы Form1_Load (листинг 1.12).

Листинг 1.11. Ссылки на ресурсы

```
' Программа обеспечивает ссылку для посещения почтового сервера
' www.mail.ru, ссылку для просмотра папки C:\Windows\ и ссылку для
' запуска текстового редактора Блокнот с помощью элемента
```

```
' управления LinkLabel

Public Class Form1
   Private Sub Form1_Load(ByVal sender As System.Object,
                  ByVal e As System.EventArgs) Handles MyBase.Load
      Me.Text = "Щелкните по ссылке:"
      LinkLabel1.Text = "www.mail.ru"
      LinkLabel2.Text = "Папка C:\Windows\"
      LinkLabel3.Text = "Вызвать " & ChrW(34) & "Блокнот" & ChrW(34)
      Me.Font = New System.Drawing.Font("Courier New", 12.0!)
      LinkLabel1.LinkVisited = True
      LinkLabel2.LinkVisited = True
      LinkLabel3.LinkVisited = True
   End Sub

   ' Обработка события щелчок на какой-либо ссылке:
   Private Sub Link_Clicked(ByVal sender As System.Object, ByVal e As  _
        System.Windows.Forms.LinkLabelLinkClickedEventArgs) Handles _
                  LinkLabel1.LinkClicked, LinkLabel2.LinkClicked,
                  LinkLabel3.LinkClicked

      Dim s As String = sender.ToString.Substring(38, 1)
      ' - или …Ctype(sender, Button).Text…
      Select Case s ' выбор ссылки:
         Case "w"
            System.Diagnostics.Process.Start(
                        "IExplore.exe", "http://www.mail.ru")
         Case "П"
            System.Diagnostics.Process.Start("C:\Windows\")
         Case "В"
            System.Diagnostics.Process.Start("Notepad", "text.txt")
      End Select
   End Sub
End Class
```

Как видно из программного кода, в свойстве Text каждой из ссылок LinkLabel задаем текст, из которого пользователь поймет назначение каждой ссылки. В задании свойства Text ссылки LinkLabel3 для того, чтобы слово "Блокнот" было в двойных кавычках, используем функцию chrw(34). Эта функция возвращает символ, соответствующий указанному коду (34) символа в *кодировке Unicode.* В данном случае функция chrw(34) возвращает символ "двойная кавычка" (''). Для большей

выразительности задаем шрифт Courier New, 12 пунктов. Поскольку свойство LinkVisited = True, то соответствующая ссылка отображается как уже посещавшаяся (изменяется цвет).

Чтобы обрабатывать событие Click по каждой из ссылок, создадим один пустой обработчик, например, на вкладке Design дважды щелкнув на любой из ссылок. Вообще говоря, здесь требуется программировать три таких обработчика, но в этом случае программный код будет выглядеть *слишком громоздким* (хотя структура кода будет простой). Поступим следующим образом, будем обрабатывать эти три события одной процедурой. Назовем эту процедуру (см. листинг 1.12) Link_Clicked(), а после ключевого слова Handles перечислим через запятую все три события, которые мы хотим обрабатывать в этой процедуре.

Далее, также как и в программе о трех кнопках и калькуляторе, в зависимости от объекта (ссылки), создающего события (LinkLabell, LinkLabel2, LinkLabel3), мы вызываем одну из трех программ: либо Internet Explorer, либо Windows Explorer, либо Блокнот. Информация об объекте, создающем событие Click, записана в объектную переменную sender. Она позволяет распознавать объекты (ссылки), создающие события. Чтобы "вытащить" эту информацию из sender, в строковом представлении (ToString) объектной переменной sender с помощью функции Substring выделим подстроку с параметрами (38, 1). Указанные параметры означают, что следует выделить в подстроку один символ, начиная с 38-й позиции.

На 38-м месте находится первая буква свойства Text каждой из ссылок, например, для первой ссылки www.mail.ru, на 38-м месте находится буква "w”. По первым буквам каждой из ссылок идентифицируем ссылку и с помощью метода Start вызываем либо Internet Explorer, либо Windows Explorer, либо Блокнот. Вторым параметром метода Start является имя ресурса, подлежащее открытию. Именем ресурса может быть или название Web-страницы, или имя текстового файла.

Вызов (запуск) исполняемых файлов помимо метода Start может быть осуществлен также командой Shell, например, таким образом:

```
Shell("notepad C:\textl.txt", AppWinStyle.MaximizedFocus)
```

Пример 13. Греческие буквы, математические операторы. Символы Unicode

Немного ликбеза. Хранение текстовых данных в памяти ЭВМ *предполагает кодирование символов по какому-либо принципу.* Таких кодировок несколько. Каждой кодировке соответствует своя таблица символов. В этой таблице каждой ячейке соответствует номер в таблице и символ. Мы упомянем такие кодовые

таблицы: ASCII, ANSI Cyrillic (другое название этой таблицы Windows 1251), а также Unicode.

Первые две таблицы являются однобайтовыми, т. е. каждому символу соответствует 1 байт данных. Поскольку в 1 байте – 8 битов, байт может принимать 2* = 256 различных состояний, этим состояниям можно поставить в соответствие 256 различных символов. Так в таблице ASCII от 0 до 127 – базовая таблица – есть английские буквы, цифры, знаки препинания, управляющие символы. От 128 до 255 – это расширенная таблица, в ней находятся русские буквы и символы псевдографики. Некоторые из этих символов соответствуют клавишам IBM. Еще эту таблицу называют "ДОСовской" по имени операционной системы DOS, где она используется. Эта кодировка используется также в Интернете.

В операционной системе Windows используется преимущественно ANSI (Windows 1251). Базовые символы с кодами от 0 до 127 в этих таблицах совпадают, а расширенные – нет. То есть русские буквы в этих таблицах находятся в разных местах таблицы. Из-за этого бывают недоразумения. В ANSI нет символов псевдографики. ANSI Cyrillic – другое название кодовой таблицы Windows 1251.

Существует также двухбайтовый стандарт Unicode. Здесь один символ кодируется двумя байтами. Размер такой таблицы кодирования – 2^{16} = 65 536 ячеек. Unicode включает в себя практически все современные письменности. Разве что здесь нет старославянских букв. Когда в текстовом редакторе MS Word мы выполняем команду **Вставка | Символ,** то вставляем символ из таблицы Unicode. Также в Блокноте можно сохранять файлы в кодировке Unicode: **Сохранить как | Кодировка Юникод.** В этом случае в Блокноте будут, например, греческие буквы, математические операторы и др. Размер файла при сохранении в Блокноте будет в два раза больше.

Напишем программу, которая приглашает пользователя ввести радиус *R*, чтобы вычислить длину окружности. При программировании этой задачи длину окружности в метке **Label** называем греческой буквой , приводим формулу для вычислений с греческой буквой = 3.14. Результат вычислений выведем в диалоговое окно MessageBox также с греческой буквой.

После традиционного запуска VB2010 и выбора шаблона **Windows Forms Application** перетащим в форму метку **Label** и текстовое поле **TextBox.**

Вывод греческих букв на метку Label1 и в диалоговое окно MessageBox можно осуществить таким путем. В текст программы через буфер обмена вставляем греческие буквы из текстового редактора MS Word. В этом случае VB.NET требовал ’’Save with UNICODE encoding”, т. е. записи файла Forml.vb в кодировке Unicode. Однако VB2010 сохраняет файлы vb – файлы в формате Unicode по умолчанию.

Более технологично пойти другим путем, а именно вставлять подобные символы с помощью функции chrw, а на вход этой функции подать номер символа в

таблице Unicode. Этот номер легко выяснить, выбрав в редакторе MS Word пункты меню **Вставка | Символ.** Здесь в таблице следует найти этот символ и соответствующий ему код знака в шестнадцатеричном представлении. Чтобы перевести шестнадцатеричное представление в десятичное, следует перед шестнадцатеричным числом поставить &H. Например, после выполнения оператора n = &H3B2 в переменной n будет записано десятичное число 946. На этом месте в таблице Unicode расположена греческая буква .

Именно таким образом мы программировали данную задачу (листинг 1.13).

Листинг 1.13. Использование символов Unicode

```
' Программа демонстрирует возможность вывода в форму, а также в
' диалоговое окно MessageBox греческих букв. Программа приглашает
' пользователя ввести радиус R, чтобы вычислить длину окружности.

Public Class Form1

   Private Sub Form1_Load(ByVal sender As System.Object,
                ByVal e As System.EventArgs) Handles MyBase.Load

      Me.Font = New System.Drawing.Font("Times New Roman", 12.0!)
      Me.Text = "Греческие буквы"
      Label1.Text = "Найдем длину окружности:" &
                vbCrLf & ChrW(&H3B2) & " = 2" & ChrW(&H2219) &
                  ChrW(&H3C0) & ChrW(&H2219) & "R," & vbCrLf &
              "где " & ChrW(&H3C0) & " = " & Math.PI & vbCrLf &
                    vbCrLf & "            Введите радиус R:"
      TextBox1.Clear()
   End Sub
   Private Sub TextBox1_KeyDown(ByVal sender As Object, ByVal e As  _
       System.Windows.Forms.KeyEventArgs) Handles TextBox1.KeyDown

      Dim beta As Decimal
      ' Если пользователь нажал Enter:
      ' If e.KeyCode = Keys.Return Then
      If e.KeyData = Keys.Return Then

         ' Проверка - число ли введено:
         If IsNumeric(TextBox1.Text) = False Then _
                  MessageBox.Show("Вводите число") : Exit Sub
```

```
        beta = 2 * Math.PI * CType(TextBox1.Text, Decimal)

        ' ChrW(&H3B2) - греческая буква бета
        MessageBox.Show("Длина окружности " &
            ChrW(&H3B2) & " = " & String.Format("{0:F4}", beta))

        ' или просто Format(beta, "##0.000")

        ' & String.Format("{1,10:F2} | {2,10:F2} |", X(i), Y(i))
        ' Выражение "10:F2" означает, что переменную X(i) следует
        ' размещать в десяти символах по фиксированному формату
        ' с двумя знаками после запятой.

    End If
  End Sub
End Class
```

Обрабатывая событие Form1_Load, мы задали шрифт Times New Roman, 12 пунктов, и инициализировали свойство Text длинной метки Label1. Различные шестнадцатеричные номера соответствуют греческим буквам и арифметической операции "умножить", в инициализации строки участвует также константа = 3.14. Ее *более* точное значение получаем из Math.PI. Константа vbCrLf означает переход на новую строку, ее можно вызвать из перечисления controlChars, здесь она называется CrLf.

Обрабатывая событие TextBox1_KeyDown (генерируется в момент первоначального нажатия клавиши), мы отслеживаем нажатие клавиши <Enter>, сравнивая данные о нажатой клавише e.KeyData с кодом клавиши <Enter> Keys.Return. Пользователь вводит в текстовое поле значение радиуса и после того как нажимает клавишу <Enter>, мы проверяем с помощью функции IsNumeric, число ли введено в текстовое поле. Если число (True), то вычисляем значение beta, при этом выполняем преобразование строковой переменной TextBox1.Text в переменную типа Decimal (десятичный тип) с помощью функции CType. Десятичный тип представляет восьмибайтовое число (как Double), которое может иметь до 28 десятичных знаков.

После вычисления длины окружности beta выводим ее значение вместе с греческой буквой – ChrW(&H3B2) в диалоговое окно MessageBox. Здесь используем функцию String.Format, выражение "{0:F4}" означает, что значение переменной beta следует выводить по фиксированному формату с *четырьмя знаками* после запятой. В комментарии показано, как можно было бы воспользоваться функцией Format *для той же цели*.

Самостоятельная работа студента по изучению учебной дисциплины

Какое число будет напечатано в результате выполнения фрагмента программы?

```
s:=4;
for i:=1 to 4 do begin
s:=s+2;
for j:=1 to 4 do
s:=s+3;
end;
writeln(s);
```

Сколько раз выполнится цикл?

```
const  b:boolean=false;
a:integer=48;
begin
while not b do
  begin
b:=a<5;
a:=a div 3+2;
  end
end.
```

Задан массив C1 целых чисел (7, 6, -1, 1, 9, -2, -8, 4, -3, 2).
Какое значение будет выведено на экран в результате выполнения фрагмента программы?

```
FOR  m:=10  downto 1 do
if  c1[m]<0  then  b:=c1[m];
FOR  i:=1  to 10 do
if  c1[i]<0  then  if  c1[i]>b  then  b:=c1[i];
writeln(b+c1[5] div c1[9]);
```

Какое значение будет выведено на экран в результате выполнения программы?

```
Const n=7;
Var i,k,a: integer;
B:array[1..n] of integer;
  Begin
For i:=1 to n do  B[i]:=(i+2)*(n-3);
a:=B[B[6]-2*B[2]+2];
k:=0;
For i:=1 to n do if B[i]>2*(a-2) then k:=k+1;
write(5*k);
 End.
```

Какое значение будет напечатано в результате выполнения программы?

```
var x,m:integer;
begin
m:=5;
for x:=-50 to 100 do
begin
if not ((x<-40)or(x>=-10)) or (x>=15)and not(x>=65)
then m:=m+1;
m:=m+1
end;
writeln(m)
end.
```

Какое значение будет напечатано в результате выполнения программы?

```
    type    mass=array[1..3] of integer;
            matr=array[1..3] of mass;
    var     i,j:integer;      b:mass;
            c:array[1..10] of integer;
   const    a:matr=((15,-2,4),(11,-3,9),(5,-6,7));
  begin
        for i:=1 to 3 do begin
                c[i]:=3-i;
                for j:=1 to 3 do c[i]:=c[i]+a[i,j]
                end;
        writeln(c[2])
end.
```

Какое значение будет напечатано в результате выполнения программы?

```
uses crt;
var a,b,c:integer;
procedure prim(var x,a:integer);
 const b:integer=6;
 begin
    clrscr;
    x:=a+b-c; b:=4
 end;
 begin
      a:=2; b:=7; c:=-10;
      prim(c,b);
      writeln(c+b)
 end.
```

В одномерном массиве, состоящем из n вещественных элементов, вычислить:

1) сумму отрицательных элементов массива;

2) произведение элементов массива, расположенных между максимальным и минимальным элементами.

Упорядочить элементы массива по возрастанию.

Примечание. Размерности массивов задаются именованными константами.

В одномерном массиве, состоящем из *n* вещественных элементов, вычислить:

1) сумму положительных элементов массива;

2) произведение элементов массива, расположенных между максимальным по модулю и минимальным по модулю элементами.

Упорядочить элементы массива по убыванию.

Примечание. Размерности массивов задаются именованными константами.

В одномерном массиве, состоящем из n целых элементов, вычислить:

1) произведение элементов массива с четными номерами;

2) сумму элементов массива, расположенных между первым и последним нулевыми элементами.

Преобразовать массив таким образом, чтобы сначала располагались все положительные элементы, а потом – все отрицательные (элементы, равные 0, считать положительными).

Примечание. Размерности массивов задаются именованными константами.

В одномерном массиве, состоящем из *n* вещественных элементов, вычислить:

1) сумму элементов массива с нечетными номерами;

2) сумму элементов массива, расположенных между первым и последним отрицательными элементами.

Сжать массив, удалив из него все элементы, модуль которых не превышает 1. Освободившиеся в конце массива элементы заполнить нулями.

Примечание. Размерности массивов задаются именованными константами.

В одномерном массиве, состоящем из n вещественных элементов, вычислить: 1) максимальный элемент массива;

2) сумму элементов массива, расположенных до последнего положительного элемента.

Сжать массив, удалив из него все элементы, модуль которых находится в интервале [a,b]. Освободившиеся в конце массива элементы заполнить нулями.

Примечание. Размерности массивов задаются именованными константами.

В одномерном массиве, состоящем из *n* вещественных элементов, вычислить:

1) минимальный элемент массива;

2) сумму элементов массива, расположенных между первым и последним положительными элементами.

Преобразовать массив таким образом, чтобы сначала располагались все элементы, равные нулю, а потом – все остальные.

Примечание. Размерности массивов задаются именованными константами.

В одномерном массиве, состоящем из n целых элементов, вычислить:

1) номер максимального элемента массива;

2) произведение элементов массива, расположенных между первым и вторым нулевыми элементами.

Преобразовать массив таким образом, чтобы в первой его половине располагались элементы, стоявшие в нечетных позициях, а во второй половине – элементы, стоявшие в четных позициях.

Примечание. Размерности массивов задаются именованными константами.

В одномерном массиве, состоящем из *n* вещественных элементов, вычислить:

1) максимальный по модулю элемент массива;

2) сумму элементов массива, расположенных между первым и вторым положительными элементами.

Преобразовать массив таким образом, чтобы элементы, равные нулю, располагались после всех остальных.

Примечание. Размерности массивов задаются именованными константами.

В одномерном массиве, состоящем из n целых элементов, вычислить:

1) минимальный по модулю элемент массива;

2) сумму модулей элементов массива, расположенных после первого элемента, равного нулю.

Преобразовать массив таким образом, чтобы в первой его половине располагались элементы, стоявшие в четных позициях, а во второй половине – элементы, стоявшие в нечетных позициях.

Примечание. Размерности массивов задаются именованными константами.

В одномерном массиве, состоящем из *n* вещественных элементов, вычислить:

1) количество элементов массива, значения которых лежат в диапазоне от A до B;

2) сумму элементов массива, расположенных после максимального элемента. Упорядочить элементы массива по убыванию модулей элементов.

Примечание. Размерности массивов задаются именованными константами.

В одномерном массиве, состоящем из n вещественных элементов, вычислить:

1) количество элементов массива, равных 0;

2) сумму элементов массива, расположенных после минимального элемента. Упорядочить элементы массива по возрастанию модулей элементов.

Примечание. Размерности массивов задаются именованными константами.

В одномерном массиве, состоящем из *n* вещественных элементов, вычислить:

1) количество отрицательных элементов массива;

2) сумму модулей элементов массива, расположенных после минимального по модулю элемента.

Заменить все отрицательные элементы массива их квадратами и упорядочить элементы массива по возрастанию.

Примечание. Размерности массивов задаются именованными константами.

В одномерном массиве, состоящем из n вещественных элементов, вычислить:

1) произведение отрицательных элементов массива;

2) сумму положительных элементов массива, расположенных до максимального элемента.

Изменить порядок следования элементов в массиве на обратный.

Примечание. Размерности массивов задаются именованными константами.

Написать программу сортировки элементов одномерного массива по возрастанию их абсолютных значений. Целочисленные элементы массива ввести с клавиатуры. Полученный результат вывести на экран. Размерность массива должна быть задана именованной константой.

Организуйте ввод N слов с клавиатуры в массив, предварительно введя число N. Выведите на экран только те слова, которые содержат более 5 символов.

Организуйте ввод N слов с клавиатуры, предварительно введя число N. Если слово начинается на букву «О», выведите одно сообщение, если нет, то выведите другое сообщение.

Организуйте ввод с клавиатуры значения x (должно быть больше 1). Рассчитайте сумму ряда $y = 1 + 1/x + 1/x^2 + ... + 1/x^n$ *с точностью до* $1/x^n < \varepsilon$ *. Значение* ε *задайте именованной константой.*

Рассчитайте число сочетаний $C_n^k = \frac{n!}{k! \cdot (n-k)!}$. Числа n и k предварительно введите с клавиатуры. Предусмотрите проверки: k, n – целые >0, и $n > k$; если условие не выполняется, повторите ввод.

Организуйте ввод с клавиатуры значений 10-ти элементов статического массива A. Организуйте массив B, такой, что B(i)=A(i)^2 +1, если $A(i) \geq 0$ и $B(i) = 1$, если $A(i) < 0$. Полученные значения элементов массива B выведите на экран.

Организуйте ввод с клавиатуры значений 10-ти элементов статического массива A и 10-ти элементов статического массива B. Организуйте массив C, такой, что C(i)=|A(i) –B(i)|×2 и выведите на экран его элементы.

Организуйте ввод с клавиатуры N элементов массива A (значение N также должно вводиться с клавиатуры). Выведите на экран значения элементов этого же массива, отсортированные по убыванию.

Найти корни квадратного уравнения $y = ax^2 + bx + c$ по формуле $x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$. Значения a, b, c ввести с клавиатуры. Предусмотреть случаи, когда дискриминант больше 0, равен 0 и меньше 0. Вывести на экран соответствующее сообщение с корнями уравнения.

Написать программу, которая изменяет порядок следования элементов массива на обратный, например: {9,8,7,6,1,2,3,4,5} → {5,4,3,2,1,6,7,8,9}. Элементы массива должны быть целого типа; количество элементов массива и их значения должны быть произвольными и введены с клавиатуры; элементы полученного массива должны быть последовательно выведены на экран.

Написать программу, которая загадывает число из заданного диапазона, переходит в режим диалога, в ходе которого просит человека угадать это число, подсказывая больше или меньше ответ, который дал человек. Ввести регулируемое ограничение на количество попыток.

Практические задания для подготовки

1. Реализуйте программу, которая вычисляет для заданного n сумму:

$$1+\frac{1}{2^4}+\frac{1}{3^4}+\ldots+\frac{1}{n^4}.$$

Убедитесь, что при $n \to \infty$ эта сумма сходится к значению $\frac{\pi^4}{90}$.

2. Для заданного n массив заполняется n случайными числами из промежутка от 1 до n. Составить программу, которая подсчитывает количество одинаковых значений в массиве.

3. В заданном массиве найти минимальное значение, максимальное значения и среднее арифметическое элементов массива.

4. Задана функция $y = x - \ln(x)$. Вычислить значения этой функции на отрезке от 0,01 до 10 с шагом 0.5, полученные значения сохранить в файл.

5. Составить программу, которая позволяет численно определить вероятность того, что при подбрасывании трех костей сумма выпавших значений больше 10.

6. Для одномерного массива реализовать циклический сдвиг вправо элементов за заданную длину. Например, если задан массив (1, 2, 3, 4, 5), а длина 3, то получаем массив (4, 5, 1, 2, 3). Элементы полученного массива записать в файл.

7. Вводится две строки из нулей и единиц. В результирующую строку записать сумму двоичных чисел из введенных строк.

8. Заданы два одномерных массива. Постройте массив, содержащий все элементы, которые входят или в первый массив, или во второй, но не в оба вместе. Элементы полученного массива сохранить в файл.

9. Организуйте ввод с клавиатуры значений 10-ти элементов массива A. Выведите на экран сумму элементов, расположенных между минимальным и максимальным элементами массива.

10. Найти сумму 10 членов ряда $1 \cdot 3 \cdot 4 + 2 \cdot 4 \cdot 5 + \ldots$

11. Каждую цифру полученного числа записать в файл.

12. Написать программу, которая по введенному числу находит произведение его цифр.

13. По введенной строке и заданному символу определить встречается ли он в строке заданное число раз

14. Посчитать сколько раз встречается подстрока ‘ab’ в заданной строке

15. В строке состоящей из букв упорядочить символы по возрастанию

Контрольные вопросы

1. Понятие алгоритма, способы записи алгоритма, запись алгоритма с помощью блок-схем, графические символы. Базовые алгоритмические структуры: следование, развилка, выбор, цикл-пока, цикл-до, цикл с параметром.

2. Состав программного обеспечения, понятие жизненного цикла программного обеспечения. Назначение спецификации, основные разделы спецификации на программное обеспечение. Пример разработки спецификации.

3. Понятие «хаотическое» программирование. Принципы структурного программирования. Разработка программного обеспечения сверху вниз (нисходящая стратегия), понятие макетирования.

4. Инструменты обеспечения модульности программы на языке программирования Паскаль. Локальные и глобальные параметры. Процедуры и функции. Понятие формальных и фактических параметров. Параметры значения, параметры переменные, параметры константы.

5. Структура программы на языке Паскаль. Понятие тип данных, простые и структурированные типы данных. Определение нового типа с помощью type. Операторы языка Паскаль: простые и составные операторы. Использование операторов для реализации базовых алгоритмических структур.

6. Понятие переменной: тип, имя и значение переменной. Целые и вещественные типы данных. Числовые выражения. Операции присваивания. Перечисляемый и интервальный тип данных. Простейшие линейные алгоритмы. Операторные скобки begin, end.

7. Понятие массива: размер, индекс и элемент массива. Доступ к элементу массива. Основные алгоритмы для одномерных массивов – поиск минимума (максимума), сумма элементов, поиск заданного элемента, поиск всех элементов, удовлетворяющих некоторому критерию. Понятие многомерного массива, способ задания многомерного массива на языке Паскаль, доступ к его элементам.

8. Строка как составной тип данных. Сравнение и сложение строк. Средства доступа к фрагменту строки. Стандартные алгоритмы работы со строками – замена символов в строке, подсчет числа различных символов, поиск заданного фрагмента строки. Понятие символа, алфавита, кода символа. Таблица кодов ASCII.

9. Понятие множества на языке Паскаль: способ задания, операции сравнения, присваивания, объединения, пересечения, вычитания. Понятие записи: способ задания, доступ к полям записи, оператор присоединения with, записи с вариантами.

10. Файловый тип данных на языке Паскаль: текстовые файлы, типизированные и нетипизированные файлы. Стандартные процедуры и функции работы с файлами: Assign, Reset, Rewrite, Close, Eof, IOResul. Работа с типизированными файлами – запись в файл, чтение из файла, работа с процедурами и функциями: Seek, FileSize, Truncate.

Работа с текстовыми файлами: Append, Read, Readln,Write,Writeln.

11. Понятие рекурсивного алгоритма: шаг рекурсии, базис рекурсии. Реализация рекурсивных алгоритмов на языке Паскаль. Примеры рекурсивных алгоритмов: вычисление факториала числа, чисел Фибоначчи. Понятие подстановочной модели при использовании рекурсии. Линейный и древовидно-рекурсивный процессы. Достоинства и недостатки рекурсии.

12. Понятие динамических структур данных, необходимость их использования. Указатели как особый тип данных в языке Паскаль: понятие базового типа, выделение памяти под базовый тип, освобождение памяти, получение физического адреса переменной базового типа. Реализация динамических структур данных на языке Паскаль с использованием указателей: стек, очередь.

13. Понятие объекта в объектно-ориентированном программировании. Основные принципы объектно-ориентированного программирования: инкапсуляция, наследование, полиморфизм. Задание объекта в языке программирования Visual Basic, доступ к свойствам и методам объекта в программе, реализация наследования. Виртуальные методы и конструкторы. Понятие о таблице виртуальных методов и назначении конструкторов.

14. Понятие тестирования программного обеспечения. Стратегии тестирования «белого ящика». Методы стратегии «белого ящика»: выделение класса эквивалентности: построение тестов, анализ граничных условий. Тестирование модульных программ: восходящее и нисходящее тестирование.

15. Методы стратегии «белового ящика»: метод покрытия операторов, метод покрытия переходов, метод покрытия условий, метод комбинированного покрытия условий. Понятие отладки программного обеспечения, типы ошибок, методы их обнаружения. Понятие альфа-тестирования, бета-тестирования.

Глоссарий

Аргумент- Константа, переменная или выражение, передаваемые в процедуру.

Базовый класс - Исходный класс, от которого наследуются производные классы.

Библиотека динамической компоновки (DLL) - Библиотека подпрограмм загружается или связывается с приложениями во время выполнения.

Ведущее приложение - Любое приложение, поддерживающее работу с Visual Basic для приложений, например, Microsoft Excel, Microsoft Project и т. д.

Время компиляции - Период, в течение которого осуществляется преобразование исходного кода в исполняемый.

Вызов процедуры - Оператор в коде, содержащий инструкцию для выполнения процедуры в Visual Basic.

Выражение - Комбинация ключевых слов, операторов, переменных и констант, которая возвращает строку, число или объект. Выражение можно использовать для выполнения вычислений, обработки текста или проверки данных.

Графический метод - Метод, который будет выполняться на объект, например, формы, PictureBox или принтера и выполняет операции рисунка во время выполнения, такие как анимация или моделирование. Графические методы являются Circle, Cls, строки, PaintPicture, точки, Print и PSet.

Динамический обмен данными (DDE) - Стандартный протокол для обмена данными с использованием активных каналов между приложениями под управлением ОС Microsoft Windows.

Директива компилятора - Команда, задающая действие компилятора.

Документ - Любой изолированный объект, созданный с помощью приложения и обладающий уникальным именем файла.

Дочерняя форма MDI - Форма, содержащаяся в форме MDI в приложении многодокументным интерфейсом (MDI). Чтобы создать дочернюю форму, присвойте свойству MDIChild формы MDI значение True.

Закрепленное окно - Окно, прикрепленное к фрейму главного окна.

Значок - Графическое представление объекта или понятия; обычно используется для представления свернутых приложений в Microsoft Windows. Значок — это растровое изображение размером не более 32 x 32 пикселя. Значки имеют расширение имени файла .ico.

Идентификатор - Элемент выражения, задающий ссылку на константу или переменную.

Исполняемый файл - Приложение Windows, которое может выполняться вне среды разработки. Исполняемый файл с расширением .exe.

Класс - Официальные определения объекта. Класс действует как шаблон, из которого создается экземпляр объекта во время выполнения. Класс определяет свойства объекта и методы, используемые для управления поведением объекта.

Ключевое слово - Слово или символ, распознаваемые как часть языка программирования Visual Basic, например, оператор или имя функции.

Командная строка - Путь, имя файла и аргументы, предоставленные пользователем при запуске программы.

Комментарий - Текст, добавляемый к коду, который объясняет принципы работы кода. В Visual Basic строка комментария можно начать с любого из апостроф (****') или ключевое слово Rem пробел.

Конструктор - Предоставляет окно конструирования в среде разработки Visual Basic. Можно использовать это окно для визуальной разработки новых классов. Visual Basic имеются встроенные окна конструктора для форм. Профессиональный и корпоративные выпуски Visual Basic включают конструкторы для элементов управления ActiveX и документы ActiveX.

Контейнер - Объект, который может содержать другие объекты.

Логическая ошибка - Программная ошибка, которая может привести к неверным результатам или остановить выполнение кода. Например, логическая ошибка может быть вызвана неправильными именами переменных, неправильными типами переменных, бесконечными циклами.

Логическое выражение - Выражение, которое оценивается как True или False.

Массив - Набор последовательно индексируемых элементов, имеющих один внутренний тип данных. Каждый элемент массива имеет уникальный идентификационный номер индекса. Изменения, внесенные в один элемент массива, не влияют на другие элементы.

Метод - Процедура, выполняющая действия в отношении объекта.

Модуль класса - Модуль, в котором содержится определение класса, а также его свойств и методов.

Модуль кода - Модуль, содержащий общий код, который может совместно использоваться всеми модулями в проекте. Модуль кода указывается как стандартный модуль в более поздние версии Visual Basic.

Модуль формы - Файл с расширением FRM в проекте Visual Basic, который содержит графическое описание формы, ее элементы управления и их свойства; объявления констант, переменных и внешних процедур на уровне формы; а также процедуры событий и общие процедуры.

Модуль - Набор объявлений и процедур.

Надстройка - Настраиваемое средство, позволяющее расширить возможности среды разработки Visual Basic.

Область кода - Область в окне кода, который используется для ввода и редактирования кода. Окно кода может содержать один или несколько областей кода.

Обозреватель объектов - Диалоговое окно, в котором вы можете просмотреть содержимое библиотеки объектов и сведения о выбранных объектах.

Общедоступный - Переменные, объявляемые с помощью оператор Public , видимым для всех процедур во всех модулях во всех приложениях, если Option Private Module не действует. В этом случае переменные являются общими только в проекте, в котором они находятся.

Объект ActiveX - Объект, взаимодействующий с другими приложениями или инструментами программирования через интерфейсы автоматизации.

Объект автоматизации - Объект, взаимодействующий с другими приложениями или инструментами программирования через интерфейсы автоматизации.

Объектная переменная - Переменная, содержащая ссылку на объект.

Объявление - Неисполняемый код, имя константа, переменная или процедура, который определяет его характеристики, такие как тип данных. Для процедур DLL объявления укажите имена, библиотеки и аргументы.

Окно объектов - Список в левом верхнем углу окна код, в котором приведены формы и элементов управления в форме, к которому подключен код или поля со списком, расположенный в верхней части окна Свойства, в котором приведены формы и его элементы.

Окно проекта - Окно, в котором отображается список форм, классов и стандартных модулей.

Окно процедур - Список в правом верхнем углу окна кода и окна отладки, которая отображает процедуры, распознанные для объекта в поле "объект ".

Окно свойств - Окно, используемое для отображения или изменения свойств выбранной формы или элемента управления во время разработки. Некоторые дополнительные элементы управления настраиваемых свойств windows.

Оператор - Синтаксически полная конструкция, которое выражает несколько видов действий, описание или определение. Инструкция обычно занимает отдельную строку, хотя двоеточие (****:) можно использовать для включения более одного оператора в строке. Можно также использовать знак объединения строк (_) продолжить одну логическую строку на следующую физическую строку.

Переменная - Именованное расположение для сохранения, которое может содержать данные, которые могут быть изменены во время выполнения программы. Каждая переменная имеет имя, однозначно идентифицирующее в его области. Имена переменных должны начинаться с буквы, должны быть уникальными в заданной области видимости, и, кроме того, не могут содержать более 255 знаков.

Подпрограмма - Процедура, выполняющая конкретную задачу в рамках программы, но не возвращающая значение. Процедура Sub начинается с оператора Sub и заканчивается оператор End Sub .

Пользовательский тип - Любой тип данных, определенных с помощью оператора типа. Типы пользовательских данных могут содержать один или несколько элементов любого типа данных. Массивы определяемых пользователем и других типов данных создаются с помощью оператора Dim.

Проверка синтаксиса - Средство проверки кода на правильность синтаксиса. Если включена функция проверки синтаксиса, сообщение отображается при вводе кода, который содержит ошибку синтаксиса.

Проект - Набор модулей.

Процедура Property - Процедура, которая создает и управляет свойствами в модуле класса. Процедуры Property начинается с помощью оператора Property Let, Property Get или Property Set и заканчивается оператор End Property .

Процедура - Именованные последовательности инструкций. Имя процедуры всегда определяется на уровне модуля. Весь исполняемый код должен содержаться в процедуре. Процедуры не могут быть вложенными в другие процедуры.

Свойство - Именованный атрибут объекта. Свойства определяют характеристики объекта, такие как размер, цвет и расположение на экране или состояние объекта, например, включен или отключен.

Связанное окно - Окно, связанное с любым другим окном, за исключением главного.

Связанный фрейм - Фрейм окна, содержащий несколько связанных друг с другом окон.

Среда разработки - Часть приложения, где осуществляется написание кода, создание элементов управления, свойств форм и т.д. Это отличается от работы приложения.

Стек - Фиксированный объем памяти, используемый средой Visual Basic для хранения локальных переменных и аргументов во время вызова процедуры.

Твип - Единицы измерения экрана, равное 1/20 пункта.

Точка останова - Выделенная строка программы автоматически останавливает выполнение. Точки останова не сохраняются в коде.

Уровень модуля - Описывается код в разделе описаний модуля. Любой код вне процедуры указывается как код уровня модуля.

Файл ресурсов - Файл в проекте Visual Basic с расширением имени файла .res, который может содержать точечные рисунки, текстовые строки или другие данные. Только один файл ресурсов может быть связан с проектом.

Форма MDI - Окно, которое является фоновым приложения многодокументным интерфейсом (MDI).Форма MDI является контейнером для любых дочерних форм MDI в приложении.

Форма - В поле окна или диалогового окна. Формы являются контейнерами для элементов управления. Форма многодокументным интерфейсом (MDI) также может выступать в качестве контейнера для дочерних форм и некоторых элементов управления.

Функция - Процедура, выполняющая конкретную задачу в рамках программы и возвращающая значение. Функция начинается с оператор Function и заканчивается на оператор End Function .

Элемент ActiveX - Объект, который можно направлять в форме для включения или улучшения взаимодействия пользователей с приложением. Элементы управления ActiveX, связанные с событиями и могут включаться в другие элементы управления. Эти элементы управления имеют входить в состав.

Элемент управления - Объект, который можно поместить на форме, которая содержит собственный набор распознаваемых свойств, методов и событий. Использование элементов управления для ввода данных пользователем, отображения выходных данных и запуска процедур обработки событий. Можно управлять с помощью методов большинства элементов управления.

Библиографический список

1. Turbo Pascal Учебное пособие. (Серия:"Учебное пособие") (ГРИФ) / Фаронов В.В. Изд-во Питер – 2007.
2. Turbo Pascal. Практикум 2-е изд. (Серия:"Учебное пособие") (ГРИФ) / Немнюгин С.А. Изд-во Питер – 2007.
3. Turbo Pascal. Программирование на языке высокого уровня Учебник для вузов. 2-е изд. (Серия:"Учебник для вузов") (ГРИФ) / Немнюгин С.А. Изд-во Питер – 2007.
4. Артамонов Ю.Н., Елизаров В.С., Новиков А.Н., Пронькин Н.Н. Технология программирования. Учебно-методический комплекс для направления 230200.62 "Информационные системы". – Московский городской университет управления Правительства Москвы. Москва, 2009.
5. Гапоненко В.Ф., Пронькин Н.Н. и др. Основы теории управления. Учебно-методический комплекс. – Московский городской университет управления Правительства Москвы. Москва, 2011.
6. Гапоненко В.Ф., Пронькин Н.Н. и др. Проектирование информационных систем в управлении. Учебно-методический комплекс. – Московский городской университет управления Правительства Москвы. Москва, 2011.
7. Глущенко В.М., Гапоненко В.Ф., Елизаров В.С., Пронькин Н.Н. Компьютерные технологии в экологии и природопользовании. Учебно-методический комплекс для направления 020800.62 "Экология и природопользование". – Московский городской университет управления Правительства Москвы. Москва, 2010.
8. Глущенко В.М., Гапоненко В.Ф., Елизаров В.С., Пронькин Н.Н. Экоинформатика: основы и перспективы развития. Учебно-методический комплекс для направления 020800.62 "Экология и природопользование". – Московский городской университет управления Правительства Москвы. Москва, 2010.
9. Глущенко В.М., Елизаров В.С., Каманин И.О., Пронькин Н.Н. Информатика. Учебно-методический комплекс для направления 230200.62 "Информационные системы". – Московский городской университет управления Правительства Москвы. Москва, 2009.
10. Глущенко В.М., Елизаров В.С., Каманин И.О., Пронькин Н.Н. Современные офисные технологии. Учебно-методический комплекс для направления 230200.62 "Информационные системы". – Московский городской университет управления Правительства Москвы. Москва, 2009.
11. Глущенко В.М., Елизаров В.С., Новиков А.Н., Пронькин Н.Н. Информационные технологии в управленческой деятельности. Учебное пособие для государственных гражданских служащих г. Москвы, обучающихся по образовательной программе повышения квалификации. – Московский городской университет управления Правительства Москвы. Москва, 2010.

12. Глущенко В.М., Елизаров В.С., Новиков А.Н., Пронькин Н.Н. Информационно-справочные системы. Учебно-методический комплекс для студентов специальности 100103.65 "Социально-культурный сервис и туризм". – Московский городской университет управления Правительства Москвы. Москва, 2009.

13. Глущенко В.М., Елизаров В.С., Новиков А.Н., Пронькин Н.Н. Теория систем и системный анализ. Учебно-методический комплекс. – Московский городской университет управления Правительства Москвы. Москва, 2011.

14. Глущенко В.М., Пронькин Н.Н. и др. Информационное обеспечение размещения государственных и муниципальных закупок. Пособие для государственных гражданских служащих г. Москвы, обучающихся по образовательной программе профессиональной переподготовки "Управление государственными и муниципальными заказами" / В.М. Глущенко [и др.]. Москва, 2011.

15. Глущенко В.М., Пронькин Н.Н. и др. Информационное обеспечение размещения государственных и муниципальных закупок. Учебно-методический комплекс для государственных гражданских служащих г. Москвы, обучающихся по образовательной программе профессиональной переподготовки "Управление государственными и муниципальными заказами" / В.М. Глущенко [и др.]. Москва, 2011.

16. Глущенко В.М., Пронькин Н.Н. и др. Информационные системы и технологии. Учебник. – Московский городской университет управления Правительства Москвы. Москва, 2012.

17. Глущенко В.М., Пронькин Н.Н. и др. Моделирование систем. Учебно-методический комплекс. – Московский городской университет управления Правительства Москвы. Москва, 2011.

18. Глущенко В.М., Пронькин Н.Н. и др. Теория информационных процессов и систем. Учебно-методический комплекс. – Московский городской университет управления Правительства Москвы. Москва, 2011.

19. Глущенко В.М., Пронькин Н.Н. и др. Экономическая безопасность в сфере государственных и муниципальных закупок. Пособие для государственных гражданских служащих г. Москвы, обучающихся по образовательной программе профессиональной переподготовки "Управление государственными и муниципальными заказами" / В. М. Глущенко [и др.]. Москва, 2011.

20. Глущенко В.М., Пронькин Н.Н. и др. Экономическая безопасность в сфере государственных и муниципальных закупок. Учебно-методический комплекс для государственных гражданских служащих г. Москвы, обучающихся по образовательной программе профессиональной переподготовки "Управление государственными и муниципальными заказами" / В. М. Глущенко [и др.]. Москва, 2011.

21. Елизаров В.С., Ковалева Е.Д., Пронькин Н.Н. Математика и информатика. Учебно-методический комплекс. – Московский городской университет управления Правительства Москвы. Москва, 2009.

22. Елизаров В.С., Малышев М.Н., Пронькин Н.Н. Информационные технологии в социальной работе. Учебно-методический комплекс. – Московский городской университет управления Правительства Москвы. Москва, 2011.

23. Елизаров В.С., Прудкин В.Е., Пронькин Н.Н. Компьютерная геометрия и графика. Учебно-методический комплекс. – Московский городской университет управления Правительства Москвы. Москва, 2009.

24. Зиборов В. В. Visual Basic 2010 на примерах. — СПб.: БХВ-Петербург, 2010. — 336 с.: ил. + CD-ROM

25. Инновационное развитие системы высшего профессионального образования в условиях социально-экономической модернизации страны. Монография / В.М. Глущенко, В.С. Елизаров, И.В. Ирошин, В.М. Крашенинников, К.М. Ипполитова, В.В. Наумов, Н.Н. Пронькин; под ред. В.М. Глущенко. – М.: Московский городской университет управления Правительства Москвы, 2010.

26. Инструментарий формирования государственного оборонного заказа: монография. – М.: Моск. городск. ун-т управления Правительства Москвы, 2011.

27. Кадровая стратегия Москвы: теория и методы обоснования структур исполнительной власти. Монография / А.К. Алексеев, В.М. Глущенко, В.С. Елизаров, В.М. Крашенинников, А.И. Прокофьев, Н.Н. Пронькин; под ред. В.М. Глущенко. – М.: Московский городской университет управления Правительства Москвы, 2011.

28. Кудрявцев А.С., Пронькин Н.Н. и др. Информационные системы в управлении городским хозяйством. Учебно-методический комплекс для направления 230200.62 "Информационные системы" очной формы обучения высшего профессионального образования. – Московский городской университет управления Правительства Москвы. Москва, 2011.

29. Любина О.Н., Пронькин Н.Н. Программы MICROSOFT WORD И MICROSOFT EXCEL: основные возможности и их использование на государственной службе города Москвы. Учебно-методический комплекс для государственных гражданских служащих города Москвы, обучающихся по образовательным программам повышения квалификации. – Московский городской университет управления Правительства Москвы. Москва, 2011.

30. Майо Дж. Самоучитель Microsoft Visual Studio 2010. — СПб.: БХВ-Петербург, 2011. — 464 с.: ил.

31. Москва как система: историко-методологические проблемы. Монография / В.М. Глущенко, А.Н. Новиков, Н.Н. Пронькин, Г.Ф. Шилова; под ред. В.М. Глущенко. Изд. 2-е, переработ. – М.: Московский городской университет управления Правительства Москвы, 2012.

32. Московский мегаполис: системный анализ, междисциплинарный подход, информационные технологии управления. Монография / В.М. Глущенко,

Н.Н. Пронькин, Г.Ф. Шилова и др.; под ред. В.М. Глущенко. – М.: Московский городской университет управления Правительства Москвы, 2012.

33. Новиков А.Н., Пронькин Н.Н. Информационное обеспечение процесса управления финансами предприятия авиационно-промышленного комплекса. – Вестник Московского авиационного института. 2009. Т. 16. № 6. С. 25.

34. Новиков А.Н., Пронькин Н.Н. Определение направлений процесса формирования государственного оборонного заказа. – Труды МАИ. 2010. № 37. С. 26.

35. Программирование в C++Builder 6 и 2006 / Архангельский А.Я., Тагин М.А. М.: Бином-Пресс – 2007

36. Программирование в Delphi Уч. по кл. версиям Delphi. / Архангельский А.Я. М.: Бином-Пресс – 2006

37. Пронькин Н.Н. Архитектура ЭВМ и систем. Учебно-методический комплекс для направления 230200.62 "Информационные системы". – Московский городской университет управления Правительства Москвы. Москва, 2011.

38. Пронькин Н.Н. Информатика. Учебно-методический комплекс для направления 020800.62 "Экология и природопользование" – Московский городской университет управления Правительства Москвы. Москва, 2011.

39. Пронькин Н.Н. Информатика. Учебно-методический комплекс для специальности 071401.65 " Социально-культурная деятельность ". – Московский городской университет управления Правительства Москвы. Москва, 2012.

40. Пронькин Н.Н. Экономико-математический и программный инструментарий формирования государственного оборон ного заказа. Канд. Дисс. – Московский городской университет управления Правительства Москвы. Москва, 2010.

41. Пронькин Н.Н. Экономико-математический и программный инструментарий формирования государственного оборонного заказа. Автореферат диссертации на соискание ученой степени кандидата экономических наук / Моск. гос. ун-т управления. Москва, 2010.

42. Российская социальная система: оглянуться в ушедшее время. Ради будущего. (Синергетика в гуманитарных науках). Под ред. профессора Старостенкова Н.В.: Монография. / Ляпунова Н.В., Потехина Е.В., Пронькин Н.Н., Старостенков Н.В., Шилова Г.Ф.– М.: Экслибрис-Пресс, 2017. – 152 с.

43. С.А. Орлов Технологии разработки программного обеспечения. Разработка сложных программных систем – СПб, Питер, 2003.

44. Технология программирования (Серия:"Информационные технологии от первого лица ") (ГРИФ) / Терехов А.Н. М.: БИНОМ. Лаб. зн., ИНТУИТ.РУ – 2006.

45. Технология программирования Учебник для вузов. 3-е изд.,перераб. и доп. (Серия:"Информатика в техническом университете") (ГРИФ) / Иванова Г.С. Изд-во МГТУ им. Н.Э. Баумана – 2006 Изд-во МГТУ им. Н.Э. Баумана – 2006.

ISBN 978-5-6044576-2-7

Усл. печ. л. 5,6.
Объем издания 4,7 MB
Оформление электронного издания:
НОО Профессиональная наука, mail@scipro.ru
Дата размещения: 10.05.2020 г.
URL: http://scipro.ru/conf/algorithmiclanguages.pdf