Microsoft®

Основы программирования на примере Visual Basic®.NET

УЧЕБНОЕ ПОСОБИЕ

2005

Дорогие ученики!

Это учебное пособие разработано всемирно известной корпорацией Майкрософт, мировым лидером в производстве программного обеспечения, в рамках инициативы «Партнерство в образовании». Задача инициативы — предоставить школам возможность повышения качества обучения за счет использования информационных технологий в учебном процессе.

Информационные технологии обладают достаточным потенциалом для того, чтобы дать вам возможность приобрести навыки, которые будут необходимы в будущей работе, — навыки эффективной обработки информации и управления ею, навыки общения и совместной (групповой) работы. Для развития этих и других навыков, получивших название «навыки XXI века», и служат разработанные Майкрософт учебные пособия.

Участие Майкрософт в создании этих учебников обеспечивает высокое качество учебных пособий и передачу экспертного знания, накопленного внутри корпорации, непосредственно учителю и ученику. Некоторые учебные курсы Майкрософт серии «Партнерство в образовании» предназначены для подготовки грамотных пользователей персональных компьютеров, другие курсы предназначены для ребят, более глубоко интересующихся информационными технологиями и желающих стать специалистами в этой области.

Учебные курсы Майкрософт серии «Партнерство в образовании» переведены на языки многих стран мира и успешно используются во многих странах.

Вы держите в руках учебное пособие **«Основы программирования на примере Visual Basic .NET»**, которое поможет вам погрузиться в увлекательный мир объектно-ориентированного программирования и почувствовать себя творцом, способным создавать интересные программы. Это учебное пособие позволит вам лучше понять работу программиста.

Среди других учебников, разработанных компанией Майкрософт, вы найдете следующие:

- **Учебные проекты с использованием Microsoft Office.** Курс предполагает выполнение различных увлекательных проектов, знакомящих учеников с некоторыми, ранее неизвестными областями деятельности (например, с такими, как основы маркетинга, грамотное составление резюме и поиск работы, оптимальные подходы к совершению покупок и др.).
- **Основы компьютерных сетей.** Курс знакомит с основами грамотного построения и поддержки компьютерных сетей и Интернета, помогая учащимся приобрести знания и навыки, востребованные в сегодняшнем высокотехнологичном обществе.
- **Персональный компьютер: настройка и техническая поддержка.** Курс дает необходимую теоретическую и практическую подготовку для работы в качестве специалистов службы технической поддержки. Программа курса включает обучение ремонту и настройке компьютеров, базам данных и основам работы служб технической помощи.

Мы желаем вам успехов на пути обретения новых знаний и будем рады, если вам понравится наш курс! О своих впечатлениях об этом учебном курсе вы можете рассказать нам, написав по электронной почте на адрес: mailto:russia@microsoft.com

С самыми наилучшими пожеланиями,
сотрудники российского Представительства Майкрософт
http://www.microsoft.com/rus

Рекомендации по использованию учебно-программного комплекса

1. Учебно-программный комплекс по курсу «Основы программирования на примере Visual Basic .NET» позволяет научиться разрабатывать проекты на языке объектно-ориентированного программирования Visual Basic .NET. В состав учебно-программного комплекса входят:

- учебное пособие по языку объектно-ориентированного программирования Visual Basic .NET;
- компьютерный практикум на компакт-диске Microsoft-CD, который содержит дистрибутив системы объектно-ориентированного программирования Visual Basic 2005 Express Edition, учебное пособие в формате Web-страниц, а также ответы на задания для самостоятельного выполнения, т. е. готовые проекты на языке программирования Visual Basic .NET;
- методическое пособие для учителей по основам объектно-ориентированного программирования на языке Visual Basic .NET, доступное для загрузки с web-сервера www.microsoft.com\rus\education

2. Министерством образования и науки РФ для обучения программированию рекомендована система программирования Visual Basic .NET. Русскую версию Visual Basic .NET 2003 компания Майкрософт предлагает для учебных заведений по специальным низким ценам и с расширенными условиями по использованию (специальная программа лицензирования для средних учебных заведений: http://www.microsoft.com/Rus/Licensing/Volume/Academic/PilSa.mspx). Облегченная версия Visual Basic 2005 Express Edition по разрешению Microsoft размещена на CD-диске, который входит в состав учебно-методического комплекса.

3. В процессе выполнения проектов в системах объектно-ориентированного программирования производится запись информации на диск. Поэтому необходимо перед запуском проектов скопировать заархивированные папки проектов с Microsoft-CD на жесткий диск.

4. В тексте пособия приняты следующие шрифтовые выделения:

- *курсивом* выделены важные понятия и термины, а также названия диалоговых окон, пунктов меню и управляющих элементов (текстовых полей, кнопок и т. д.) графического интерфейса;
- шрифтом `Courier` выделены тексты программ на языке программирования Visual Basic.

5. Практические задания для самостоятельного выполнения, приведенные в конце параграфов, обозначаются значком  .

6. Материалы, содержащие дополнительную интересную информацию, выделены значком  .

7. Параллельно изложению основ программирования на языке Visual Basic .NET предлагается совершить «путешествие во времени», в начале 12 глав представлена история развития Microsoft через создание различных версий операционных систем и языка программирования Basic.

Глава 1

Программы в повседневной жизни

Microsoft

В июле 1975 года Биллом Гейтсом и Полом Алленом была создана компания Microsoft. В настоящее время корпорация Microsoft является ведущим разработчиком программного обеспечения для компьютеров. Операционная система Windows и ее различные версии вот уже более 15 лет устанавливаются на большинство персональных компьютеров в мире. Язык программирования Visual Basic является одним из наиболее распространенных языков объектно-ориентированного программирования.

1.1. Программы в повседневной жизни

Во множестве самых обычных предметов вокруг нас заложены программы, а еще больше разных вещей создано и проверено с помощью программ. В наше время программы используются не только в калькуляторах, роботах и компьютерах. Сейчас программируемыми стали многие бытовые приборы и игрушки, и даже в автомобилях есть программы.

Например, в автомобиле почти миллион деталей, и многие из них наверняка проектировались с помощью компьютера. Аэродинамику корпуса, скорее всего, разрабатывали и тестировали с помощью модели аэродинамической трубы на компьютере. Сиденья тоже, наверное, были спроектированы на компьютере и включены в 3D-модель автомобиля. Кроме того, многие детали могли быть сделаны роботами, которыми управляют компьютеры.

В автомобиле могут быть система GPS (Global Positioning System — система глобального позиционирования), микроволновка и стереосистема. Эти приборы можно программировать и дистанционно управлять ими. Это значит, что в них есть программы, определяющие, что эти приборы делают, когда мы нажимаем их кнопки. Эти программы постоянно хранятся в чипах на платах в этих приборах, но появились они там благодаря тому, что их когда-то написали люди.

Самое характерное для программ — то, что они могут делать почти всё, решать практически любые задачи. Сейчас программы повсеместно используются на работе и дома, для написания писем, подсчета расходов и т. д. Почти во всех компаниях для учета денег, клиентов, покупок и произведенных товаров тоже используются программы. Для хранения, извлечения, просмотра и фильтрации информации есть огромное множество разных программ. Кроме того, множество программ пишутся для исследования сложных проблем — например, проблем искусст-

Вопросы для размышления

1. Перечислите не менее четырех примеров применения компьютеров в повседневной жизни.

венного интеллекта, глобального потепления или генетики. Есть специальные программы для проектирования машин, рисования, игр — какую бы задачу вы ни придумали, для ее решения, вероятно, уже есть программа. Вспомните об электронной музыке: можно спорить о том, хороша она или не очень, но бесспорно то, что она написана на компьютере. Да, еще существуют программы для работы в Интернете, сетевые программы и программы виртуальной реальности!

Кроме того, некоторые люди пишут программы просто ради интереса! Одна школьная учительница написала программу, автоматически составлявшую домашние задания по математике. Ее сестра написала программу, которая писала японские стихи хокку и переводила их на исландский. Школьники собирают роботов из конструктора Лего и пишут программы по управлению ими.

1.2. Чем занимаются программисты

Многие люди считают, что программисты — это непонятные личности, которые сидят за компьютерами, набирая непонятные тексты, поедают засохшие бутерброды и пьют кофе, работая ночи напролет. Хотя такое представление часто оказывается верным, не это главное.

Хорошую программу нельзя сотворить за одну ночь. Нужно долго работать, планировать и проверять, чтобы создать программу, которая будет делать то, что должна. Кроме того, программа должна позволять легко изменять ее, в ней не должно быть ошибок и с ней должно быть удобно работать. В создании больших программ могут участвовать сотни программистов, а ими нужно управлять, организовывать их и контролировать.

Поэтому, хотя написание программ — одна из задач программистов, программисты делают и множество других вещей:

- Решают, что будет делать программа.
- Проектируют пользовательский интерфейс.
- Решают, какой язык программирования использовать.
- Проектируют архитектуру программы и решают, как части программы будут взаимодействовать между собой.
- Определяют стили написания и оформления кода программы.
- Решают, кто будет писать конкретные части программы.
- Формируют расписание разработки и следят за тем, сколько времени уходит на написание кода.
- Обучают других программистов работе с инструментами разработки программ.
- Подготавливают и обеспечивают работу компьютеров и сетей, используемых другими программистами.
- Пишут программный код.
- Пишут документацию к программному коду.
- Создают базы данных для хранения информации, которая используется программами или создается ими.
- Контролируют создание элементов оформления и графики.
- Контролируют ввод данных в базы данных.
- Тестируют программный код и проверяют правильность его работы.
- Подготавливают программы к установке на компьютеры пользователей или серверы.
- Решают, какие новые возможности добавить в программу.

Вопросы для размышления

1. С помощью Интернета или других ресурсов выясните, какова зарплата у программистов и других специалистов по компьютерам в месте, где вы живете.
2. Какие компании — разработчики программного обеспечения вы знаете?

- Исправляют найденные пользователями ошибки.
- Обучают пользователей работе с программой.
- Пишут документацию и обучающие материалы к программе.

Как видите, написание программного кода — это только одна из множества вещей, которыми занимаются программисты.

1.3. Что такое программа

Компьютерная программа состоит из строк кода, записанного на специальном языке, приближенном к обычному человеческому языку. Программу можно писать на одном из множества языков — язык можно выбирать по его возможностям и потребностям вашей программы. Когда программа написана, она компилируется — преобразуется в язык, понятный компьютеру, чтобы компьютер смог ее выполнить.

На одном и том же языке программирования можно писать программы разных типов. Например, можно написать игру, программу для просмотра рисунков в виде слайд-шоу и программу, просчитывающую траекторию ракеты, — все на одном и том же языке программирования. Эти программы делает разными код, который вы напишете, — именно он придает программам разную функциональность.

Код, который вы пишете, определяет, что программа будет делать. Код определяет, что произойдет, когда вы нажмете кнопку или выберете пункт в списке. Ваш код обеспечивает «разум» программы — он определяет, как программа будет принимать решения, сколько раз она будет выполнять какие-то действия и какие вычисления она будет производить. Можно написать код для выполнения расчетов, написания текста, реагирования на дейст-

Первые компьютерные программы приходилось писать на машинном языке. Постепенно были созданы машинно-независимые языки программирования, использовавшиеся для решения разных задач:

- FORTRAN (расшифровывается как FORmula TRANslator — транслятор формул) — язык, предназначенный для научных и технических расчетов.
- COBOL (расшифровывается как COmmon BusinessOriented Language — стандартный язык для делового применения) — язык в основном, предназначавшийся для коммерческих приложений, обрабатывавших большие объемы нечисловых данных.
- LISP (расшифровывается как List Processing — обработка списков) — язык, созданный для исследований в области искусственного интеллекта.

вия пользователя, сбора данных или показа информации на экране.

Писать код не так уж просто, но это может оказаться очень интересно и захватывающе. Изучив язык программирования, вы получите новое средство выражения своего стремления творить, исследовать, решать проблемы и играть. Программирование похоже на работу с глиной, металлом или математикой.

1.4. Возможности языков программирования

Компьютерные языки используются практически для того же, для чего и обычные человеческие языки, — для сообщений. Они позволяют сообщить компьютеру, что он должен делать, чтобы решить задачу.

Знаете ли вы, что постоянно появляются новые языки программирования? За последние 50 лет было создано множество разных языков программирования. Как и человеческие языки, некоторые языки программирования больше не используются. На их место пришли другие языки. Некоторые языки программирования развиваются и используются уже 20 или 30 лет.

При создании нового языка программирования обычно берутся лучшие черты и возможности уже существующих языков, к ним добавляется что-то новое, и получается новый язык. Современные языки программирования учитывают изменения в компьютерах и развитие ранее созданных программ.

Первые компьютерные программы приходилось писать на машинном языке, для того чтобы их сразу понимал компьютер. Эти программы представляли собой длинные последовательности единиц и нулей. Это было очень неудобно! Поэтому были созданы программы, названные компиляторами. Компиляторы могли преобразовывать в

- BASIC (расшифровывается как Beginner's All-Purpose Symbolic Instruction Code — универсальный язык символьных инструкций для начинающих) — язык, отличающийся простотой использования.
- Pascal (назван его создателем Виртом в честь великого физика Блеза Паскаля) — язык, позволяющий легко кодировать основные алгоритмические структуры.

Затем появились новые языки, с теми или иными преимуществами над существовавшими ранее. Вот несколько примеров:

- C и C++ — языки, позволяющие создавать быстро и эффективно выполняющийся программный код.
- Java — язык, обеспечивающий платформенную независимость, т. е. позволяющий создавать программы, которые выполняются в среде различных операционных систем.

На следующем этапе развития были созданы объектно-ориентированные языки программирования, позволяющие визуально конструировать графический интерфейс приложений: понятную компьютеру форму код, написанный на более удобных для людей языках. Почти все современные языки программирования — компилируемые. Это значит, что написанные на них программы, напоминающие обычный английский текст, преобразуются в понятный компьютеру вид.

Начиная новый проект, вы должны решить, какой язык программирования будете использовать. Есть ли у выбранного языка все нужные возможности? Легко ли его использовать? Знаете ли вы этот язык? Установлен ли он на вашем компьютере? Знать больше одного языка программирования никогда не вредно! В этом пособии будет изучаться Visual Basic .NET, но будет рассказано также немного и о J# и C#.

1.5. Синтаксис языков программирования

В путешествиях вам может понадобиться найти человека, разговаривающего по-английски. На любом языке можно сформулировать вопрос «Говорите ли вы по-английски?» Например:

Английский: Do you speak English?

Немецкий: Sprechen Sie Englisch?

Испанский: Habla ingles?

Португальский: Vock fala ingles?

В этих примерах отличаются не только слова, но и порядок слов в предложениях — существительные, глаголы и местоимения расположены по-разному. Слова и порядок их размещения образуют синтаксис языка. Синтаксис языка определяет правила, позволяющие составлять правильные предложения на этом языке.

- Visual Basic — язык, созданный корпорацией Microsoft для разработки приложений с графическим интерфейсом в среде операционной системы Windows.
- Delphi — среда разработки (язык Object Pascal), созданная компанией Borland для разработки приложений с графическим интерфейсом в среде операционной системы Windows.

В настоящее время многие программисты выбирают интегрированную систему программирования Visual Studio .NET на платформе .NET Framework, которая предоставляет возможность создавать приложения на различных языках объектно-ориентированного программирования, в том числе таких, как:

- Visual Basic .NET — мощный и простой в применении язык.
- J# (читается J-шарп) — язык, созданный на основе языка Java.
- C# (читается C-шарп) — язык, созданный на основе языков C и C++.

Как и у человеческих языков, у любого языка программирования тоже есть синтаксис. Синтаксис — это словарь, набор грамматических правил и структур, образующих язык программирования. Синтаксис определяет правила написания правильных строк кода и объединения этих строк в работающую программу.

Например, во всех современных языках программирования есть оператор `If...Then...Else`. Оператор `If...Then...Else` — один из способов, позволяющих программе принять решение, исходя из имеющейся у нее информации. Рассмотрим пример разного синтаксиса операторов `If...Then...Else` в языках Visual Basic .NET, C# и J#. Не волнуйтесь, если вы не понимаете, что означают эти операторы. Просто пока обратите внимание на похожие элементы и различия между разными языками. Эти различия определяются синтаксисом разных языков.

На Visual Basic .NET:

```
If x>5 Then
  MessageBox.Show("Я больше 5.")
Else
  MessageBox.Show ("Я не больше 5.")
End If
```

На C#:

```
if (x>5)
{
  MessageBox.Show ("Я больше 5.");
}
else
{
  MessageBox.Show ("Я не больше 5.");
}
```

Вопросы для размышления

1. Проанализируйте приведенный пример. В каких языках программирования синтаксис оператора одинаков, а каких — различается?

На J#:

```
if (x>5)
{
  MessageBox.Show ("Я больше 5.");
}
else
{
  MessageBox.Show ("Я не больше 5.");
}
```

Изучая язык программирования, нужно изучить его синтаксис, структуру и правила написания программ. Нужно знать правила пунктуации языка, а также его зарезервированные слова. Эти слова называются зарезервированными, потому что они используются только как команды языка. Зарезервированные слова, называемые также ключевыми словами, имеют особое значение и обычно выполняют специальные функции — они указывают программе, что делать. Например, слова `If` и `Else` являются зарезервированными в большинстве языков программирования. Они используются в операторах принятия решений в программах.

Изучая язык программирования, нужно изучать не только синтаксис, но и функциональность и возможности этого языка. Не забывайте, что не на любом языке программирования можно сделать что угодно. Но во всех современных языках программирования есть основные функциональные возможности, которые можно использовать.

Рассмотрим некоторые элементы синтаксиса, одинаковые в языках C# и J#. В C# и J# каждая строка кода заканчивается точкой с запятой. Блоки кода помещаются в фигурные скобки, т. е. символы { и }. Строка комментария начинается с символов //. Язык C# и J# чувствителен к регистру символов. `MyCase` в них — это не то же самое, что `myCase` или `MYCASE`.

Приведем некоторые элементы синтаксиса языка Visual Basic .NET:

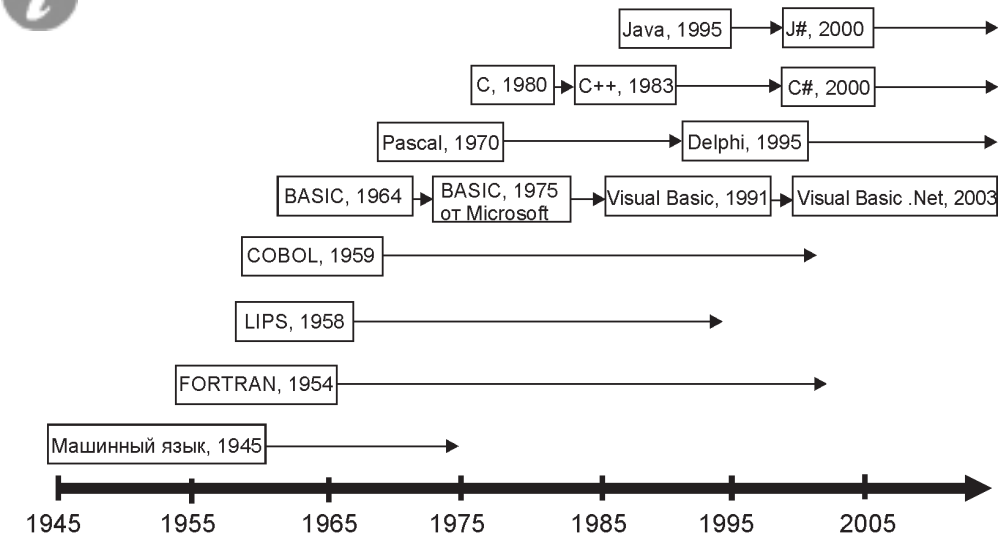
- Концы строк кода не помечаются специальными символами, например точкой с запятой (;).
- Строки комментариев начинаются с символа «апостроф» (').

Вопросы для размышления

1. О каких различиях синтаксиса языков программирования Visual Basic .NET, C# и J# вы узнали?

- Для выделения блоков кода не используются фигурные скобки { и }.
- Visual Basic .NET НЕ чувствителен к регистру. `MyCase` — это то же самое, что и `myCase` или `MYCASE`. В Visual Basic эти три обозначения считаются одинаковыми.

История развития языков программирования



Тест по теме «Программы в повседневной жизни»

1. 

Синтаксис языка программирования, в частности, определяет

- ❍ *Словарь, правила записи операторов и структуры языка*
- ❍ *Последовательность строк программы*
- ❍ *Тип языка программирования*
- ❍ *Выбор компилятора*

2. 

Что происходит с программой при компиляции?

- ❍ *Сохранение программы на жестком диске*
- ❍ *Перевод программы на другой машинно-независимый язык программирования*
- ❍ *Преобразование программного кода в понятную компьютеру форму*
- ❍ *Преобразование программного кода в понятную человеку форму*

3. 

Множество языков программирования появилось потому, что

- ❍ *Ученым нужно что-то делать*
- ❍ *Каждый язык предназначен для решения задач определенного типа*
- ❍ *Каждой компьютерной фирме нужен свой язык программирования*
- ❍ *Каждой стране нужен свой язык программирования*

4. 

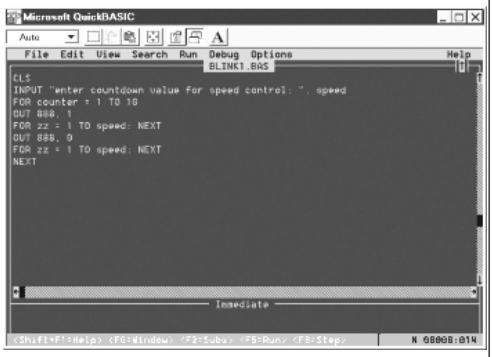
Программа на современном языке программирования состоит из

- ❍ *Последовательностей нулей и единиц*
- ❍ *Строк кода, написанного на языке, приближенном к человеческому*
- ❍ *Математических формул*
- ❍ *Логических формул*

Глава 2

Система программирования Visual Basic .NET

Microsoft Язык BASIC (Beginner's All-Purpose Symbolic Instruction Code) — простой язык программирования для начинающих — был создан в 1964 году. В 1975 году Билл Гейтс и Пол Аллен разработали интерпретатор языка BASIC для персональных компьютеров. Затем в середине 80-х годов XX века появилась система программирования Microsoft Quick BASIC, а в 1991 году — система программирования и язык Visual Basic.



Между прочим, можно набирать тексты программ в обычном текстовом редакторе вроде Notepad или Microsoft Word. Правда, писать программы таким образом очень неудобно.

2.1. Visual Studio .NET и IDE

Система программирования Microsoft Visual Studio .NET — это инструмент разработки программ, позволяющий писать программы на нескольких языках программирования, известных как языки .NET. Вместе с Visual Studio .NET поставляются следующие языки .NET:

- Visual Basic .NET
- C# (произносится Си-шарп)
- J# (произносится Джей-шарп)
- C++ (произносится Си плюс плюс)
- ASP .NET (для создания web-страниц)

Для работы со всеми этими языками Visual Studio .NET предоставляет один и тот же интерфейс IDE. IDE расшифровывается как Integrated Development Environment (Интегрированная среда разработки).

Интегрированная среда разработки Visual Studio .NET облегчает разработку программ. Например, если вы добавляете на форму кнопку, система программирования Visual Studio .NET автоматически создает код для этой кнопки. Конечно, Visual Studio .NET не может написать за вас весь код, так как система программирования не знает, что должна делать ваша программа. Но система программирования, безусловно, может уменьшить объем кода (и затраты времени), который вам приходится писать собственноручно.

Система программирования Visual Studio .NET помогает обнаруживать и исправлять ошибки до запуска программы. Она автоматически раскрашивает код в зависимости от назначения этого кода. Например, комментарии всегда отображаются зеленым цветом, ключевые слова — синим, а синтаксические ошибки подчеркиваются волнистой красной линией.

Visual Studio .NET помогает упорядочивать код, храня его в нескольких отдельных файлах. Это позволяет разделить программу на функциональные блоки. Например, код каждой формы можно хранить в отдельном файле.

Visual Studio .NET позволяет легко компилировать и запускать программы — для этого нужно сделать всего несколько щелчков мышью.

С помощью утилит отладки, входящих в Visual Studio .NET, можно искать ошибки в программе и отслеживать ее выполнение.

В следующем параграфе вы узнаете, как с помощью Visual Studio .NET написать первую программу. Вы встретите также и примеры кода на C# и J#.

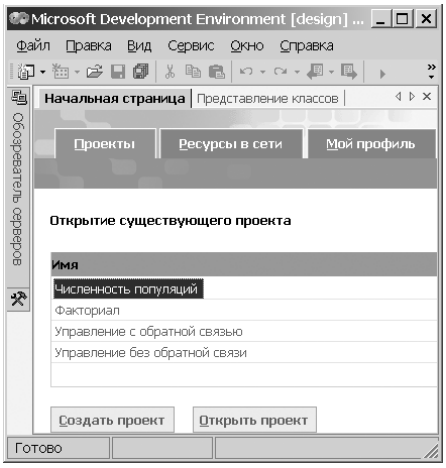
2.2. Запуск и настройка Visual Studio .NET

Чтобы запустить Visual Studio .NET 2003:

1. Щелкните на кнопке *Пуск*.
2. Выберите пункт *Программы*.
3. Выберите пункт *Microsoft Visual Studio .NET 2003*.
4. Выберите пункт *Microsoft Visual Studio .NET 2003*.

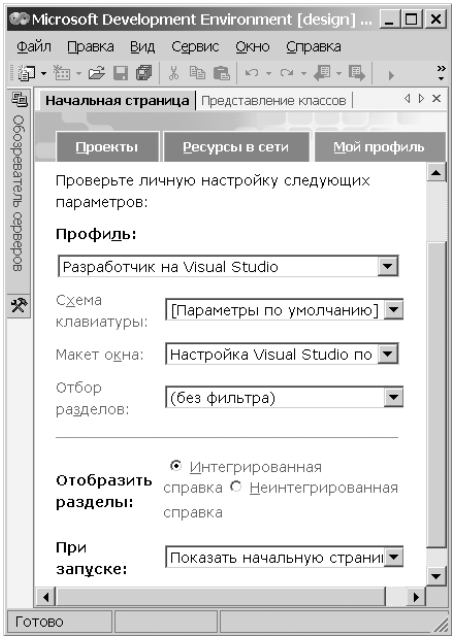
На *Рабочем столе* можно создать значок для быстрого запуска Visual Studio. Он позволяет быстро запустить Visual Studio .NET — просто сделайте на этом значке двойной щелчок мышью!

Первое, что вы увидите на экране, запустив Visual Studio .NET, — начальную страницу (*Начальная страница*). По умолчанию будет открыта вкладка *Проекты*. Обычно на ней приводится список недавно открывавшихся проектов (*Открытие существующего проекта*). Для открытия существующего проекта щелкнуть по кнопке *Открыть проект*, а для начала создания нового проекта — щелкнуть по кнопке *Создать проект*.



Вкладка *Мой профиль* позволяет настроить Visual Studio .NET под ваш стиль работы, соответственно вашим предпочтениям.

1. Щелкните по ярлычку вкладки *Мой профиль*.



2. В появившемся диалоговом окне в выпадающем списке *Профиль:* выберите пункт *Разработчик на Visual Studio*.

3. Убедитесь, что в выпадающих списках выбраны настройки по умолчанию:
   - *Схема клавиатуры*: *[Параметры по умолчанию]*
   - *Макет окна*: *Visual Studio по умолчанию*
   - *Отбор разделов*: *(без фильтра)*

4. В пункте *Отобразить разделы* выберите опцию *Интегрированная справка*.

5. В выпадающем списке *При запуске* выберите пункт *Показать начальную страницу*.

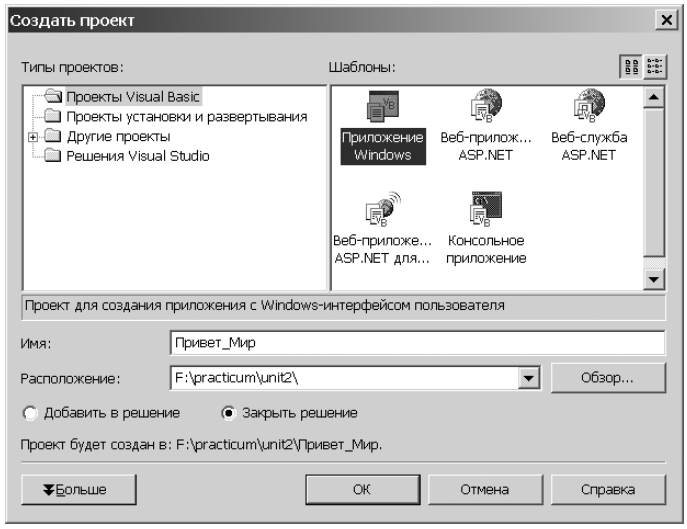
6. Перейдите на вкладку *Проекты*, щелкнув по ее ярлычку в верхней части страницы.

2.3. Создание первого проекта

Теперь, настроив систему программирования Visual Studio .NET, мы разберемся с вкладкой *Проекты*. Эта вкладка, как уже говорилось в § 2.2, позволяет и создавать новые проекты, и открывать уже существующие. Когда вы создадите свой первый проект, он появится в списке существующих проектов. Чтобы открыть существующий проект, нужно выбрать его из списка щелчком мышью.

А теперь приступим к созданию первого проекта «Привет, Мир». Этот проект выведет на экран окно с сообщением "Привет, Мир".

1. Щелкните на кнопке *Новый проект*, чтобы создать новый проект. Откроется диалоговое окно *Создать проект*.



В окне *Создать проект* вы увидите, что в левой части, в панели *Типы проектов:* перечислено множество типов проектов, которые можно создавать в Visual Studio .NET. С некоторыми из этих типов вы вскоре познакомитесь. Проект «Привет, Мир» будет приложением Windows, поэтому в правой части окна (панель *Шаблоны:*) необходимо выбрать шаблон *Приложение Windows*. В окне *Создать проект* нужно выбрать не только тип проекта и шаблон, но и имя проекта и папку, в которой будут размещаться его файлы.

2. Выберите тип проекта *Проекты Visual Basic* в панели *Типы проектов:*.

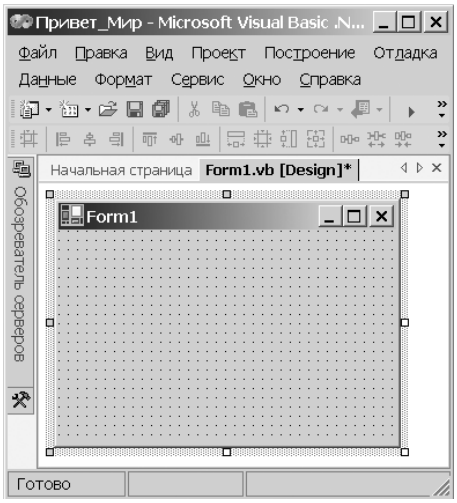
3. Выберите шаблон *Приложение Windows* в панели *Шаблоны:*.

4. Введите *Привет_Мир* в качестве имени проекта в текстовом поле *Имя* (в имени проекта лучше не использовать пробелы).

5. Выберите папку, в которой хотите сохранить проект. В папке, которую вы выбрали для размещения проекта, будет создана новая папка Привет_Мир, и проект "Привет_Мир" будет сохранен в созданной папке.
6. Нажмите кнопку *OK*. Откроется новый созданный проект, и на экране будут отображены форма `Form1.vb` (в левой части) и `Solution Explorer` (в правой части). Ничего сложного, правда?

2.4. Конструирование графического интерфейса проекта

Окно *Конструктор форм* (*Form Design Window*). Чаще всего мы будем создавать приложения Windows, начинающиеся с формы. Форма — это элемент графического интерфейса, с помощью которого осуществляется взаимодействие пользователя с программой. Форма представляет собой поле с размещаемыми на нем объектами разных типов — кнопками, текстовыми полями. При нажатии кнопки запускается код, выполняющий запланированные действия.

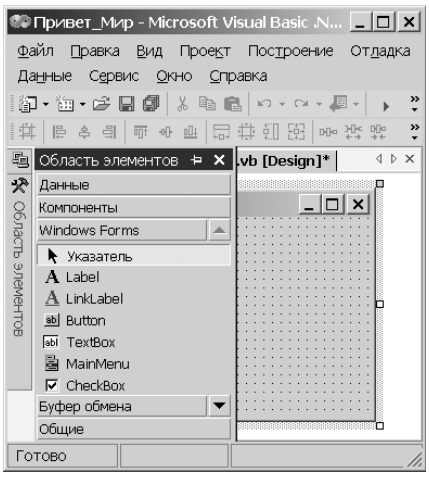


Любое окно в Visual Studio .NET можно закрыть щелчком по кнопке со значком «x» в верхнем правом углу этого окна. Большинство окон в Visual Studio .NET можно открыть соответствующими командами из меню *Вид* (*View*).

Можно автоматически скрыть окно, щелкнув на значке в виде кнопки для крепления бумаги, расположенном в верхнем правом углу окна. При автоматическом скрытии окна оно перестает отображаться на экране. На панели, которая отобразится вдоль правого края окна, будут присутствовать вкладки для автоматически скрытых окон. Если подвести курсор к вкладкам на этой панели, то скрытые окна будут отображаться. Автоматическое скрытие окон — очень удобный способ увеличения рабочей площади окна. Если щелкнуть по значку кнопки для бумаг в окне, выдвинувшемся из панели, режим автоматического скрытия для этого окна будет выключен.

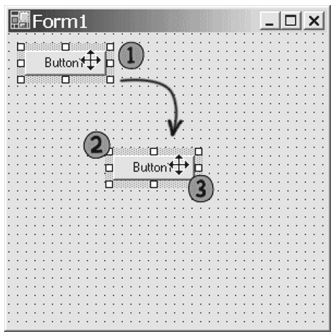
В основной рабочей области в левой части окна будет отображена пустая форма, на которой нет кнопок и других объектов. Эта область называется окном *Конструктор форм*. Ярлычок этого окна содержит надпись *Form1.vb [Design]*.

Окно *Область элементов* (*Toolbox*). Доступ к значительной части функций Visual Studio .NET выполняется с использованием различных окон. Например, чтобы поместить на форму объекты типа кнопок и текстовых полей, нужно воспользоваться окном *Область элементов*. В этом окне содержатся все объекты, которые можно поместить на форму, — кнопки, переключатели, текстовые поля, выпадающие списки и т. д.



1. Откройте меню *Вид* (*View*) в строке меню.
2. Выберите пункт *Область элементов* (*Toolbox*) — откроется окно *Область элементов*.
3. Двойным щелчком по пункту *Button* поместите на форму кнопку. (Эта кнопка может оказаться спрятанной под окном *Область элементов*.)

Перемещение кнопки и изменение ее размеров. Теперь, поместив кнопку на форму, мы можем переместить эту кнопку в нужное нам место и придать ей требуемый размер.



1. Подведите курсор мыши к кнопке, нажмите левую кнопку мыши и, удерживая ее нажатой, перетащите кнопку в центр формы, после чего отпустите кнопку мыши.
2. Чтобы изменить размер кнопки, подведите курсор мыши к одному из белых квадратиков вокруг кнопки (это метки изменения размера).
3. Нажмите левую кнопку мыши, и, удерживая ее нажатой, перемещайте метку, чтобы растянуть или сжать кнопку до требуемого размера.

Ничего сложного, правда? Теперь мы добавим код, который заставит программу выполнять действия при нажатии этой кнопки.

2.5. Создание программного кода проекта

Когда мы добавили кнопку на форму, Visual Studio .NET автоматически добавил к коду этой формы несколько новых строк. Взгляните на пример ниже.

```
Public Class Form1
Inherits System.Windows.Forms.Form

Код, автоматически созданный конструктором форм
Windows
Private Sub Button1_Click(ByVal sender As_
System.Object, ByVal e As System.EventArgs)_
Handles Button1.Click

End Sub
End Class
```

Непонятно? Мы разберемся со всем, что здесь написано, немного позже. А пока просто найдите фрагмент кода со словом `Button1_Click`. Этот фрагмент выполняется, когда пользователь щелкает по кнопке, которую мы поместили на форму. Пока в данном фрагменте не предусмотрено действий реакции на щелчок. Мы должны сообщить приложению, что оно должно делать при щелчке по этой кнопке. Именно в этом и заключается смысл программирования!

1. Сделайте двойной щелчок по кнопке `Button1`. Откроется окно *Редактор кода*.

2. Щелкните по пустой строке над строкой `End Sub`.

3. Введите в пустую строку с клавиатуры строку кода (точно так, как показано ниже):

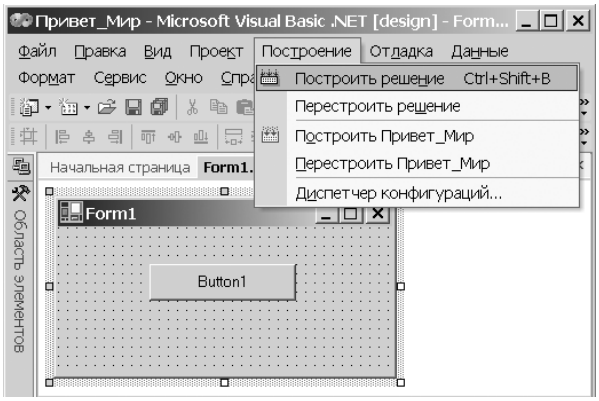
```
Private Sub Button1_Click(ByVal sender As_
System.Object, ByVal e As System.EventArgs)_
Handles Button1.Click
MessageBox.Show("Привет, Мир.")
End Sub
```

Этот код будет выполняться, когда пользователь будет щелкать по кнопке на форме.

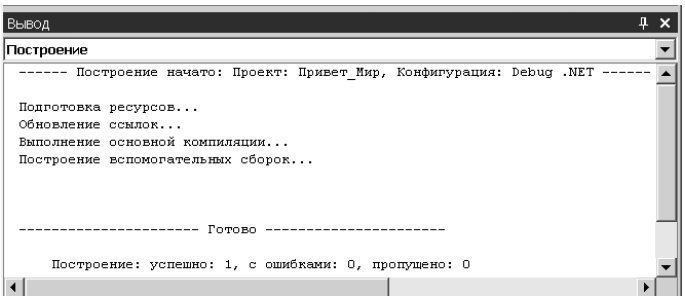
2.6. Построение решения

Теперь нужно построить решение. При построении решения код, написанный вами и Visual Studio .NET, компилируется в понятные компьютеру инструкции.

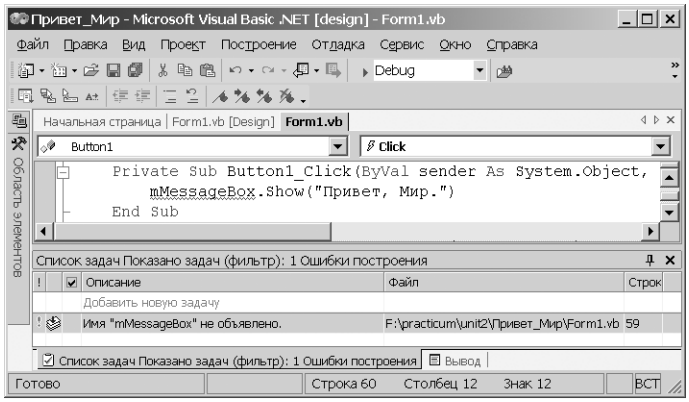
1. Откройте меню *Построение* (*Build*) в строке меню.
2. Выберите пункт *Построить решение* (*Build Solution*) — начнется построение решения.



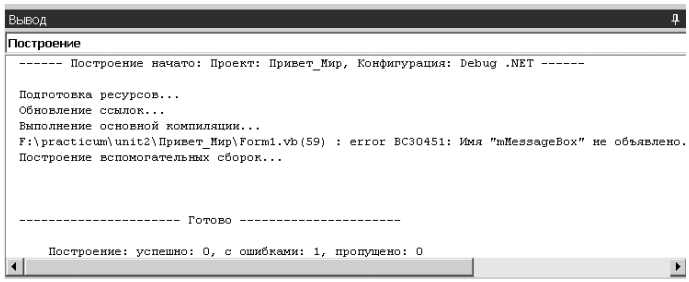
Процесс построения решения будет отображаться в окне *Вывод*. Если в программном коде не было сделано ошибок, после завершения построения в окне *Вывод* будет выведено сообщение о том, что построение выполнено успешно.



В программном коде могли быть сделаны ошибки, пусть, например, вместо `MessageBox` вы набрали `mMessageBox`. Если Visual Basic может определить, где произошла ошибка, он подчеркивает это место в программном коде синей волнистой линией. Кроме того, открывается окно *Список задач* (*Task List*), содержащее список ошибок, которые нужно исправить. Двойной щелчок по сообщению об ошибке перемещает курсор в коде к этой ошибке.



В окне *Вывод* (*Output*) появится сообщение о том, что построение выполнить не удалось.

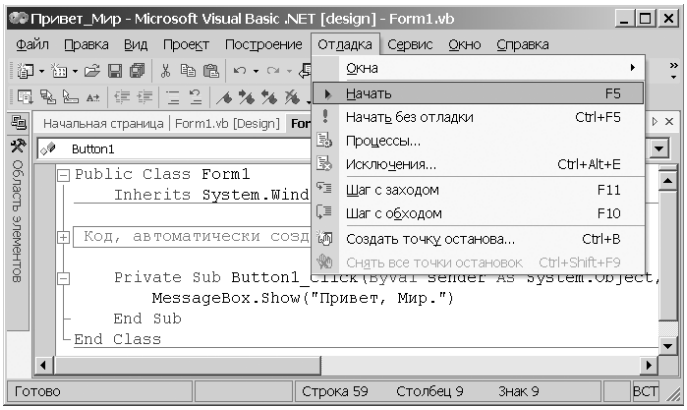


Если построение выполнить не удалось, вернитесь назад и введите строку кода точно в том виде, в каком она приведена в этом курсе. Затем еще раз попытайтесь построить решение, как показано ранее.

2.7. Запуск проекта

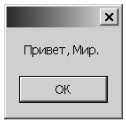
Вам удалось скомпилировать программу. Это большой шаг к тому, чтобы стать программистом! Теперь скомпилированную программу можно запустить.

1. Откройте меню *Отладка* (*Debug*).



2. Выберите в этом меню пункт *Начать* (*Start*). (На экране появится форма проекта с кнопкой, которую вы на нее поместили.)

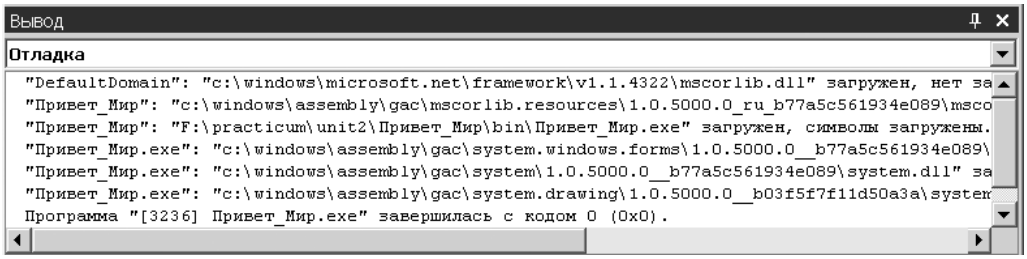
3. Щелкните на кнопке, появится окно сообщения с текстом «Привет, Мир». Именно это и должна была делать программа! Поздравляем! Вы только что создали ваш первый проект на языке Visual Basic .NET!



4. Нажмите кнопку *OK*, чтобы закрыть окно сообщения. (Окно сообщения закроется, но основное окно проекта останется открытым. Можете еще раз нажать кнопку, и окно сообщения появится снова.)

5. Нажмите кнопку со значком ✕ в верхнем правом углу формы, чтобы закрыть программу.

После завершения выполнения программы в окне *Вывод* (*Debug*) отображается отладочная информация. Чтобы посмотреть выведенные сообщения полностью, вам, вероятно, придется воспользоваться линейкой прокрутки.

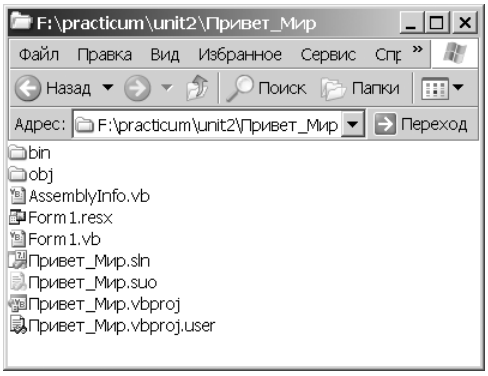


2.8. Сохранение проекта

Сохранение проекта. Пора сохранить ваш проект. Сохранение выполняется так же, как и в других программах в операционной системе Windows.

1. Откройте меню *Файл* (*File*).
2. В меню *Файл* (*File*) выберите команду *Сохранить все* (*Save All*).
3. Чтобы выйти из Visual Studio .NET, откройте меню *Файл* (*File*).
4. В меню *Файл* (*File*) выберите команду *Выход* (*Exit*).

Файлы и папки решения. Когда вы создаете приложение на Visual Basic, создается целый набор файлов. Вы должны знать, какие это файлы и для чего они предназначены. На рисунке ниже показан обычный набор файлов решения.



Перейдите в папку ..\practicum\unit2\, в которой вы создали проект «Привет, Мир». Откройте эту папку, и вы увидите, что для проекта была создана отдельная папка, названная Привет_Мир. Зайдите в папку Привет_Мир, чтобы посмотреть файлы, созданные для вашего проекта Visual Studio .NET. В папке будут следующие файлы:

Привет_Мир.sln — это файл решения, в котором хранится описание всех файлов и настроек этого решения. Вообще-то в решение можно включить несколько проектов, но в решении «Привет, Мир» проект только один — это файл Привет_Мир.vbproj. К файлу Привет_Мир.sln система обращается, когда вы открываете решение.

Файл Form1.vb содержит форму и связанный с ней код.

Откройте папку ..\bin\, сделав по ее значку двойной щелчок мышью. В папке ..\bin\ содержится исполняемый файл, полученный компиляцией программы на Visual Basic. Это файл Привет_Мир.exe. Его можно запустить на другом компьютере, даже если на нем не установлена система программирования Visual Studio .NET.*

---

* Для выполнения программ на языках .NET на компьютерах должна быть установлена .NET Framework. В Windows XP, начиная с Service Pack2, она входит в состав операционной системы.

Microsoft-CD

Задания для самостоятельного выполнения

2.1. В системе программирования Visual Basic .NET создать проект «Привет, Мир», описанный в параграфе. Готовый проект содержится в самораспаковывающемся архиве Привет_Мир.exe.

2.2. В системе программирования Visual Basic .NET создать проект «Мое имя», который должен выводить на экран окно сообщения, содержащее ваше имя. Готовый проект содержится в самораспаковывающемся архиве Мое_имя.exe.

2.9. Вывод сообщений на форму

Часто вашим программам понадобится вести диалог с пользователем. Возможно, программа должна будет сообщить что-то пользователю, позволить ему ввести какие-то данные или что-то указать.

Окно сообщений. Такие ситуации встречаются настолько часто, что Visual Studio .NET поддерживает набор стандартных окон, например окон сообщений (`MessageBox`), для обработки таких ситуаций.

В первой написанной вами программе вы использовали строку кода, выводившую на экран окно сообщения:

```
MessageBox.Show("Привет, Мир.")
```

В языке C# окно сообщения вызывается почти так же, как и в Visual Basic .NET. Точно так же оно вызывается и в языке J#:

```
{
MessageBox.Show
("Привет, Мир");
}
```

Вид окна сообщения встроен в Visual Basic .NET. Вам не нужно создавать вид окна, чтобы использовать его в проекте. Все, что нужно сделать в программе — вызвать метод `Show()` и задать текст, который нужно вывести на экран. Синтаксис использования окна сообщения таков:

```
MessageBox.Show("Текст сообщения")
```

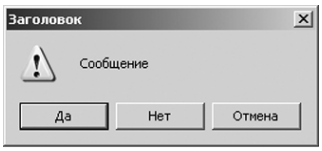
Обратите внимание, что текст сообщения должен быть помещен в кавычки.

Функция вывода сообщений MsgBox(). Функция MsgBox() позволяет выводить сообщения с помощью окна сообщений, в котором можно разместить определенный набор кнопок и информационный значок о типе сообщения.

```
MsgBox("Сообщение"
[,ЧисКод1+ЧисКод2][,"Заголовок"])
```

Аргумент "Сообщение" выводится в окно сообщений, аргумент ЧисКод1+ЧисКод2 определяет внешний вид панели, а строка "Заголовок" печатается в строке заголовка панели. Последние два аргумента не являются обязательными, поэтому заключаются в квадратные скобки.

Например, для функции MsgBox("Сообщение", 48 + 3, "Заголовок") будет выведено следующее окно сообщений:



Внешний вид окна сообщений можно менять, используя различные значения ЧисКод1 и ЧисКод2. Значение ЧисКод1 определяет вид пиктограммы, которая помещается в окно сообщений, а значение ЧисКод2 определяет набор кнопок, размещаемых в окне.

Значения ЧисКод1 и ЧисКод2, определяющие вид окна сообщений

ЧисКод1	Тип сообщения	Пиктограмма
16	Ошибка	
32	Вопрос	
48	Внимание	
64	Информация	

ЧисКод2	Набор кнопок
0	*ОК*
1	*ОК, Отмена*
2	*Стоп, Повторить, Пропустить*
3	*Да, Нет, Отмена*
4	*Да, Нет*
5	*Повторить, Отмена*

Элемент управления `Label` (Надпись). `Label` предназначен для отображения текста или изображений, которые пользователь не может изменить с клавиатуры. Отображаемый на надписи текст содержится в свойстве `Text`. Свойство `Alignment` позволяет задать выравнивание текста на надписи.

Например, для вывода на надпись `Label1` текста «Привет, Мир» необходимо в программный код ввести следующую строку:

```
Label1.Text = "Привет, Мир."
```

Элемент управления `TextBox` (Текстовое поле). `TextBox` используется для приема данных, вводимых пользователем, или для отображения текста. На элемент управления `TextBox` обычно помещается редактируемый текст, хотя его можно также сделать доступным только для чтения. В текстовые поля можно выводить несколько строк текста, размещать текст в соответствии с размером элемента управления и добавлять основные элементы форматирования.

Например, для вывода в текстовое поле `TextBox1` текста «Привет, Мир» необходимо в программный код ввести следующую строку:

```
TextBox1.Text = "Привет, Мир."
```

Microsoft-CD

Задания для самостоятельного выполнения

2.3. В системе программирования Visual Basic .NET модернизировать проект «Привет, Мир», добавив вывод сообщения с использованием функции вывода сообщений, надписи и текстового поля. Заготовка проекта содержится в самораспаковывающемся архиве Привет_Мир.exe.

2.4. В системе программирования Visual Basic .NET модернизировать проект «Мое имя», добавив вывод сообщения с использованием функции вывода сообщений, надписи и текстового поля. Заготовка проекта содержится в самораспаковывающемся архиве Мое_имя.exe.

Тест по теме «Система программирования Visual Basic .NET»

1. 

В Visual Studio .NET проект хранится

- ❍ *В одном файле проекта*
- ❍ *В одном файле решения*
- ❍ *В иерархической системе папок решения*
- ❍ *В папке проекта*

2. 

С помощью какого окна выбираются элементы управления для размещения их на форме?

- ❍ *Редактор кода (Code)*
- ❍ *Конструктор форм (Form Design Window)*
- ❍ *Область элементов (Toolbox)*
- ❍ *Окно свойств (Properties)*

3. 

Перед запуском программы на выполнение нужно

- ❍ *Скомпилировать программу*
- ❍ *Сохранить проект*
- ❍ *Настроить профиль*
- ❍ *Запустить справочную систему*

4. 

Visual Studio .NET можно настраивать под конкретного пользователя на странице

- ❍ *Проекты (Projects)*
- ❍ *Ресурсы в сети (Online Resources)*
- ❍ *Справка (Help)*
- ❍ *Мой профиль (Profile)*

Глава 3

Алгоритмы и программы

Microsoft В 1981 году была разработана операционная система MS-DOS (Microsoft Disk Operation System — дисковая операционная система) для персональных компьютеров. MS-DOS требовала ввода команд с клавиатуры в командной строке. В операционной системе Windows существует возможность перехода в режим MS-DOS, который иногда предпочитают опытные программисты.

3.1. Основные элементы кода

Во всех языках программирования используются базовые элементы одних и тех же видов, хотя синтаксис, с помощью которого эти элементы записываются, свой в каждом языке. Из этих элементов можно составлять любые, даже самые сложные программы.

Помните строку кода, которую вы добавили в проект «Привет, Мир»? Она выглядела так:

```
MessageBox.Show("Привет, Мир.")
```

Эта строка выводит на экран информацию для пользователя с использованием метода `Show` объекта `MessageBox`. Во многих объектно-ориентированных языках программирования есть такой метод, сообщающий информацию пользователю, хотя его синтаксис разный в разных языках. Есть и другие элементы кода, поддерживаемые всеми языками программирования.

Вот несколько примеров других элементов программ, поддерживаемых всеми языками:

- Переменные используются для хранения информации, например чисел или текста. Это «контейнеры» для хранения информации или результатов вычислений.
- Операторы присваивания помещают, или записывают, значения в переменные. Значение может быть числом, текстом, значением другой переменной или результатом вычислений.
- Операторы сравнения позволяют сравнивать значение переменной с числом или со значением другой переменной. Эти операторы как бы задают вопрос, ответ на который — всегда «истина» или «ложь», т. е. «да» или «нет».
- Операторы принятия решения решают, что делать дальше. Принимаемое решение основывается на результатах сравнения. Если результат сравнения «истина», то выбирается

первый вариант дальнейших действий. Если результат сравнения «ложь», то выбирается второй вариант. Операторы принятия решения часто называются операторами ветвления, поскольку последовательность выполнения программы можно «разветвлять» с помощью этих операторов.

- Операторы цикла используются для многократного выполнения каких-нибудь действий. Эти операторы уменьшают количество кода, которое вам придется писать, создавая программу, если какие-то действия нужно повторять много раз.

Изучая программирование, вы узнаете, как с помощью операторов этих видов заставлять компьютер делать то, что вам хочется. Изучая Visual Basic .NET (или любой другой язык программирования), вы должны будете запомнить синтаксис, с помощью которого в этом языке программирования записываются основные элементы программ.

3.2. Алгоритм в форме псевдокода

Рассмотрим, как планировать и организовывать программу еще до того, как вы начнете писать ее код.

Многие программисты сначала создают алгоритм, который может быть записан на так называемом псевдокоде, а потом переводят их на настоящий язык программирования. Алгоритм можно записать на естественном языке (английском, русском или вашем родном), объяснив, что программа должна делать, шаг за шагом.

На псевдокоде описывается структура программы и последовательность ее работы. Алгоритм на псевдокоде должен быть достаточно понятным, чтобы вы и другие программисты могли в нем разобраться, поскольку он закладывает основу для написания настоящего кода программы. Алгоритм

Начинать писать программу с создания алгоритма в форме псевдокода удобно, потому что потом псевдокод можно преобразовать в код на разных языках программирования, и не приходится с самого начала привязываться к одному языку. Сначала нужно описать с помощью псевдокода логику работы и структуру программы. Затем псевдокод нужно «перевести» на тот язык программирования, который вы решите использовать.

состоит из операторов разных базовых видов — операторов присваивания, сравнения, принятия решений и т. д. Создав алгоритм, сравнительно несложно записать его в виде программного кода.

Предположим, нам нужно написать программу, которая вычислит среднюю стоимость бензина в рублях за литр, если бензин покупался на заправках различных фирм. Что эта программа должна делать? Чтобы ответить на этот вопрос, мы сначала напишем программу с помощью псевдокода. Вот действия, которые наша программа должна будет выполнять, чтобы вычислить среднюю стоимость бензина:

1. Просуммировать все деньги, которые мы потратили на бензин, и записать эту сумму в переменную (`TotalRub`).
2. Найти общее количество купленного бензина и записать это количество в другую переменную (`TotalLiter`).
3. Разделить значение первой переменной (`TotalRub`) на значение второй переменной (`TotalLiter`). Результат деления записать в третью переменную (`RubPerLiter`).
4. Вывести сообщение, в котором указана средняя цена бензина.

Псевдокод показывает, что должна будет делать программа. Теперь можно написать код, выполняющий действия, перечисленные в псевдокоде.

Вот еще один пример, на этот раз с большим количеством шагов. Это пример описания программы для робота, который будет заменять проколотые шины.

1. Определить, какая шина проколота (оператор сравнения).
2. Определить размер проколотой шины (оператор сравнения).

3. Если нет шины нужного для замены размера, то получить шину для замены со склада (операторы сравнения и принятия решения).
4. Поднять автомобиль с помощью домкрата.
5. Снять колпак, чтобы добраться до крепления шины.
6. Снять шину, откручивая по очереди все гайки с помощью гаечного ключа (оператор цикла).
7. Удалить проколотую шину.
8. Надеть новую шину.
9. Закрутить по очереди все гайки с помощью гаечного ключа (оператор цикла).
10. Вернуть на место колпак и закрепить его.
11. Убрать домкрат.

В этом псевдокоде есть операторы присваивания, операторы сравнения, операторы принятия решений и операторы цикла. Например, нужно запоминать размер проколотой шины — это делает оператор присваивания. Нужно сравнить размер проколотой шины с размерами шин, которые есть в запасе, — это оператор сравнения. После оператора сравнения используется оператор принятия решения. Если у нас есть шина нужного размера, мы можем продолжать работу, в противном случае нужно сначала обратиться на склад. Откручивание гаек и их закручивание можно воспринимать как циклически повторяющиеся операции. Эти операции нужно повторять для каждой гайки — сначала при откручивании, а потом при закручивании.

Microsoft-CD

Задания для самостоятельного выполнения

3.1. Запишите в форме псевдокода алгоритм включения компьютера, запуска операционной системы Windows и системы программирования Visual Studio .NET.

Добавляйте в код комментарии, но не перестарайтесь. Комментируя код, вы объясняете его назначение, а не пишете роман.

3.3. Комментарии в коде

Комментарии. Современные языки программирования, включая языки из Visual Studio .NET, позволяют добавлять в код программ комментарии. Комментарии не компилируются вместе с программой и не выполняются при ее выполнении. Они позволяют добавлять в программу замечания, поясняющие ее работу и назначение. Комментарии помогают вспомнить, что делают отдельные части программ, и позволяют другим программистам легче разбираться в написанном вами коде.

Комментарии следует добавлять в код всегда. Наличие комментариев — это признак опытности программиста и хорошего стиля программирования. Хороший программист всегда выделит время на документирование и объяснение написанного им кода. Если вы изменяете или добавляете какие-то функции в программу, добавьте в нее комментарии. Кроме того, добавляйте комментарии к трудным для понимания частям программы или к частям, выполняющим сложные или редко используемые действия или вычисления.

Только в Visual Basic .NET комментарии обозначаются апострофом. В языках в C# и J# строки комментариев начинаются с двух символов косой черты (//).

В Visual Basic .NET комментарий в строке начинается с символа апострофа ('). Любая строка в коде, начинающаяся с апострофа, считается комментарием и не будет компилироваться вместе с программой и выполняться.

У комментариев есть еще одна важная функция кроме документирования программ. Поскольку строки комментариев не компилируются и не выполняются, можно помечать строки кода как строки комментариев, чтобы эти строки не выполнялись. Этот прием называется «закомментировать строку кода». С помощью закомментирования фрагментов кода можно разыскивать ошибки в программах. Если в вашей программе есть ошибка, можете по очереди закомментировать отдельные строки или целые блоки кода, пока ошибка не перестанет появляться. Ошибка, скорее всего, будет находиться в

строке, которую вы закомментировали последней, прежде чем ошибка исчезла. Удобно, правда?

Псевдокод в качестве комментариев. С помощью комментариев можно описать структуру программы. Если вы уже написали псевдокод, описывающий работу вашей программы, воспользуйтесь этим псевдокодом. Скопируйте и вставьте псевдокод в программу и поместите апостроф в начале каждой строки псевдокода, чтобы превратить его в комментарии. А теперь напишите настоящий код на Visual Basic .NET под каждой строкой псевдокода. У вас получится программа, которая не только работает, но и уже документирована!

Вот пример, основанный на псевдокоде вычисления средней цены бензина, который был описан в предыдущем параграфе. Обратите внимание, что псевдокод образует комментарии к программе.

```
'Просуммировать все деньги, которые мы
'потратили на бензин, и записать эту сумму в
'переменную (TotalRub).
Dim TotalRub As Double
TotalRub = 800 + 1700 + 1350

'Найти общее количество купленного бензина и
'записать это количество в другую переменную
'(TotalLiter).
Dim TotalLiter As Double
TotalLiter = 50 + 100 + 75
'Разделить значение первой переменной
'(TotalRub) на значение второй переменной
'(TotalLiter). Результат деления записать в
'третью переменную (RubPerLiter).
Dim RubPerLiter As Double
RubPerLiter = TotalRub / TotalLiter
'Вывести сообщение, в котором указана средняя
'цена бензина.
MessageBox.Show (RubPerLiter)
```

Отступы и пробельные символы. Хотя вы, возможно, поняли не весь показанный выше код на

По умолчанию Visual Basic .NET автоматически делает отступы в начале строк кода, зависящие от кода в этих строках. Хотя автоматический отступ можно отключить, лучше его все же использовать.

языке Visual Basic .NET, мы хотим, чтобы вы обратили внимание на пару вещей. Во-первых, все комментарии начинаются с апострофа. Кроме того, комментарии выделяются в программах зеленым цветом. Это позволяет легко замечать их в коде. Во-вторых, обратите внимание на пустую строку перед каждым комментарием. Эти пустые строки делят комментарии и код на части, выполняющие отдельные действия. Поделенную на такие части программу проще читать и понимать.

Во многих случаях Visual Basic .NET автоматически добавляет пропуски и отступы, чтобы код было легче читать. Но если хотите, вы можете добавлять дополнительные пропуски и отступы. Они облегчают чтение и понимание кода и комментариев. Visual Basic .NET и другие языки .NET не обращают внимания на пустые строки и отступы. Они игнорируются при компиляции программы.

Microsoft-CD

Задания для самостоятельного выполнения

3.2. В системе программирования Visual Basic .NET создать проект «Цена бензина», описанный в параграфе. Готовый проект содержится в самораспаковывающемся архиве Цена_бензина.exe.

3.3. В системе программирования Visual Basic .NET создать проект «Цвет рыбок». В текстовые поля вводятся количества красных, синих, желтых и зеленых рыбок, а также их общее количество, и создается событийная процедура, вычисляющая процент рыбок каждого цвета и выводящая эти проценты в текстовые поля. Перевести текст на надписях и кнопках, а также комментарии в программном коде с английского языка на русский. Заготовка проекта содержится в самораспаковывающемся архиве Цвет_рыбок.exe.

Тест по теме «Алгоритмы и программы»

1. 

Какие операторы в языках программирования используются для многократного повторения одних и тех же действий?

- ❍ *Операторы сравнения*
- ❍ *Операторы цикла*
- ❍ *Операторы ветвления*
- ❍ *Операторы присваивания*

2. 

Псевдокод — это

- ❍ *Язык программирования*
- ❍ *Алгоритм, записанный на естественном языке*
- ❍ *Комментарии в программе*
- ❍ *Программа на машинном языке*

3. 

Комментарии, отступы и интервалы необходимы в программе

- ❍ *Человеку для лучшего понимания программы*
- ❍ *Компьютеру для правильного выполнения программы*
- ❍ *И человеку, и компьютеру*
- ❍ *Ни человеку, ни компьютеру*

4. 

Операторы ветвления используются для

- ❍ *Многократного выполнения одной и той же серии команд*
- ❍ *Сравнения значения переменной с числом или со значением другой переменной*
- ❍ *Присваивания переменной значения*
- ❍ *Выполнения одной серии команд, если условие выполняется, и другой серии, если условие не выполняется*

Глава 4

Формы и элементы управления

Microsoft В 1985 году была разработана для персональных компьютеров операционная система с графическим интерфейсом Windows 1.0. Пользователь получил возможность управлять компьютером с помощью щелчков мышью по пиктограммам и пунктам меню. При желании пользователь переходил в режим MS-DOS, где вводил команды с клавиатуры в командной строке.

4.1. Форма — основа графического интерфейса

Приложение Windows (Windows Application) имеет графический пользовательский интерфейс, позволяющий пользователю взаимодействовать с приложением. Пользовательский интерфейс есть не у всех программ — некоторые программы выполняют все, что нужно, сами или работают в фоновом режиме, не требуя от пользователя никаких действий после запуска.

Когда вы создаете приложение Windows на Visual Basic, Visual Studio автоматически добавляет в ваш проект файл формы `Form1.vb`. В эту форму можно добавлять элементы управления, например кнопки, надписи, текстовые поля, выпадающие списки и т. д. Эти элементы управления позволяют пользователю взаимодействовать с программой. Предположим, что в приложении есть форма для описания проколотых шин, в которой задается информация об этих шинах. Вот примерный список элементов управления, которые вам понадобятся, с описанием их назначения:

- Текстовое поле, в котором указывается марка автомобиля.
- Группа переключателей, с помощью которых можно указать, какая шина проколота (правая передняя, левая передняя, правая задняя, левая задняя).
- Флажок, который устанавливается, если замена нужна экстренно.
- Выпадающий список, в котором можно выбрать нужный размер шины для замены.
- Кнопка, при нажатии которой выполняется поиск подходящих шин для замены.

- Текстовое поле, в котором отображается информация о подходящих найденных шинах, включая их производителя, модель, размер, количество в запасе и цену.

Таким образом, на форме будут два текстовых поля, четыре переключателя, выпадающий список, флажок и кнопка. Их можно расположить на форме множеством разных способов — именно здесь и начинается самое интересное. Форма — это то же самое, что холст для художника, а элементы управления — то же, что краски. Но нужно быть аккуратным. Форма должна выполнять задачу, для выполнения которой она предназначена. Она должна соответствовать требованиям программы. Ей должно быть легко пользоваться, и на ней не должно быть ничего лишнего. Создавая форму, нужно сначала решить: для чего нужна эта форма? Что она должна делать?

Форма должна аккуратно выглядеть. На ней не должно быть кучи разноцветных элементов или раздражающих визуальных эффектов, вроде мерцающего текста — это будет отвлекать пользователя. Элементы управления должны располагаться в четком и логичном порядке. Нужно продумать расположение элементов, в которые пользователь будет вводить информацию, работая с программой. Иногда нужно предупреждать пользователя, что он не ввел всю нужную информацию или ввел неверную информацию. Создавая формы, попытайтесь представить себя пользователем и подумайте, как можно будет работать с вашей программой.

Создавая форму, нужно помнить, что все приложения Windows выглядят более или менее похоже. Пользователь будет ожидать, что ваше приложение соответствует определенным стандартам и ведет себя соответственно. Например, первое меню в строке меню приложений Windows — это обычно меню *Файл*, в котором обычно есть пункт *Выход*, при выборе которого программа завершает работу.

Пользователь придет в замешательство и останется недовольным, если, например, при щелчке по кнопке с надписью *Вычислить результат* цвет фона окна станет красным, а текст всех надписей окажется написанным по-французски.

Как вы уже знаете, когда вы создаете новое приложение Windows, Visual Studio автоматически добавляет в него форму `Form1.vb`. Это основная, или начальная, форма приложения. Именно ее пользователь видит при запуске программы. С основной формы пользователь может обратиться к любой другой части приложения с помощью расположенной на ней строки меню или панели инструментов.

Выбирая пункты меню или нажимая кнопки в основной форме, пользователь может открывать другие формы. Эти формы могут предназначаться для ввода или отображения информации или для взаимодействия с программой другими способами. Обычно у каждой формы есть конкретное предназначение, заключающееся, например, в отображении результатов игры, вводе данных или настройке программы. После того, как пользователь введет нужную информацию или просмотрит показанную, он обычно закрывает форму и возвращается к основной форме. Хотя Visual Studio создает только форму `Form1`, в проект можно добавить столько других форм, сколько вам понадобится.

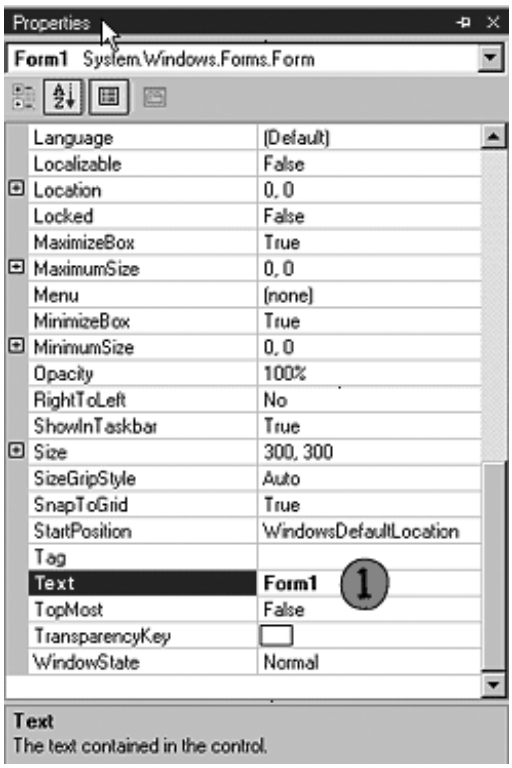
4.2. Свойства формы

Теперь мы можем начинать создание проекта «Замена шины». Сначала необходимо создать новую форму, затем разместить на ней элементы управления и задать значения их свойств, и, наконец, создать программный код обработчиков событий, на которые реагируют элементы управления.

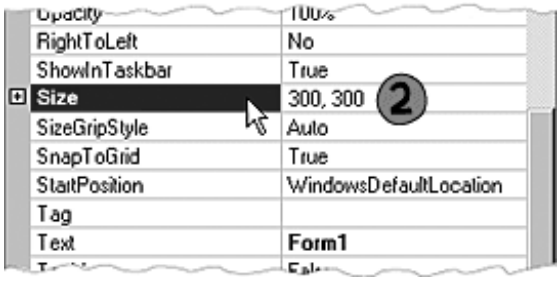
Создание формы проекта и изменение ее свойств. Когда вы создаете новое приложение Windows, форма `Form1` создается автоматически. Ее размер по умолчанию — 300 × 300 пикселей. Значение свойств, установленных по умолчанию, можно легко узнать в окне *Свойства* (*Properties*). Кроме того, в окне свойств можно и изменять значения свойств.

Создав проект, один раз щелкните левой кнопкой мыши по форме `Form1`, чтобы выбрать ее. Нажмите клавишу {F4}, чтобы открыть окно *Свойства*.

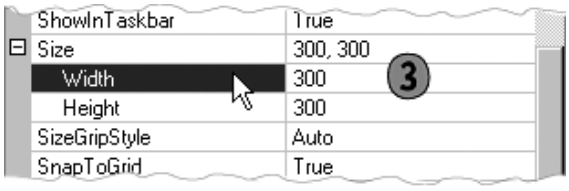
1. Поскольку сейчас выбрана форма `Form1`, то в окне *Свойства* отображаются свойства этой формы. Заметьте, что по умолчанию подсвечивается свойство `Text`.



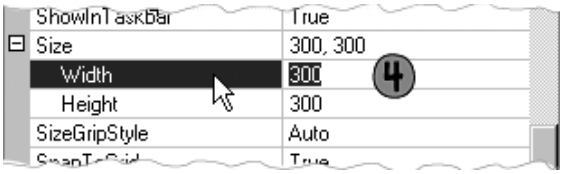
2. Прокрутите вниз список свойств формы и найдите в нем свойство `Size`. Обратите внимание на значок «плюс» рядом со свойством `Size`. Это показывает, что свойство `Size` можно развернуть, чтобы добраться до других свойств.



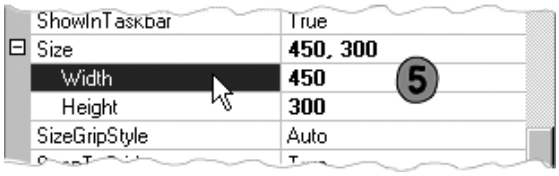
3. Щелкните по значку «плюс» рядом со свойством `Size`. Откроются свойства `Height` и `Width`. Значение свойства `Height` (высота) формы `Form1` равно 300, и значение свойства `Width` (ширина) тоже равно 300. Давайте изменим размер формы `Form1`.



4. Выполните двойной щелчок по свойству `Width`, чтобы получить возможность изменить его значение (300). Нажмите клавишу {Backspace}, чтобы очистить значение свойства.



5. Введите число 450. Нажмите клавишу {Enter}. Вот и все! Вы только что изменили ширину формы — значение ее свойства `Width`. Посмотрите, как изменилась ширина формы на экране, когда вы нажали клавишу {Enter}. Кроме того, теперь в свойстве `Width` в окне *Свойства* показано новое значение.



А теперь давайте изменим значение свойства `BackColor` формы `Form1`. Свойство `BackColor` — это цвет фона формы и элементов управления, расположенных на ней. В окне *Свойства* прокрутите список, чтобы найти свойство `BackColor`. Щелкните по значению `Control`. Справа появится значок со стрелкой вниз. Щелкните по этому значку. Откроется окно с палитрой. Перейдите в этом окне на вкладку *Custom*. Выполните двойной щелчок по любому квадратику с цветом, чтобы выбрать этот цвет. А теперь посмотрите на форму!

Постройте решение и запустите проект. Посмотрите, как теперь выглядит форма! Кроме того, теперь она прямоугольная, а не квадратная!

Теперь вы знаете, как изменять значения свойств формы с помощью окна *Свойства*. Просмотрите список свойств формы `Form1` и познакомьтесь с ними. Окно *Свойства* позволяет вам изменять значения свойств формы на этапе проектирования, т. е. во время создания формы. Как вы вскоре увидите, окно *Свойства* позволяет изменять значения свойств не только формы, но и кнопок, текстовых полей и других элементов управления.

Microsoft-CD

Задания для самостоятельного выполнения

4.1. В системе программирования Visual Basic .NET создать форму для проекта «Замена шины», описанную в параграфе.

4.3. Элементы управления и их свойства

Вы уже знаете, как поместить на форму кнопки. Как вы наверняка уже видели, в окне *Область элементов* доступно множество других элементов управления. Рассмотрим, как использовать самые распространенные из них и как задавать значения их наиболее полезных свойств. Начнем с надписей.

Надписи. Надпись (`Label`) — это элемент управления, позволяющий размещать на форме текст. Этот элемент часто используется для сообщения пользователю каких-то сведений или для указания порядка ввода информации в форме.

Откройте окно *Область элементов* и выполните двойной щелчок по элементу управления `Label`. Этот элемент будет добавлен на форму. Перетащите его в подходящее место. Для этого подведите к нему курсор мыши, нажмите левую кнопку мыши и, удерживая ее нажатой, перемещайте мышь. Выберите элемент и откройте окно *Свойства*. В этом окне теперь отображается список свойств элемента `Label`. Прокрутите список, чтобы добраться до свойства `Text`, двойным щелчком по нему выделите текст и удалите его, нажав клавишу {Backspace}. Установите значение *Введите марку автомобиля*.

Текстовые поля. Текстовые поля (`TextBox`) можно использовать для вывода информации на экран и для получения информации от пользователя. В окне *Область элементов* сделайте двойной щелчок по элементу `TextBox`. На форму будет добавлен элемент `TextBox1`. Перетащите этот элемент так, что-

бы он находился справа от надписи Label1. В окне *Свойства* найдите свойство Text, выполните по нему двойной щелчок и нажмите клавишу {Backspace}, чтобы удалить текст. Установите значение False свойства ReadOnly, чтобы пользователь смог вводить текст в это текстовое поле. (Значение False установлено по умолчанию.)

Текстовые поля часто используются для вывода больших объемов информации. Если текст действительно длинный, вы, вероятно, захотите добавить к текстовому полю полосы прокрутки. Добавьте на форму еще один элемент TextBox. Установите значение True его свойства Multiline. Задайте значение 100 пикселей его свойства Height. Для свойства Width тоже задайте значение 100 пикселей. Если для свойства Multiline установлено значение False, вам не удастся изменить значение свойства Height. Установите значение Vertical свойства Scrollbars. Установите значение True свойства ReadOnly, чтобы пользователь не мог вводить текст в это текстовое поле.

Переключатели. Переключатели (RadioButton) позволяют пользователю выбирать один вариант из нескольких предложенных. Пользователь может включить только один переключатель, т. е. один вариант из нескольких. В окне *Область элементов* выполните двойной щелчок по элементу RadioButton. На форму будет добавлен элемент RadioButton1 — это один переключатель (один вариант).

Выберите элемент RadioButton1 на форме, щелкните по нему правой кнопкой и выберите команду *Копировать* из открывшегося меню. Щелкните в пустом месте на форме Form1, щелкните правой кнопкой мыши и выберите из меню команду *Вставить*. На форме появится еще один переключатель RadioButton2. Повторно вызывая команду *Вставить*, создайте переключатели RadioButton3 и RadioButton4.

В языках J# и C# формы создаются точно так же, как и в Visual Basic .NET. Это несложно благодаря Visual Studio .NET. Все, что нужно — это открыть окно *Область элементов* и перетащить нужные элементы на форму. Затем нужно открыть окно *Свойства* и настроить свойства элементов.

Выберите RadioButton1 и откройте окно *Свойства*. Прокрутите список свойств и найдите в нем свойство Text. Измените его значение с RadioButton1 на Левое,переднее. Измените значения свойства Text других переключателей: у RadioButton2 — на Левое,заднее, у RadioButton3 — на Правое,переднее, у RadioButton4 — на Правое,заднее.

Флажки. Флажки (CheckBox) тоже позволяют пользователю делать выбор, но в отличие от переключателей из группы флажков можно выбирать сразу несколько вариантов (в том числе все). В окне *Область элементов* выполните двойной щелчок по элементу CheckBox. На форму будет добавлен элемент CheckBox1. Откройте окно *Свойства* и измените значение свойства Text на Экстренно.

Выпадающий список. Выпадающий список (ComboBox) предоставляет пользователю набор пунктов для выбора. Когда пользователь выбирает один из пунктов списка, выбранный пункт отображается в верхней строке. В окне *Область элементов* выполните двойной щелчок по элементу ComboBox. На форму будет добавлен элемент ComboBox1.

Откройте окно *Свойства* и измените значение свойства Text на Выберите размер шины. Найдите в списке свойств свойство Items. Щелкните по значению (Коллекция). Рядом с ним появится кнопка с многоточием. Щелкните по этой кнопке, откроется окно, в котором можно ввести список пунктов, которые будут отображаться в выпадающем списке. Каждый пункт списка должен начинаться с новой строки. Вводя каждый пункт, нажимайте клавишу {Enter}, чтобы создать новую строку. Введите в окне число 12, нажмите {Enter}, введите 13, нажмите {Enter}, введите 14, нажмите {Enter} и введите 15. Нажмите кнопку *OK*, чтобы закрыть окно.

Кнопки. Добавьте на форму кнопку Button. Установите для ее свойства Text значение Заменить шину. Измените цвет фона (свойство BackColor) на

У формы и многих элементов управления есть одинаковые по смыслу свойства. Вот некоторые распространенные свойства:

- `BackColor` — цвет фона;
- `Enabled` — доступ к элементу;
- `ForeColor` — цвет;
- `Location` — место, в котором элемент находится;
- `Name` — имя;
- `Size` (`Height`, `Width`) — размер (высота, ширина);
- `Text` — текст или надпись;
- `Visible` — видим элемент или невидим.

красный. Найдите в списке свойств свойство `Font`. Щелкните по этому свойству, чтобы появилась кнопка с многоточием. Щелкните по этой кнопке, чтобы открыть окно выбора шрифта. В этом окне измените начертание шрифта на полужирное: свойству `Bold` присвойте значение `True`. Нажмите кнопку *OK*, чтобы закрыть окно выбора шрифта.

Постройте решение и запустите проект. Попробуйте щелкать по разным элементам управления в окне запустившегося приложения. Что произойдет, если щелкнуть по выпадающему списку? Что будет, если выбрать одну из позиций переключателя, какую-то другую позицию, установить или сбросить флажок? Попробуйте ввести текст в текстовое поле справа от надписи *Введите марку автомобиля*. Обратите внимание на то, что второе текстовое поле серое и вы не можете ввести в него текст. Нажмите кнопку со значком  в верхнем правом углу окна, чтобы закрыть ваше приложение.

Поэкспериментируйте с разными свойствами элементов управления, которые вы поместили на форму. Можете перемещать элементы по форме, чтобы она выглядела аккуратнее и была удобнее в использовании. Можете добавить надписи, поясняющие пользователю назначение каждого элемента управления и дающие указания, какую информацию нужно вводить. Как, по вашему мнению, должны располагаться элементы на форме? Какой элемент должен быть первым? Какой должен быть последним?

Вы только что создали форму для проекта «Замена шины». Создавая ее, вы познакомились с несколькими видами элементов управления и некоторыми их свойствами. Чтобы форма заработала, нужно добавить к ней код на Visual Basic .NET.

Microsoft-CD

Задания для самостоятельного выполнения

4.2. В системе программирования Visual Basic .NET на форму для проекта «Замена шины» поместите элементы управления, описанные в параграфе.

4.4. Генерация событий

События генерируются в результате действий пользователя. Например, события генерируются, когда пользователь нажимает кнопку, выбирает пункт в выпадающем списке или изменяет текст в текстовом поле. Щелчки или двойные щелчки по элементам управления, перемещение курсора на эти элементы управления или с них тоже генерируют события. Когда происходят эти события, исполняется код Visual Basic .NET, связанный с ними. Именно код Visual Basic .NET заставляет программу делать то, что она должна делать.

Когда вы создавали приложение «Привет, Мир», вы выполнили двойной щелчок по кнопке и открылось окно редактора кода, в котором вы ввели код в обработчик события `Button1_Click`. Событие `Button1_Click` — это событие по умолчанию для кнопки (нажатие этой кнопки). У каждого элемента управления есть событие по умолчанию. Когда вы работаете в редакторе форм, двойной щелчок по элементу управления создает обработчик события по умолчанию для этого элемента (если этот обработчик не создавался ранее) и открывает этот обработчик в редакторе кода.

Например:

- Для кнопки событие по умолчанию — `Click` (щелчок по кнопке).
- Для текстового поля событие по умолчанию — `TextChanged` (изменение текста).
- Для позиции переключателя событие по умолчанию — `CheckedChanged` (включение/выключение).
- Для флажка событие по умолчанию — `CheckedChanged` (установка/сброс флажка).
- Для выпадающего списка событие по умолчанию — `SelectedIndexChanged` (выбор пункта списка).

В проекте «Замена шины» добавим код в обработчики событий по умолчанию для элементов на форме, которую мы только что создали. Этот код будет выводить на экран окно, в котором будет сообщаться тип элемента, сгенерировавшего сообщение. В редакторе форм выполните двойной щелчок по очереди по каждому элементу управления на форме. Для каждого элемента будет открываться редактор кода, в котором будет создан обработчик события по умолчанию для этого элемента. Вставьте в каждый создаваемый обработчик следующую строку кода:

```
MessageBox.Show(sender.GetType.Name)
```

Постройте решение и запустите проект. Попробуйте щелкать по разным элементам управления, изменять текст в текстовом поле и выбирать разные пункты из выпадающего списка. Каждый раз, когда какой-то элемент сгенерирует событие по умолчанию, будет выведено окно, сообщающее тип элемента, сгенерировавшего событие.

В следующих главах проект «Замена шины» будет доработан так, что в обработчиках событий будет размещаться код, выполняющий все, что должно делать это приложение. Например, при нажатии кнопки *Заменить шину* в текстовом поле будет выводиться список подходящих типов шин.

У каждого элемента управления есть множество событий, помимо события по умолчанию. У большинства элементов управления есть десятки событий. Например, у текстового поля есть такие события:

- `TextChanged` (изменение текста — событие по умолчанию);
- `Click` (щелчок по текстовому полю);
- `DoubleClick` (двойной щелчок по текстовому полю);
- `MouseEnter` (попадание курсора мыши на текстовое поле);
- `MouseLeave` (уход курсора мыши с текстового поля);

В C# и J# у элементов управления есть такие же события, как и в Visual Basic .NET. Это потому, что эти элементы управления принадлежат к платформе .NET; Visual Studio просто упрощает работу с ними.

- и множество других.

Чтобы увидеть все события, которые есть у элемента, щелкните по нему правой кнопкой и выберите из открывшегося меню команду *Просмотреть код*. Откроется окно редактора кода. Над этим окном есть два выпадающих списка. В правом списке перечислены все события элемента управления. Для событий, выделенных жирным шрифтом, уже созданы обработчики событий.

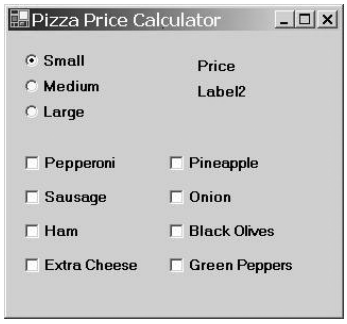
Microsoft-CD

Задания для самостоятельного выполнения

4.3. В системе программирования Visual Basic .NET в проекте «Замена шины» создать для элементов управления обработчики событий, описанные в параграфе. Готовый проект содержится в самораспаковывающемся архиве Замена_шины.exe.

4.4. В системе программирования Visual Basic .NET создать проект «Цена пиццы», позволяющий клиентам определять стоимость их заказов. Заготовка проекта, содержащая программный код проекта, хранится в архиве Цена_пиццы.exe. Создать форму, изменить надпись в заголовке формы и поместить на нее следующие элементы управления:

- 3 переключателя;
- 2 надписи;
- 8 флажков:



4.5. В системе программирования Visual Basic .NET создать проект «Крестики-нолики», позволяющий реализовать на компьютере широко известную игру «Крестики-нолики». Заготовка проекта, содержащая программный код проекта, хранится в архиве Крестики-нолики.exe. Создайте форму, измените надпись в заголовке формы и поместите на нее элементы управления так, как описано ниже:

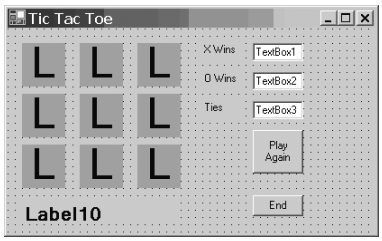
1. Щелкните в пустом месте на форме, чтобы выбрать ее. Измените значение ее свойства `Text` на `Tic Tac Toe`.
2. Создайте 9 элементов `Label` с именами от `Label1` до `Label9`. Расположите их так, чтобы элемент `Label1` был в верхнем левом углу, а `Label9` — в нижнем правом. Можно сэкономить время, создав один элемент `Label` и настроив его свойства, а затем скопировав его в буфер и вставив из него, чтобы получить восемь копий. Присвойте свойствам элемента следующие значения:

`Height`	`48`
`Width`	`48`
`Font`	`Microsoft Sans Serif, Bold, 36`
`BackColor`	**Можно не изменять**
`Text`	**Можно не изменять**

3. Создайте надпись Label10 под предыдущими девятью:

`Height`	`25`
`Width`	`175`
`Font`	`Microsoft Sans Serif, Bold, 16`
`BackColor`	**Можно не изменять**
`Text`	**Можно не изменять**

4. Создайте 3 надписи с именами от `Label11` до `Label13`. Измените значения их свойства `Text` на `X Wins`, `O Wins` и `Ties`, как показано в примере.
5. Создайте 3 текстовых поля с именами от `TextBox1` до `TextBox3`. Поместите их рядом с метками, созданными на шаге 4.
6. Создайте кнопку `Button1`. Измените значение ее свойства `Text` на `Play Again`.
7. Создайте кнопку `Button2`. Измените значение ее свойства `Text` на `End`.



Тест по теме «Формы и элементы управления»

1. 

На форму могут быть помещены

- ❍ *Программный код*
- ❍ *Элементы управления*
- ❍ *Другая форма*
- ❍ *Комментарии к программе*

2. 

Значения свойств формы и элементов управления могут быть заданы с помощью окна

- ❍ *Область элементов*
- ❍ *Конструктор*
- ❍ *Свойства*
- ❍ *Вывод*

3. 

Какой элемент управления может не только отображать текст, но и получать его от пользователя?

- ❍ *Текстовое поле*
- ❍ *Надпись*
- ❍ *Кнопка*
- ❍ *Выпадающий список*

4. 

Какую функцию выполняют события в Visual Basic .NET?

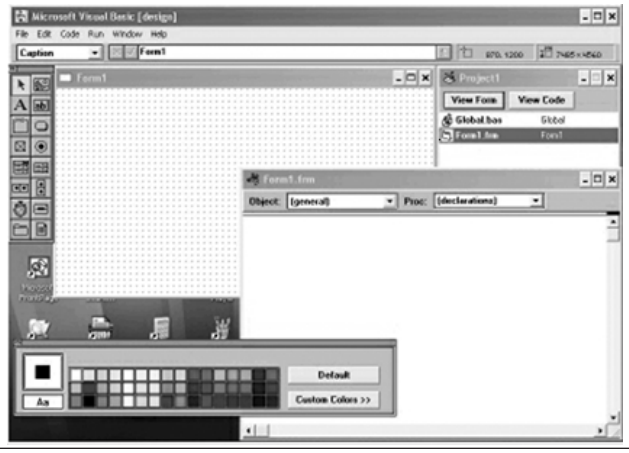
- ❍ *Осуществляют запуск проекта на выполнение*
- ❍ *Осуществляют компиляцию проекта*
- ❍ *Осуществляют построение решения*
- ❍ *Вызывают выполнение связанного с ними программного кода*

Глава 5

Свойства и методы

Microsoft

В 1991 году был создан язык объектно-ориентированного программирования Visual Basic 1.0 для персональных компьютеров. Система программирования Visual Basic 1.0 получила графический интерфейс, который позволил визуально конструировать интерфейс проектов. Уже в первой версии система программирования имела IDE (Integrated Development Environment — Интегрированная среда разработки), включавшую окна *Область элементов* (*Toolbox*), *Конструктор форм (Design), Программный код (Code)* и *Свойства (Properties)*.



С помощью окна *Свойства* задаются начальные значения свойств формы и ее элементов. Во время работы программы значения свойств изменяются с помощью кода.

5.1. С чего начинается код

Создавать формы и задавать в окне *Свойства* (*Properties*) значения свойств интересно, но это только начало! Окном *Свойства* пользоваться несложно, но оно не слишком универсальное. Что, если пользователь захочет изменить цвет фона формы, работая с программой? Что, если пользователю понадобится изменить размер или начертание шрифта? Как программа может узнать о том, установлен ли флажок и выбрана ли какая-то позиция переключателя? Поскольку у пользователей не будет доступа к окну *Свойства*, когда они запустят программу, как же все это сделать?

Ответ прост: с помощью кода на Visual Basic .NET! Можно сделать все, перечисленное выше, и много чего еще, с помощью кода. Можно написать код, который будет считывать значения свойств элементов управления и записывать в эти свойства новые значения. Это простая и мощная возможность.

Чтобы отредактировать обработчик события по умолчанию у элемента управления, выберите этот элемент на форме и выполните по нему двойной щелчок. Откроется окно редактора кода, и обработчик будет готов для редактирования. Обработчики других событий можно выбрать из выпадающего списка в верхнем правом углу окна редактора кода.

А где же находится весь этот код? Этот код пишется в редакторе кода. В большинстве случаев он находится в обработчиках событий элементов управления формы. Помните событие нажатия кнопки? Когда пользователь выполняет какое-то действие, например нажимает кнопку, код в обработчике этого события выполняется и может считывать или записывать значения свойств элементов управления. Добавляя код в обработчики событий элементов управления, вы делаете первый шаг к работающей программе. Помните форму проекта «Заменить шину», которую мы создали раньше? С ее помощью вы не могли сделать ничего, пока не добавили к ней код.

5.2. Чтение значений свойств в коде

Давайте рассмотрим примеры чтения значений свойств в коде на Visual Basic .NET. Во-первых, как

считывать значения свойств, которые задаются в окне *Свойства* (*Properties*), и, во-вторых, как считывать значения свойств, которые задают пользователи во время работы с программой?

Создайте новое приложение Windows и назовите его «Чтение свойств». Откройте окно *Область элементов* (*Toolbox*) и поместите на форму кнопку. Откройте окно *Свойства* и измените значение свойства `Text` кнопки `Button1` на `Читать текст`.

Выполните двойной щелчок по кнопке, чтобы отредактировать код обработчика нажатия кнопки. Введите в него следующий код:

```
MessageBox.Show(Button1.Text)
```

Постройте и запустите проект. Нажмите кнопку с надписью *Читать текст*. Появится окно сообщения с текстом «Читать текст» — это значение свойства `Text` элемента управления `Button1`. Код прочитал значение свойства `Text` элемента `Button1` и вывел это значение на экран в окне сообщения.

Вот еще один пример. Добавьте на форму `Form1` еще одну кнопку. С помощью окна *Свойства* измените значение ее свойства `Text` на `Размер формы`. Сделайте двойной щелчок по кнопке `Button2`, чтобы отредактировать обработчик ее нажатия. Введите в обработчик следующую строку кода:

```
MessageBox.Show(Form1.ActiveForm.Height & "," &_
Form1.ActiveForm.Width)
```

Постройте и запустите проект. Нажмите кнопку с надписью *Размер формы*. Появится окошко сообщения с текстом «300,300» (или другим, если вы изменили размер формы). Ваш код только что отобразил высоту и ширину формы `Form1`, разделив их запятой!

Значения каких еще свойств может читать код на Visual Basic .NET? На самом деле код на Visual Basic .NET может работать со всеми свойствами элемента управления, перечисленными в окне *Свойст-*

ва. Код может читать значения свойств вроде Height и Width, BackColor, ForeColor и координат X и Y. Однако значения многих свойств в окне *Свойства* задаются при создании программы и не изменяются при ее выполнении. Поэтому читать эти значения в коде не слишком полезно.

Гораздо полезнее возможность считывать значения свойств элементов, которые могут изменяться во время работы программы, например считывать текст из текстового поля или значение свойства Checked у флажка. Какой тип у автомобиля? Какая шина проколота? Каков размер шины?

Вот более близкий к практике пример. Давайте напишем код, который будет сообщать нам, установлен флажок или нет. Откройте окно *Область элементов* и добавьте на форму флажок CheckBox1. Добавьте на Form1 еще одну кнопку. Откройте окно *Свойства* и измените значение свойства Text кнопки Button3 на Проверь меня. Выполните двойной щелчок по кнопке Button3, чтобы отредактировать обработчик ее нажатия. Введите в обработчик следующую строку кода:

```
MessageBox.Show(CheckBox1.Checked)
```

Постройте и запустите проект. Нажмите кнопку с надписью *Проверь меня*. Какое сообщение будет выведено? Установите флажок и еще раз нажмите кнопку с надписью *Проверь меня*. Что теперь будет в окне сообщения? При каждом нажатии кнопки выводится окно сообщения, в котором сообщается, установлен ли флажок (т. е. выводится значение свойства Checked флажка). Если флажок установлен, будет выведено True.

И последний пример. Посмотрим, как надо считывать и выводить на экран текст, который пользователь набрал в текстовом поле.

Добавьте на форму Form1 текстовое поле. Откройте окно *Свойства* и удостоверьтесь, что установлено значение False свойства ReadOnly. Задайте

В J# и C# значения свойств элементов управления считываются так же, как и в Visual Basic .NET. Вот пример кода на J# или C#, выводящего значение свойства `Text` текстового поля:

```
{
MessageBox.Show
(textBox1.Text);
}
```

За исключением точки с запятой в конце строки и фигурных скобок, код выглядит так же, как и в Visual Basic .NET. Кроме того, в J# и C# обозначение `textBox1` начинается со строчной буквы `t`.

Microsoft-CD

для свойства `Text` значение `Печатать здесь`. Добавьте на форму `Form1` еще одну кнопку. Измените значение свойства `Text` кнопки `Button4` на `Читать текст`. Выполните двойной щелчок по этой кнопке, чтобы отредактировать код в обработчике ее нажатия. Введите следующую строку кода:

```
MessageBox.Show(TextBox1.Text)
```

Постройте и запустите проект. Выполните двойной щелчок в текстовом поле и нажмите клавишу {Backspace}, чтобы стереть текст. Введите какой-нибудь новый текст. Нажмите кнопку с надписью *Читать текст*. Что будет выведено в окне сообщения? Измените текст в текстовом поле и еще раз нажмите кнопку с надписью *Читать текст*. Что произойдет?

Эти примеры показали, как с помощью кода на Visual Basic .NET считывать значения свойств элементов управления и формы. Но это только половина возможностей, потому что в коде на Visual Basic .NET можно и задавать новые значения свойств.

Задания для самостоятельного выполнения

5.1. В системе программирования Visual Basic .NET создать проект «Чтение свойств», описанный в параграфе. Готовый проект содержится в самораспаковывающемся архиве Чтение_свойств.exe.

5.3. Присваивание значений свойствам в коде

Конечно, здорово, что мы можем считывать значения свойств с помощью кода на Visual Basic .NET. Наш код может получать информацию о действиях пользователя. Программа может выполнять фрагменты кода в ответ на действия пользователя. А верно ли обратное? Что, если нам нужно задать новые

значения нескольких свойств элементов управления во время работы программы? Например, что если нам хочется, чтобы флажок «Экстренно» всегда был установлен при запуске программы? Мы хотим, чтобы программа отображала все доступные размеры шин в текстовом поле, если пользователь нажмет кнопку с надписью *Сменить шину*. Мы хотим, чтобы список доступных размеров шин автоматически загружался в выпадающий список, когда пользователь вводит номер автомобиля.

Конечно, все это можно сделать с помощью кода на Visual Basic .NET! Присваивать значения свойствам элементов управления так же просто, как и считывать уже заданные значения. Чтобы присвоить значение свойству, в коде на Visual Basic .NET используется оператор присваивания.В коде оператор присваивания выглядит как простое равенство:

```
TextBox1.Text = "Присваивание"
```

Правая часть равенства (справа от знака «равно») вычисляется первой. Затем вычисленное значение присваивается левой части равенства (слева от знака «равно»).

Вот несколько примеров присваивания значений свойствам с помощью кода на Visual Basic .NET. Первый пример записывает текст в текстовое поле. Это прекрасный способ сообщать что-нибудь пользователю — им можно пользоваться вместо отображения отдельных окон сообщений. Создайте новое приложение Windows и назовите его «Установка свойств». Добавьте на форму кнопку и два текстовых поля. В окне *Свойства* (*Properties*) задайте для свойства `Text` кнопки `Button1` значение `Установить текст`. Очистите свойства `Text` текстовых полей `TextBox1` и `TextBox2`, выполнив по ним двойной щелчок и нажав клавишу {Backspace}. Выполните двойной щелчок по кнопке `Button1`, чтобы отредактировать обработчик ее нажатия. Добавьте в обработчик следующую строку кода:

Обработчик события `Form_Load` часто используется для инициализации программы. Инициализация устанавливает определенные значения, прежде чем форма отображается на экране. Например, можно добавить в обработчик код, устанавливающий размер формы и цвет ее фона или выбирающий определенную позицию переключателя перед отображением формы на экране.

```
TextBox2.Text = TextBox1.Text
```

Постройте и запустите проект. Введите какой-нибудь текст в `TextBox1` и нажмите кнопку с надписью *Установить текст*. Что произойдет? Как видите, текст из текстового поля `TextBox1` был «скопирован» в `TextBox2`. Код на Visual Basic .NET считывает значение свойства `Text` текстового поля `TextBox1` и присваивает считанное значение свойству `Text` текстового поля `TextBox2`.

Давайте добавим код, который будет автоматически устанавливать один флажок и сбрасывать другой, когда пользователь запускает приложение. Добавьте на форму `Form1` флажок и задайте для его свойства `Text` значение `Экстренно`. Добавьте на форму еще один флажок и задайте для его свойства `Text` значение `Обязательно`. Выполните двойной щелчок в пустом месте на форме (не на элементах управления). Откроется обработчик события `Form1_Load`. Код этого обработчика выполняется, когда форма выводится на экран при запуске программы. Добавьте в этот обработчик следующие две строки:

```
CheckBox1.Checked = True
CheckBox2.Checked = False
```

В языках J# и C# значения свойствам присваиваются точно так же, как и в Visual Basic .NET. Код, приведенный ниже, показывает, как присваивать значение свойству `Checked` флажка в C# или J#.

```
{
checkBox1.Checked =
True;
}
```

Постройте и запустите проект. Какой флажок установлен? Ваш код автоматически установил значение свойства `Checked` двух флажков при загрузке формы.

Мы знаем, что 14 — это самый распространенный размер шины. Давайте сделаем так, чтобы код автоматически выбирал 14 в выпадающем списке размера шины. Сначала добавьте выпадающий список `ComboBox1` на форму `Form1`. С помощью окна *Свойства* в списке `ComboBox1` установите для свойства `Items` значения `12`, `13`, `14`, `15` и `16`. Не забывайте нажимать {Enter}, чтобы начинать с новой строки каждый новый пункт списка. Выполните двойной щелчок по форме (не по выпадающему

списку), чтобы отредактировать код в обработчике `Form1_Load`. Добавьте в обработчик следующую строку кода:

```
ComboBox1.SelectedItem = "14"
```

Постройте и запустите проект. Какой пункт отображается в выпадающем списке? Как необходимо изменить код, чтобы в выпадающем списке автоматически выбирался пункт 13?

Microsoft-CD

Задания для самостоятельного выполнения

5.2. В системе программирования Visual Basic .NET создать проект «Установка свойств», описанный в параграфе. Готовый проект содержится в самораспаковывающемся архиве Установка_свойств.exe.

5.4. IntelliSense и точечная нотация

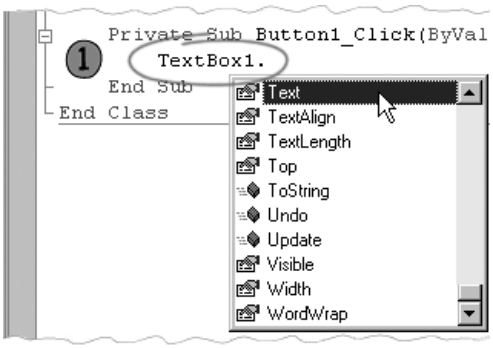
Теперь вы знаете, как с помощью Visual Basic .NET читать и записывать значения свойств форм и элементов управления. Если у каждой формы и элемента управления так много свойств, как за ними уследить? Не слишком удобно постоянно переключаться между редактором кода и окном *Свойства*, чтобы выяснить, какие свойства нам нужны. Может быть, есть более удобный способ?

Можно использовать IntelliSense. Это часть Visual Studio. IntelliSense сохраняет нам много времени при редактировании кода в окне редактора. IntelliSense знает, к какому элементу управления вы обращаетесь и какие у него есть свойства. Оно выводит список свойств и позволяет вам выбирать из него нужное. Используемое чаще всего свойство подсвечивается, когда IntelliSense отображает список. Например, для текстового поля изначально подсвечивается свойство `Text`. С помощью стрелок вверх и вниз на клавиатуре можно просматривать список и выби-

рать то свойство, которое вам требуется. При нажатии клавиши {Tab} выбранное свойство добавляется в код. Используя IntelliSense, вам не придется запоминать все свойства всех элементов управления. Кроме того, данное средство заметно сокращает потребность в наборе длинных имен на клавиатуре!

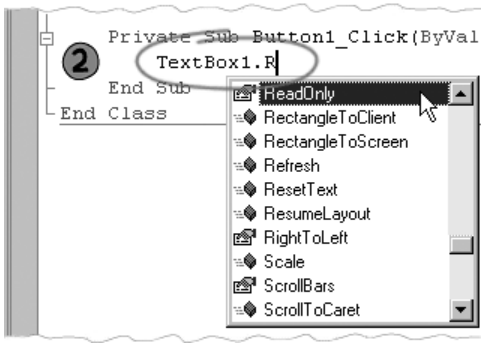
Попробуйте его использовать! Создайте новое приложение Windows и назовите его «IntelliSense». Поместите на форму текстовое поле и кнопку. Выполните двойной щелчок по кнопке, чтобы отредактировать код в обработчике нажатий на нее.

1. Введите с клавиатуры `TextBox1` и нажмите клавишу «точка» {.}. Как только вы введете точку, IntelliSense отобразит список свойств элемента `TextBox1`. Поскольку `TextBox1` — текстовое поле, этот список будет таким же, как и для любого другого текстового поля на форме. По умолчанию в списке IntelliSense подсвечивается чаще всего используемое свойство. В нашем случае по умолчанию подсвечивается свойство `Text`.

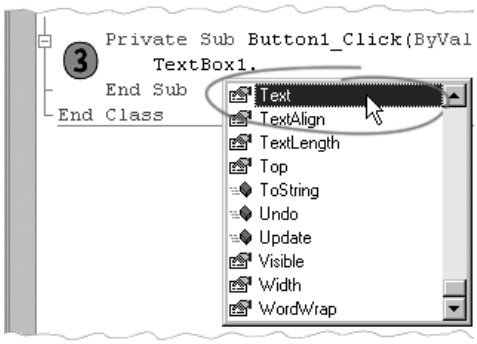


2. Теперь введите букву «R». IntelliSense прокрутит список до первого по алфавиту свойства, начинающегося с «R». По мере ввода IntelliSense ищет в списке свойство, наиболее соответствующее введенным буквам.

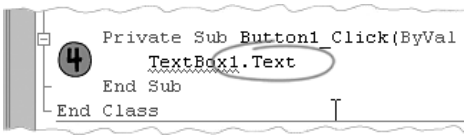
Когда свойство подсвечивается в списке IntelliSense, рядом появляется подсказка, кратко его описывающая. Эта подсказка, например, может сообщить, если свойство доступно только для чтения (`Read Only`, т. е. ему нельзя присваивать значение). Кроме того, подсказка сообщает, какие значения можно присваивать свойству, например, строку (текст) или целое число.



3. С помощью стрелки вниз прокрутите список и найдите снова свойство `Text`. Выберите свойство `Text`, чтобы оно было подсвечено.



4. Нажмите клавишу {Tab}. Посмотрите, что получилось! После `TextBox1.` в код добавлено слово `Text`.



5. Теперь введите `="Clifford"`. Код будет выглядеть так:

```
TextBox1.Text = "Clifford"
```

Создавая программы на языках C# и J# с помощью Visual Studio, тоже можно использовать IntelliSense. Это средство работает точно так же, как и в Visual Basic .NET. Просто введите имя элемента, а затем точку. IntelliSense выведет список доступных свойств и методов. Поскольку языки J#, C# и Visual Basic .NET основаны на платформе .NET, список свойств и методов элементов будет одинаковым для всех трех языков.

Правда, просто и удобно? IntelliSense, кроме того, помогает избегать ошибок при наборе текста. Это очень важно!

Вы, наверное, заметили, что каждому пункту списка IntelliSense соответствует маленький значок. Свойства помечены значками в виде руки, указывающей на лист бумаги. Методы обозначены фиолетовым «кирпичиком». (Мы скоро разберемся, что такое методы). В зависимости от того, для какого элемента управления мы просматриваем список, в нем могут быть и пункты с другими значками.

Вы обратили внимание на то, что нужно было ввести точку, прежде чем появился список IntelliSense? Это потому, что в Visual Basic .NET используется синтаксис с записью «через точку» — Dot-запись (точечная нотация). Введя имя элемента управления, например `TextBox1`, введите точку. Затем введите имя свойства. Точка соединяет имя элемента и имя свойства. Вот общая форма (синтаксис) обращения к свойству элемента:

```
Элемент.Свойство
ЭлементУправления.Свойство
```

Вот несколько примеров такой записи:

```
TextBox1.Text
CheckBox1.Checked
Form1.ActiveForm.Height
```

Обратите внимание, что в последнем примере `ActiveForm` — это свойство `Form1`, а `Height` — свойство `ActiveForm`. Поэтому для их соединения нужны две точки. IntelliSense знает, когда у свойства есть собственные свойства. Если вы не уверены, есть ли свои свойства у некоторого свойства, введите еще одну точку после имени свойства, и IntelliSense выведет список. Если IntelliSense не выводит списка, значит, либо дополнительных свойств нет и нужно удалить последнюю точку, либо вы неправильно набрали имя элемента или свойства.

Microsoft-CD

Задания для самостоятельного выполнения

5.4. В системе программирования Visual Basic .NET выполнить проект «IntelliSense», описанный в параграфе.

5.5. Методы

Наверное, вы заметили, что рядом с некоторыми пунктами в списке IntelliSense стоят фиолетовые значки в виде кирпичика. Это методы. Свойства — это характеристики элемента, например, свойства машины — цвет, количество мест, вес и мощность двигателя. Методы — это действия, которые может выполнить элемент. Например, машина может повернуть налево, остановиться или заглушить мотор. Методы не перечислены в окне *Свойства*. Их нужно вызывать в коде.

Например, у большинства элементов управления есть методы `Hide` (скрыть) и `Show` (отобразить). Если вызвать метод `Hide` элемента, этот элемент станет невидимым, хотя и останется на форме. Если вызвать метод `Show`, элемент станет видимым. Некоторые методы позволяют выполнять действия, которые обычно выполняет пользователь. Например, у кнопки есть метод `PerformClick`. Если вызвать этот метод, то будет сымитировано нажатие кнопки.

Некоторые методы требуют сообщения им определенной информации при вызове. Эта информация определяет, что метод будет делать и какие будут результаты его вызова. Информация, передаваемая методу, называется аргументами метода. У метода может быть один или несколько аргументов. Некоторые аргументы могут быть необязательными.

Например, метод `Show` окна сообщения принимает в качестве аргумента строку текста. Этот аргумент определяет, что будет выведено в окне сообщения. Вы уже использовали метод `Show` окна сообщения, хотя, вероятно, не осознавали, что вызывали метод с аргументом.

```
MessageBox.Show("Привет, Мир")
```

В этой строке кода "Привет, Мир" — это аргумент метода `Show`.

Итак, запомним следующее. Методы выполняют какие-то действия. Методы вызываются из кода, до них нельзя добраться из окна *Свойства*. Некоторые методы требуют задания определенных аргументов при вызове. Многие методы возвращают значения, которые можно использовать в коде.

Вызов методов в коде. Синтаксис вызова метода похож на синтаксис чтения или записи значения свойства. Для соединения имени элемента и метода используется точка. Некоторым методам, например `Button.Hide` и `Button.Show`, аргументы не нужны. Другим, например `MessageBox.Show`, для работы нужны аргументы. Синтаксис вызова метода следующий:

```
Элемент.Метод (Аргумент1, Аргумент2, ..., Аргумент N)
```

Помните, при вызове метода `Show` окна сообщения `MessageBox` вы задавали аргумент?

```
MessageBox.Show("Привет, Мир")
```

Вероятно, вы уже догадались, что в языках J# и C# методы вызываются точно так же, как в Visual Basic .NET. Вот пример кода на J#, удаляющего текст из текстового поля и помещающего курсор в это текстовое поле. Код на C# выглядит точно так же.

```
{
textBox1.ResetText();
textBox1.Focus();
}
```

Поработаем с методами, позволяющими скрывать и делать видимыми элементы управления, стирать текст в текстовых полях и задавать позицию курсора. Для начала создайте новое приложение Windows и назовите его «Методы». Добавьте на форму две кнопки и текстовое поле. Измените значение свойства `Text` кнопки `Button1` на `Показать`. Измените значение свойства `Text` кнопки `Button2` на `Скрыть`. В обработчик нажатия кнопки `Button1` введите следующий код:

```
TextBox1.Show()
```

В обработчик нажатия кнопки `Button2` введите следующий код:

```
TextBox1.Hide()
```

Постройте и запустите проект. Нажмите кнопку с надписью *Скрыть*, а затем — кнопку с надписью *Показать*. Каждый раз, когда вы нажимаете кнопку, программа выполняет методы элемента `TextBox`.

Добавьте на форму еще одну кнопку. Измените значение свойства `Text` кнопки `Button3` на `Сброс`. В обработчик нажатия кнопки `Button3` введите следующий код:

```
TextBox1.ResetText()
TextBox1.Focus()
```

Метод `ResetText` удаляет весь текст из текстового поля.

Метод `Focus` помещает курсор в текстовое поле.

Постройте и запустите проект. Введите в текстовое поле какой-нибудь текст и нажмите кнопку с надписью *Сброс*. Из текстового поля исчезнет весь текст, и оно будет готово к вводу нового текста.

Теперь вы можете читать и записывать значения свойств и вызывать методы в коде на Visual Basic .NET. Как вы видели, большая часть кода размеща-

ется в обработчиках событий элементов управления. Чтобы открыть код обработчика события по умолчанию, просто выполните двойной щелчок по элементу управления — откроется редактор кода с готовым к редактированию обработчиком. Чтобы прочитать или записать значение свойства или вызвать метод, введите имя элемента и точку. IntelliSense покажет список доступных свойств и методов элемента. Выберите нужное вам свойство или метод и нажмите клавишу {Tab}, чтобы ввести его в код.

Microsoft-CD

Задания для самостоятельного выполнения

5.5. В системе программирования Visual Basic .NET создать проект «Методы», описанный в параграфе. Готовый проект содержится в самораспаковывающемся архиве Методы.exe.

Тест по теме «Свойства и методы»

1. 

С помощью кода в программах

- ❍ *Можно только считывать значения свойств*
- ❍ *Можно только задавать значения свойств*
- ❍ *Можно выполнять и то, и другое*
- ❍ *Нельзя делать ни то, ни другое*

2. 

Какая строка кода определит и сообщит пользователю, выбрана ли позиция переключателя?

- ❍ `MessageBox.Show(RadioButton1.Checked)`
- ❍ `MessageBox.Show(Radio.Property)`
- ❍ `MessageBox.Show(RadioButton1.Selected)`
- ❍ `MessageBox.Show(RadioButton1.Unchecked)`

3. 

Все операторы присваивания в Visual Basic выполняются в порядке

- ❍ *Слева направо*
- ❍ *Справа налево*
- ❍ *Произвольном — это безразлично*
- ❍ *Каждый оператор по-своему*

4. 

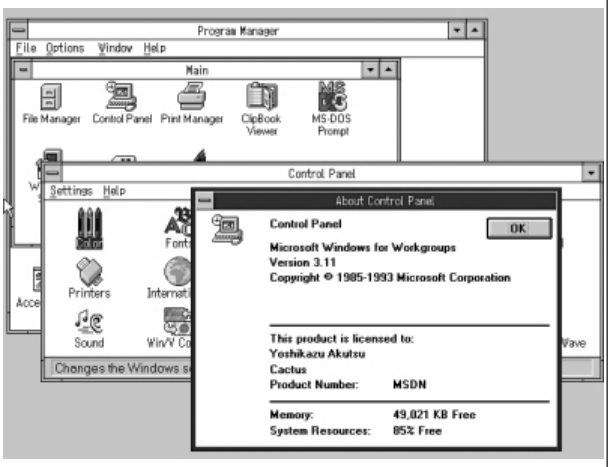
Как установить флажок с помощью кода?

- ❍ `CheckBox1.Checked = Yes`
- ❍ `CheckBox1.Unchecked = False`
- ❍ `CheckBox1.Checked = True`
- ❍ `CheckBox1.Selected = True`

Глава 6

Присваивание и переменные

Microsoft В 1992 году была создана для персональных компьютеров операционная система с графическим интерфейсом Windows 3.1, а в 1993 году выпущена операционная система Windows for Workgroups 3.11 для работы в локальной сети. Графический интерфейс этих систем позволяет пользователю управлять компьютером с помощью щелчков мышью по пиктограммам и пунктам меню. При желании пользователь мог перейти в режим командной строки MS-DOS.



6.1. Присваивание

Теперь вы знаете, как считывать и записывать значения свойств с помощью кода. Для этого служит оператор присваивания. Оператор присваивания записывает в объект значение другого объекта. В коде оператор присваивания выглядит как простое равенство:

```
TextBox1.Text = "Присваивание"
```

Напомним, что операторы присваивания вычисляются в порядке справа налево. Первой вычисляется часть справа от знака «равно». Потом левой части оператора (слева от знака «равно») присваивается вычисленное значение части справа от знака «равно». В приведенном выше операторе присваивания первым вычисляется значение `Присваивание`. Затем это значение помещается в свойство `TextBox1.Text`.

Кроме того, нужно помнить, что значение в правой части оператора присваивания должно иметь тип (текст, число, цвет и т. д.), согласующийся с типом объекта из левой части. Вы поймете, что это значит, рассмотрев следующий пример кода:

```
TextBox1.Text = "Текст"
```

Пока что мы называли текстовые переменные «текстом». В платформе .NET все текстовые переменные называются строками, т. е. имеют тип `String`. Если вы используете код для записи значения в строковую переменную, не забывайте помещать текст в кавычки, например:

```
Button1.Text = _
"Щелкнуть здесь"
```

Здесь слева от знака «равно» размещено свойство `Text` элемента `TextBox1`, которое может быть только текстом, поэтому справа от знака «равно» должен быть тоже текст. Если справа будет не текст, а что-то другое, то при попытке построить проект произойдет ошибка. Поскольку в нашем случае `"Текст"` — это текст, т. е. именно то, что может находиться в свойстве `Text`, ошибки не происходит.

Следующий код не будет компилироваться, поскольку мы пытаемся поместить в текстовое свойство `Text` значение цвета, `ForeColor`:

```
TextBox1.Text = TextBox1.ForeColor 'ошибка
```

Следующий код правильный, поскольку правая часть равенства — это целое число, а в свойство `Height` можно записывать только целые числа:

```
TextBox1.Height = 200
```

Следующий код неправильный, поскольку пытается записать в свойство `Height` дробное число:

```
TextBox1.Height = 200.5  'ошибка
```

Задания для самостоятельного выполнения

Microsoft-CD

6.1. В системе программирования Visual Basic .NET создать проект «Прыжок лягушки», осуществляющий прыжки слова "Лягушка" из крайнего левого текстового поля в последовательность текстовых полей. Заготовка проекта, содержащая графический интерфейс проекта, хранится в архиве Прыжок_лягушки.exe. Создать обработчики событий щелчков по кнопкам с надписями *Прыжок*.

6.2. Переменные

Переменные используются в коде для хранения информации, которая может в дальнейшем понадобиться программе. Переменные в коде похожи на переменные в математике. Они хранят какие-то значения. В любой момент времени в переменную можно записать новое значение.

В коде переменные используются для запоминания информации. Если бы в коде нельзя было использовать переменные, вашей программе пришлось бы запрашивать у вас информацию каждый раз, когда она была бы нужна. Представьте, что вашей программе нужно было бы три раза использовать возраст пользователя! Вы наверняка бали бы недовольны, если бы она спросила у вас возраст три раза:

```
Введите ваш возраст.
Введите ваш возраст.
Введите ваш возраст.
```

Программе, использующей переменные, достаточно будет задать вопрос только один раз. Ответ будет сохранен в переменной, которая и будет использоваться в коде каждый раз, когда эта информация понадобится программе.

Как вы уже видели, переменные позволяют сэкономить время. Они уменьшают количество информации, которую должен вводить пользователь. Кроме того, они уменьшают количество ошибок. Если бы вам пришлось раз за разом вводить длинное число, вы бы рано или поздно допустили ошибку.

Переменные используются для хранения не только введенной пользователем информации, но и результатов вычислений в программе. Переменные используются также для выполнения сравнений, по результатам которых определяется, что дальше должна делать программа, и для пересчета, например, количества повторений какого-то действия.

Знание того, как сохранять информацию в переменных, является базовым в программировании. Переменные используются во всех языках программирования. Они совершенно необходимы для выполнения вычислений определенных типов и некоторых действий в программе.

В платформе .NET есть несколько основных типов переменных, которые иногда называют базовыми. Вы уже использовали тип `String`, записывая

Поскольку C# и J# — это тоже языки .NET, в них используются те же базовые типы, что и в Visual Basic .NET, — строки, целые числа, дробные числа, булевы (логические) значения и другие.

значения в свойство `Text`. Вы использовали тип `Integer` (целые числа), записывая значения в свойство `Height` форм. Вы использовали тип `Boolean` (булевы логические величины, `True` или `False`), например, записывая значения в свойство `Visible`.

Чаще всего используются такие базовые типы: `String` (текстовые строки), `Integer` (целые числа), `Single` (дробные числа) и `Boolean` (`True` или `False`). Эти типы доступны во всех языках программирования .NET, включая Visual Basic .NET, C# и J#.

Вот примеры допустимых значений для каждого из этих типов:

Когда вы выбираете свойство или метод в списке IntelliSense, выводится подсказка к подсвеченному пункту списка. В этой подсказке указывается тип свойства или типы аргументов метода.

Тип	Значение	Комментарий
`String`	`"Hello"`	Текст должен быть в кавычках
`Integer`	`123`	Целые числа (без знаков после запятой)
`Single`	`55.12`	Дробные числа
`Boolean`	`True`	Может быть только `True` или `False`

Вот примеры присваивания значений свойствам разных типов:

```
TextBox1.Text = "Текст"
TextBox1.Visible = True
TextBox1.Width = 1000
```

Другие типы переменных. В Visual Basic .NET можно объявлять переменные множества разных типов. Кроме базовых типов (`String`, `Integer`, `Boolean`, `Single` и других), есть и более сложные типы. Эти типы встроены в платформу .NET и могут использоваться всеми языками .NET. Многие из этих типов — свойства так называемых системных классов. Системные классы (`System classes`) содержат код, обеспечивающий работу языков .NET. Для работы с языками .NET следует уметь использовать системные классы.

Пока что вам не нужно разбираться в классе System. Время от времени мы будем рассматривать фрагменты кода, использующие свойства или методы системных классов, поэтому неплохо, если вы сможете их узнавать. Рассмотрим пример, использующий системные классы для установки цвета фона формы.

Создайте приложение «Цвет» и добавьте на форму Form1 кнопку. Измените значение ее свойства Text на Установить цвет. Выполните двойной щелчок по этой кнопке, чтобы отредактировать обработчик нажатия на нее.

Добавьте в обработчик следующий код:

```
Dim MyColor As System.Drawing.Color
MyColor = System.Drawing.Color.Blue
Form1.ActiveForm.BackColor = MyColor
```

Постройте и запустите ваше приложение. Нажмите кнопку Button1. Мы объявили в коде переменную MyColor типа System.Drawing.Color. Этой переменной было присвоено значение System.Drawing.Color.Blue — один из определенных в системе цветов. Потом свойству BackColor формы Form1 было присвоено значение переменной MyColor. Обратите внимание, что свойству BackColor нужно присваивать значения типа System.Drawing.Color.

Можете попробовать изучить класс System самостоятельно. Чтобы сделать это, проще всего набрать System. (не забудьте точку) в окне редактора кода. IntelliSense выведет список всех методов и свойств системных классов. Выберите из списка свойство или метод и введите еще одну точку, чтобы увидеть, есть ли у него собственные свойства или методы. В этом списке много чего интересного!

Microsoft-CD

Задания для самостоятельного выполнения

6.2. В системе программирования Visual Basic .NET создать проект «Цвет», описанный в параграфе. Готовый проект хранится в архиве Цвет.exe.

6.3. Объявление переменных

Перед использованием переменной в программе, эту переменную нужно объявить. Чтобы объявить переменную, нужно написать строку кода, в которой указывается имя этой переменной и ее тип. Все переменные в программах на Visual Basic .NET нужно объявлять. Даже если бы не было такого требования синтаксиса, переменные объявлять полезно, поскольку это позволяет вам продумать применение переменной и выбрать для нее правильный тип. Объявление переменных ускоряет компиляцию программы и повышает эффективность ее работы. Кроме того, оно предотвращает ошибки, связанные с неправильным написанием имен переменных в коде.

Оператор объявления переменных. Чтобы объявить переменную в коде на Visual Basic .NET, используется оператор Dim. Синтаксис этого оператора следующий:

```
Dim ИмяПеременной As ТипПеременной
```

Вот несколько примеров объявления переменных разных типов:

```
Dim MyName As String
Dim MyWeight As Integer
Dim NoBrainer As Boolean
Dim DVDPrice As Single
```

В C# и J# переменные объявляются немного по-другому. Сначала пишется тип переменной, а потом ее имя. Кроме того, типы называются тоже по-другому. Например, `int` используется вместо `Integer`, а `decimal` — вместо `Single`.

Первая переменная, `MyName`, объявлена как строковая (тип `String`). Это значит, что в переменной `MyName` можно хранить только строки текста. Вторая переменная, `MyWeight`, объявлена как целочисленная (тип `Integer`). В ней можно хранить целые числа. Переменная `NoBrainer` объявлена как логическая (тип `Boolean`), а переменная `DVDPrice` — как дробное число (тип `Single`).

Важно выбирать для переменных удобные имена. Имена переменных должны быть короткими, но понятными. Используйте в них аббревиатуры толь-

ко при необходимости. Многие программисты используют стиль оформления имен, при котором каждое слово в имени переменной начинается с большой буквы. Например: `CamelCase`, `MyName`, `WinnebagoTopSpeed`, `LocalSpeedLimit`. В Visual Basic первая буква имени автоматически делается большой. В J# и C# она остается обычной.

В коде на Visual Basic .NET можно объявить сколько угодно переменных. Можно объявлять переменные любого типа, позволенного платформой .NET. Рекомендуется объявлять все нужные нам переменные в начале кода. Таким образом, мы будем знать, где искать объявления переменных и легко сможем определить, какой тип у каждой переменной. Кроме того, наш код будет аккуратным и хорошо организованным. Если нам понадобится создать еще одну переменную, мы вернемся в начало кода и объявим ее рядом с остальными переменными.

Локальные и глобальные переменные. Переменные можно объявлять как глобальные или локальные. Пока что мы писали только код внутри обработчиков событий элементов управления, например, код, обрабатывающий нажатие на кнопку. Переменные, которые мы объявим в обработчике, будут локальными. Значения локальных переменных можно считывать или записывать только в коде обработчика, в котором они объявлены. К ним нельзя обратиться из другого места в коде формы (например, из другого обработчика). Значения локальных переменных сохраняются только во время выполнения кода обработчика. Когда обработчик заканчивает свое выполнение, значения его локальных переменных теряются. Пример локальной переменной — переменная, объявленная в обработчике нажатия на кнопку.

Чтобы объявить локальные переменные в обработчике, поместите оператор Dim в начало кода обработчика.

Пример:

```
Private Sub Button1_Click(ByVal sender As_
System.Object, ByVal e As System.EventArgs)_
Handles Button1.Click
Dim MyName As String
MyName = "Bill"
TextBox1.Text = MyName
End Sub
```

До сих пор мы не использовали никаких глобальных переменных. Значения глобальных переменных можно считывать и записывать из всего кода формы. Доступ к ним есть у всех обработчиков событий и из других мест кода формы. Значения глобальных переменных сохраняются, пока форма остается открытой. Пример глобальной переменной — переменная-счетчик, подсчитывающая, сколько раз были нажаты все кнопки на форме.

Чтобы объявить глобальную переменную, которую может использовать весь код формы, нужно поместить оператор `Dim` в тело формы. Лучше всего объявлять глобальные переменные в коде формы сразу после такой строки:

```
Windows Form Designer generated code
```

Эта строка кода заключена в рамку, и слева от нее находится значок «плюс». Объявляйте свои переменные сразу после этой строки. Посмотрите на пример:

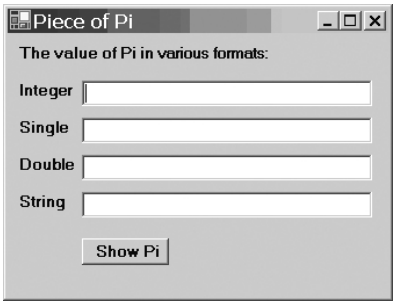
```
Windows Form Designer generated code
Dim TotalButtonClicks As Single
```

Переменная `TotalButtonClicks` будет глобальной. Ее значение можно будет считывать или записывать во всем коде формы, включая обработчики событий.

Microsoft-CD

Задания для самостоятельного выполнения

6.3. В системе программирования Visual Basic .NET создать проект «Число Pi», осуществляющий вывод числа π в текстовые поля в различных форматах. Создайте форму следующего вида:



В обработчике нажатия на кнопку объявите переменные следующих типов: `Integer`, `Single`, `Double`, `String`. Присвойте каждой переменной значение, равное числу π. Это значение в Visual Basic обозначается так: `Math.PI`. Выведите значение каждой переменной в соответствующем текстовом поле.

6.4. Переменные в программах

Объявив переменную, вы можете использовать ее в своем коде. Первое, что нужно сделать в коде, — задать начальное значение переменной. Можно задать любое значение переменной, если оно правильного типа. Для задания значения переменной используется оператор присваивания. Вот несколько примеров:

```
Dim MyName As String
MyName = "Мое имя"

Dim MyWeight As Integer
MyWeight = 128

Dim DrivingDistance As Single
DrivingDistance = 12.8
```

В C# и J# инициализация переменной нужным значением делается так:

```
int myOrbit;
myOrbit=123000;
```

Переменная MyName объявлена как принадлежащая к типу String, и ей присвоена строка "Мое имя". Переменная MyWeight объявлена как Integer и ей присвоено значение 128, целое число. Переменная DrivingDistance объявлена как Single, и ей присвоено значение 12.8, десятичная дробь.

Есть очень удобный способ объединять в одной строке кода объявление переменной и присваивание ей начального значения. Этот способ гарантирует, что у переменной с самого начала будет правильное значение. Использование переменной, которой не присвоено начальное значение, может привести к неправильной работе программы.

```
Dim MyAge As Integer = 100
```

Теперь, объявив переменную и присвоив ей начальное значение, можно использовать ее в коде каждый раз, когда вам нужно это значение. Как вы думаете, что будет выведено в окне сообщения?

```
Dim MyName As String
MyName = "Joe Cooker"
MessageBox.Show(MyName)
```

Что будет выведено в TextBox1?

```
Dim MyName As String
MyName = "Peter, Paul and Mary"
TextBox1.Text = MyName
```

Значение переменной можно изменять с помощью операторов присваивания. Какое имя будет выведено в TextBox1 в следующем примере? Переменной MyName присвоено значение "Bill", затем — "Bob", а затем — "Ben". Поскольку последнее значение MyName — "Ben", именно оно будет выведено в TextBox1.

```
Dim MyName As String
MyName = "Bill"
MyName = "Bob"
MyName = "Ben"
TextBox1.Text = MyName
```

Вот небольшой пример кода на J# или C#. Как видите, есть и общие черты, и отличия этого кода от кода на Visual Basic. NET. Обратите внимание на объявление переменных.

```
string MyName;
string YourName;
MyName =
"Allen Ginsberg";
YourName = myName;
MessageBox.Show
(YourName);
```

Можно присвоить переменной значение другой переменной. Что будет выведено в окне сообщения?

```
Dim MyName As String
Dim YourName As String
MyName = "Allen Ginsberg"
YourName = MyName
MessageBox.Show(YourName)
```

Вот пример, содержащий ошибку: мы пытаемся присвоить переменной значение переменной другого типа. Помните, переменные объявляются как принадлежащие к определенному типу. Им можно присвоить значения только этого типа. Этот код нельзя скомпилировать:

```
'Ошибка
Dim MyName As String
Dim DVDPrice As Single
MyName = "Paul Bunion"
DVDPrice = MyName
```

Давайте напишем программу, в которой переменные используются! Создайте новое приложение Windows в Visual Studio .NET. Назовите его «Имена». Откройте окно *Область элементов* и добавьте на форму `Form1` кнопку и два текстовых поля. Очистите свойство `Text` обоих текстовых полей. Установите значение `Показать имена` свойства `Text` кнопки. Установите значение `True` свойства `ReadOnly` обоих текстовых полей. Двойным щелчком по кнопке `Button1` откройте в редакторе кода обработчик ее нажатия. Введите в обработчик следующие строки кода:

```
Dim XName As String
Dim YName As String
XName = "X is my name"
YName = "Y is my name"
TextBox1.Text = XName
TextBox2.Text = YName
```

Постройте и запустите приложение. Нажмите кнопку с надписью *Показать имена*. В вашем коде объявлены две строковые переменные `XName` и `YName`.

Переменные хранят две строки текста (заключенные в кавычки). Свойству Text поля TextBox1 будет присвоено значение переменной XName. Свойству Text поля TextBox2 будет присвоено значение переменной YName. Попробуйте изменять значения XName и YName в коде и снова запускать программу.

А теперь изменим программу. Добавьте на форму еще одно текстовое поле. Очистите его свойство Text. Убедитесь, что для свойства ReadOnly этого текстового поля установлено значение False. Выполните двойной щелчок по кнопке Button1, чтобы отредактировать обработчик ее нажатия. Измените код, чтобы он выглядел так:

```
Dim XName As String
Dim YName As String
Dim ZName As String
ZName = TextBox3.Text
XName = ZName
YName = XName
TextBox1.Text = XName
TextBox2.Text = YName
```

Постройте и запустите приложение. Введите в третье текстовое поле какой-нибудь текст. Нажмите кнопку с надписью *Показать имена*. Что произойдет? Ваш код объявляет три строковые переменные. Переменной ZName присваивается значение свойства Text текстового поля TextBox3. Переменной XName присваивается значение переменной ZName (которой ранее было присвоено значение свойства Text текстового поля TextBox3). Переменной YName присваивается значение переменной XName (которой было только что присвоено значение переменной ZName). Затем код помещает в свойство Text поля TextBox1 значение переменной XName, а в свойство Text поля TextBox2 — значение YName. Этот код показывает, как поместить в переменную значение другой переменной.

Вот еще один пример — он покажет вам, как использовать глобальные переменные. Создайте новое приложение Windows и назовите его «Количество

щелчков». Добавьте на форму Form1 три кнопки. В редакторе кода найдите следующую строку:

```
Windows Form Designer generated code
```

Сразу после нее введите следующее:

```
Dim Number As Integer
```

Замечание. Переменной Number автоматически присваивается значение 0.

Выполните двойной щелчок по кнопке Button1 в редакторе форм и добавьте в обработчик следующий код:

```
Number = Number + 1
MessageBox.Show(Number)
```

Добавьте такой же код в обработчики нажатий кнопок Button2 и Button3.

Постройте и запустите приложение. Нажимайте кнопки в произвольном порядке. При каждом нажатии будет выводиться окно сообщения, в котором будет указано общее количество нажатий. Этот код работает, потому что мы объявили переменную Number как глобальную в коде формы. Эта переменная используется для хранения общего количества нажатий. При каждом нажатии кнопки значение этой переменной увеличивается на 1. Значение в глобальной переменной формы сохраняется, пока форма не будет закрыта.

Задания для самостоятельного выполнения

Microsoft-CD

6.4. В системе программирования Visual Basic .NET создать проект «Имена», описанный в параграфе. Готовый проект хранится в архиве Имена.exe.

6.5. В системе программирования Visual Basic .NET создать проект «Количество щелчков», описанный в параграфе. Готовый проект хранится в архиве Количество_щелчков.exe.

6.6. В системе программирования Visual Basic .NET открыть проект «Найди ошибки», найти и исправить ошибки в программном коде. Готовый проект хранится в архиве Найди_ошибки.exe.

Тест по теме «Присваивание и переменные»

1.  Типом переменной не является

   - ❍ `Boolean`
   - ❍ `Form1.Button1`
   - ❍ `Single`
   - ❍ `System.Drawing.Color`

2.  Данные какого типа всегда хранятся в свойстве `Text` текстового поля?

   - ❍ `Single`
   - ❍ `Integer`
   - ❍ `Boolean`
   - ❍ `String`

3.  Переменные какого типа могут принимать одно значение из двух возможных?

   - ❍ `Single`
   - ❍ `Integer`
   - ❍ `Boolean`
   - ❍ `String`

4.  Прежде чем использовать переменную в программе, обязательно нужно

   - ❍ *Присвоить переменной значение*
   - ❍ *Объявить переменную*
   - ❍ *Обнулить переменную*
   - ❍ *Вывести значение переменной в текстовое поле*

Глава 7

Операции

Microsoft В 1993 году была создана операционная система с графическим интерфейсом Windows NT для серверов и рабочих станций, которая содержала утилиты для работы в локальной сети и в Интернете. Операционная система обеспечивала большую информационную безопасность (использовала файловую систему NTFS, позволяла устанавливать политики безопасности для рабочих групп и др.). На основе Windows NT в последующие годы были разработаны операционные системы Windows 2000 и Windows XP.

7.1. Арифметические операции

Примеры арифметических операций, которые вы использовали с первого класса на уроках математики, — сложение, вычитание, умножение и деление. В программировании тоже есть такие операции. Кроме того, имеются и другие операции, позволяющие, например, выполнять сложные расчеты и объединять текстовые строки.

Операции обычно обрабатывают два значения, хотя для некоторых нужно только одно. Эти значения называются операндами. Вот общий синтаксис операции:

```
Операнд1 Операция Операнд2
```

Пример:

```
3 + 4
```

В выражении `3 + 4` есть два операнда (`3` и `4`) и операция (+). Операция выполняет действие (в данном случае сложение) над двумя операндами (`3` и `4`).

Компьютерные программы отлично подходят для выполнения вычислений. Они никогда не ошибаются и не устают делать одно и то же раз за разом. Звучит слишком хорошо, чтобы быть правдой? Вы правы, тут есть проблема. Программы могут выполнять вычисления, но не могут подготавливать задачи для решения. Если вы программист, это должны делать вы.

Именно поэтому важно понимать, как используются в программах арифметические операции. Именно они сообщают программе, какие действия нужно выполнить, чтобы решить задачу. Поскольку компьютеры предназначены для вычислений, во всех языках программирования есть стандартный набор арифметических операций. Поскольку эти операции широко распространены, во всех языках их обозначают почти одинаково.

Вот список самых распространенных арифметических операций и их обозначений в математике и в

В языках C# и J# операции обозначаются так же, как и в Visual Basic .NET. Сложение обозначается +, вычитание –, умножение *, деление /.

коде программ. Обратите внимание на то, что в программах по-другому обозначается только умножение: * вместо точки (·).

Операция	Обычное обозначение	Символ в программах
Сложение	+	+
Вычитание	–	–
Умножение	·	*
Деление	/	/

Арифметические операции чаще всего используются в операторах присваивания. Оператор присваивания выглядит как равенство, в нем используется знак «равно». Операции и операнды размещаются справа от знака «равно». Выражение справа от знака «равно» вычисляется, и результат вычислений записывается в переменную слева от знака «равно». Выражение справа от знака «равно» должно иметь значение того же типа, что и переменная слева от знака равно, иначе код не скомпилируется. Вот несколько примеров использования арифметических операций в операторах присваивания:

```
Dim FormWidth as Integer
FormWidth = 200 + 300

Dim FormHeight as Integer
FormHeight = 1000 / 2
```

Данные примеры очень простые — в каждом из них по два операнда и одной операции. В настоящих программах чаще всего выполняются сложные расчеты, и нужно использовать несколько операндов и операций в каждом операторе. К счастью, для большинства операций в одном операторе можно использовать столько операций, сколько нужно. Кроме того, в качестве операндов можно использовать переменные. Операции можно выполнять и над пере-

Скобки можно использовать в операторах точно так же, как в обычных математических задачах. То, что находится в скобках, вычисляется первым. Пример:

```
(5+7)/(1+5)= 2
```

Здесь сначала вычисляется `(5+7)`, потом `(1+5)`, затем выполняется деление. Ответ равен `2`.

Без скобок мы бы получили другой ответ:

```
5+7/1+5 = 17
```

С помощью скобок вы можете указать программе, в каком порядке нужно выполнять вычисления.

менными — если эти переменные подходят по типу. А еще можно записать в переменную значение выражения, в котором используется сама эта переменная. Давайте разберем более сложные примеры. Первый пример вычисляет длину окружности, если известен диаметр.

```
Dim Circum As Single
Dim Diameter As Integer
Dim Pi As Single
Pi = 3.14159
Diameter = 18
Circum = Diameter * Pi
MessageBox.Show(Circum)
```

В следующем примере вычисляется средняя цена бензина за два месяца — март и апрель. Обратите внимание на скобки — они используются, чтобы сложить цены и объемы, прежде чем делить.

```
Dim MarchCost As Single = 809.6
Dim AprilCost As Single = 1661.55
Dim MarchLitres As Single = 50.6
Dim AprilLitres As Single = 100.7
Dim CostPerLitres As Single
CostPerLitres = (MarchCost + AprilCost) /_
(MarchLitres + AprilLitres)
MessageBox.Show(CostPerLitres)
```

В переменную можно записать значение выражения, в котором используется эта же переменная. Посмотрите на следующий пример и попытайтесь определить, что будет выведено в окне сообщения.

```
Dim MileCounter as Integer
MileCounter = 100
MileCounter = MileCounter + 200
MileCounter = MileCounter + 400
MessageBox.Show(MileCounter)
```

В окне сообщения будет выведено 700, значение переменной `MileCounter` при вызове этого окна. Вот почему это получается. Первые две строки кода

объявляют переменную `MileCounter` и задают ее начальное значение 100. Третья строка кода берет текущее значение переменной `MileCounter` (100) и прибавляет к нему 200 — получается 300. Не забывайте, первой всегда вычисляется правая часть оператора присваивания. Затем левой части оператора присваивания (`MileCounter`) присваивается значение правой части (300). Четвертая строка кода берет текущее значение переменной `MileCounter` (теперь 300) и добавляет к нему 400, получается 700. Это значение присваивается `MileCounter`, оно же отображается в окне сообщения. Для левой части оператора присваивания не важно, как было вычислено значение правой части, если оно подходит по типу. Поэтому `MileCounter` может находиться в правой части оператора присваивания точно так же, как и любая другая переменная.

Присваивание переменной значения выражения, содержащего саму эту переменную, — это обычный прием программирования, поддерживаемый всеми языками .NET. Он избавляет нас от необходимости использовать дополнительную переменную для хранения промежуточного значения. Как видно из примера, который мы только что рассмотрели, этот прием очень удобен для подсчетов суммы набора элементов. Очень часто встречается строка кода вида `variable = variable + 1`, используемая для увеличения на 1 значения переменной. Она используется, например, при подсчете количества повторений в циклах.

Задания для самостоятельного выполнения

Microsoft-CD

7.1. В системе программирования Visual Basic .NET создать проект «Цена бензина-1», описанный в параграфе. Готовый проект содержится в самораспаковывающемся архиве Цена_бензина-1.exe.

7.2. В системе программирования Visual Basic .NET создать проект «Диаметр окружности», опи-

санный в параграфе. Готовый проект содержится в самораспаковывающемся архиве Диаметр_окружности.exe.

7.3. В системе программирования Visual Basic .NET создать проект «Путь», описанный в параграфе. Готовый проект содержится в самораспаковывающемся архиве Путь.exe.

7.4. В системе программирования Visual Basic .NET создать проект «Цена бензина-2», в котором вычисляется цена бензина, необходимого для преодоления определенного расстояния, если заданы длина пробега в километрах, для которой требуется литр горючего, и цена литра горючего. Создайте форму, разместив на ней следующие элементы управления:

1. Четыре текстовых поля, чтобы можно было отображать:
   - расстояние;
   - длину пробега в километрах, для которой требуется литр горючего;
   - цену литра горючего;
   - стоимость бензина.
2. Четыре надписи для описания текстовых полей.
3. Кнопку, при нажатии которой будет начинаться вычисление.

Для преобразования значений строкового типа, вводимых в текстовые поля, в числа использовать функцию `Val()`. Готовый проект содержится в самораспаковывающемся архиве Цена_бензина_2.exe.

7.2. Строковые операции

Не во всех программах задача сводится только к вычислениям, иногда программе нужно работать с текстовыми строками. Вы знаете, что в Visual Basic .NET и других языках программирования можно

В языках C# и J# амперсенд (&) не используется для соединения текстовых строк. Используется плюс (+). Посмотрите на следующую строку кода:

```
fullName = "Bill " +
"Bob";
```

«складывать» текст? На самом деле это не то же самое сложение, что и для чисел. «Сложение» текстовых строк называется конкатенацией — это технический термин, обозначающий соединение двух строк. В Visual Basic .NET операция конкатенации задается с помощью знаков «амперсенд» (&) или «плюс» (+). Они используются точно так же, как операция сложения, но со строками, а не с числами.

Вот примеры использования операции конкатенации:

```
"Bill" & "Mike
"1" & "2"

"Bill" + "Mike
"1" + "2"
```

Соединяя текстовые строки с помощью операции конкатенации, Visual Basic. NET не вставляет между строками пробелы. Вы должны указать в коде, нужно ли добавлять пробелы. Это можно сделать двумя способами:

```
"Bill" & " " & "Mike"
```

или

```
"Bill " & "Mike"
```

Вот пример использования операции конкатенации в операторе присваивания:

```
Dim MyName as String
MyName = "Имя" & "Фамилия"
```

Вот пример, выполняющий конкатенацию строковых переменных и выводящий результаты в текстовое поле:

```
Dim FirstName As String
Dim LastName As String
FirstName = "Bob"
LastName = "Marley"
TextBox1.Text = FirstName & " " & LastName
```

Вы заметили, как программе на Visual Basic .NET было приказано вставить пробел между значениями `FirstName` и `LastName`?

Задания для самостоятельного выполнения

Microsoft-CD

7.5. В системе программирования Visual Basic .NET создать проект «Имя, фамилия», описанный в параграфе. Готовый проект содержится в самораспаковывающемся архиве Имя_фамилия.exe.

7.3. Логические операции

В состав логических выражений могут входить логические переменные, логические значения, операторы сравнения, а также логические операции. Логические выражения могут принимать лишь два значения `True` (Истина) или `False` (Ложь).

Операторы сравнения =, <, >, , <= и >= сравнивают выражение в левой части оператора с выражением в правой части оператора и представляют результат в виде логического значения `True` или `False`. Примеры:

```
5 > 3 = True
"A" = "B" = False
```

Над элементами логических выражений могут производиться логические операции, которые на языке Visual Basic обозначаются следующим образом: логическое умножение — `And`, логическое сложение — `Or` и логическое отрицание — `Not`. При записи сложных логических выражений используются скобки. Примеры:

```
(5 > 3) And ("A" = "B") = False
(5 > 3) Or ("A" = "B") = True
Not (5 > 3) = False
```

В языке C# вместо оператора Not используется восклицательный знак (!). Вот пример его использования в коде на C#:

```
textBox3.Visible =
!(textBox1.Visible);
```

Операция `Not` (Не) — это специальная операция, обрабатывающая только один операнд. Кроме того, операнд должен быть булевым значением (`True` (Истина) или `False` (Ложь)). Оператор `Not` изменяет значение `False` на `True` или значение `True` на `False`. Посмотрите на следующие примеры.

```
Not (True)
Not (Not (True))
```

Вряд ли вам может понадобиться последнее выражение, но оно тоже правильное!

Следующий пример демонстрирует, как использовать операцию `Not`, чтобы сделать первоначально видимыми, а затем — невидимыми текстовые поля. Не забывайте, операция `Not` работает с булевыми переменными (переменными типа `Boolean`).

```
TextBox1.Visible = True
TextBox2.Visible = True
```

и

```
TextBox1.Visible = Not True
TextBox2.Visible = Not True
```

Microsoft-CD

Задания для самостоятельного выполнения

7.6. В системе программирования Visual Basic .NET создать проект «Логика», описанный в параграфе. Готовый проект содержится в самораспаковывающемся архиве Логика.exe.

7.4. Отладка кода

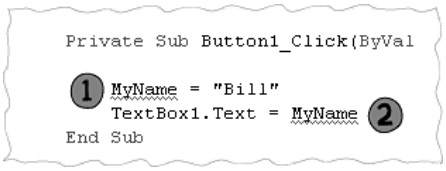
Вы многого достигли! Вы создали, построили и запустили множество проектов. Возможно, не все ваши проекты скомпилировались с первой попытки, но вы, вероятно, уже научились понимать, в чем ошибки в вашем коде, и как эти ошибки исправлять. Рассмотрим, как использовать специальные инструменты для поиска ошибок. Исправление ошибок в коде называется отладкой. Отладка — необходимый шаг для получения программ, работающих без ошибок и

выдающих правильные результаты. Сначала вы увидите, как Visual Studio помогает отлаживать код еще при его написании.

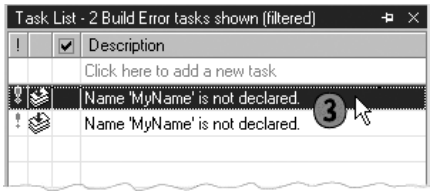
1. Создайте новое приложение Windows, назовите его «Отладка». Добавьте на форму `Form1` одну кнопку и одно текстовое поле. Выполните двойной щелчок по кнопке `Button1` и введите следующий код:

```
MyName = "Bill"
TextBox1.Text = MyName
```

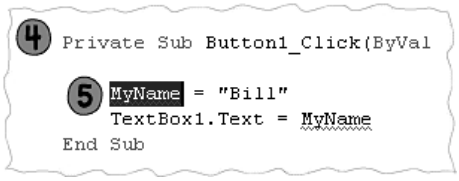
2. Обратите внимание, что еще до построения проекта Visual Studio .NET подчеркнет `MyName` в первой и второй строках волнистой синей линией. Это означает, что переменная `MyName` не объявлена. Синяя линия предупреждает вас, что в коде что-то неправильно, еще до того, как вы попытаетесь его скомпилировать.



3. Попытайтесь построить проект. В окне *Вывод* (*Output*) появится сообщение, что построение выполнить не удалось. В списке заданий *Список задач* (*Task List*) будет сообщено, почему построение не удалось. Вы увидите в этом списке две строки, сообщающие, что имя `MyName` не объявлено.



4. Выполните двойной щелчок по первому сообщению в списке *Список задач* (*Task List*). В коде курсор переместится в то место, в котором обнаружена ошибка.



5. Заметьте, что имя MyName подсвечено — именно в нем и заключается проблема, которую нужно решить. Добавьте в обработчик следующую строку кода (перед написанными ранее):

```
Dim MyName As String
```

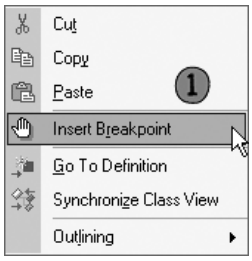
Волнистые синие линии исчезнут.

6. Постройте и запустите проект. Он скомпилируется без ошибок. Вы успешно отладили программу.

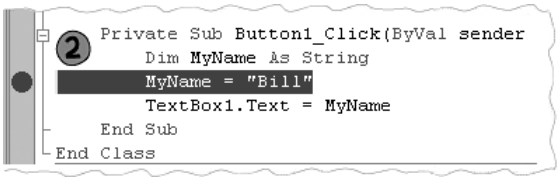
Пошаговое выполнение кода. Visual Studio .NET не может самостоятельно обнаружить некоторые ошибки в коде. Программа может скомпилироваться, но результаты ее работы будут неправильными. Это происходит, если ошибки имеются в логике работы программы, а не в синтаксисе.

В таких случаях бывает полезно пошагово выполнить код, строку за строкой, чтобы увидеть, как он выполняется и каковы результаты его выполнения. Обычно в коде задается точка останова, а затем, начиная с этой точки, код выполняется пошагово. Точка останова служит знаком остановки — выполнение прерывается в строке кода, которую вы пометили точкой останова. Нажимайте кнопку {F11} для выполнения по очереди каждой строки кода.

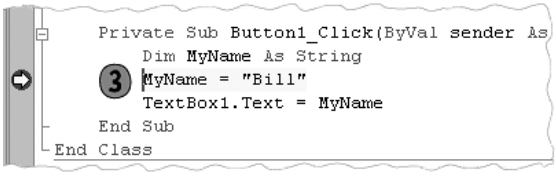
1. Найдите в обработчике нажатия кнопки `Button1` следующую строку:
   `MyName = "Bill"`
   Поместите курсор в эту строку. Щелкните по ней правой кнопкой мыши и выберите в появившемся меню пункт *Вставить точку останова* (*Insert Breakpoint*).



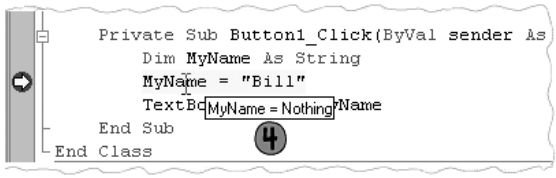
2. Строка будет выделена коричневым цветом, и рядом с ней на поле появится коричневый кружок. Постройте и запустите программу. Когда форма загрузится, нажмите кнопку на ней.



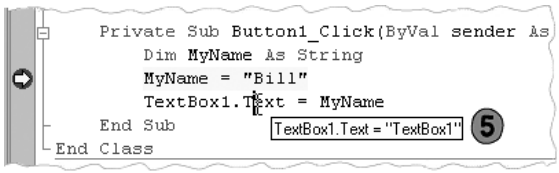
3. Начнет выполняться обработчик. Когда выполнение дойдет до строки, в которую вы поместили точку останова (`MyName = "Bill"`), выполнение прервется. Эта строка кода еще не выполнилась. Она выделена желтым цветом, и в коричневом кружке рядом с ней теперь показана желтая стрелка.



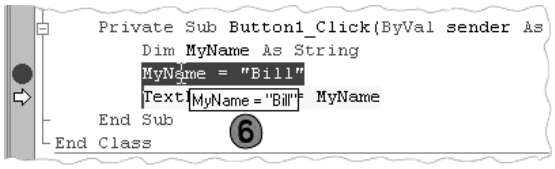
4. В редакторе кода подведите курсор к переменной `MyName`. Появится подсказка, показывающая значение этой переменной. Пока что в ней нет ничего (`MyName = Nothing`), потому что переменная была объявлена, но ей не присваивалось значение.



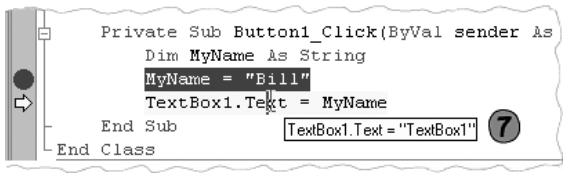
5. В редакторе кода подведите курсор к свойству `Text` элемента `TextBox1`. Появится подсказка, сообщающая вам значение свойства `Text`. В данный момент `Text = TextBox1`.



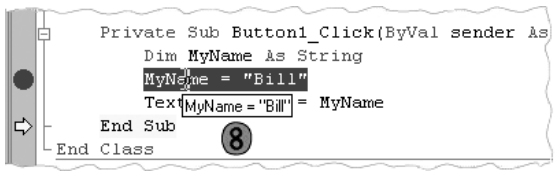
6. Нажмите клавишу {F11}. При этом выполнится текущая строка кода (`MyName = "Bill"`), и будет выделена желтым следующая. В редакторе кода подведите курсор к имени переменной `MyName` в строке `MyName = "Bill"`. Подсказка сообщит, что `MyName = "Bill"`, поскольку этой переменной только что было присвоено значение.



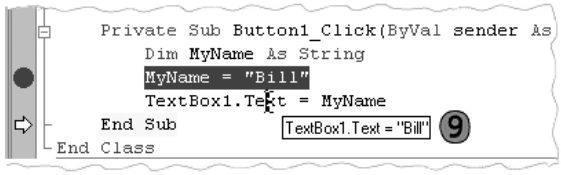
7. В редакторе кода подведите курсор к свойству `Text` элемента `TextBox1`. Подсказка сообщит, что `Text = TextBox1`. Это значение не изменилось.



8. Еще раз нажмите клавишу {F11}. Строка кода будет выполнена, и следующая строка выделится желтым цветом. Подведите курсор мыши к имени переменной `MyName` в строке `MyName = Bill`. Подсказка сообщит, что `MyName = "Bill"`, поскольку значение этой переменной не изменялось.

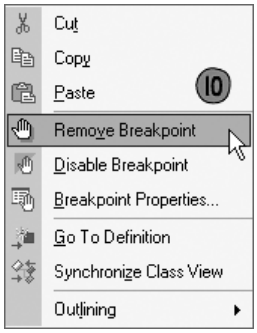


9. В редакторе кода подведите курсор к свойству `Text` элемента `TextBox1`. Подсказка сообщит, что `Text = "Bill"`, поскольку мы только что присвоили этому свойству значение переменной `MyName`.



Еще раз нажмите клавишу {F11}. Опять появится форма, поскольку обработчик закончил выполняться. Нажмите кнопку со значком ✕ в углу формы, чтобы закрыть ее.

10. В редакторе кода щелкните правой кнопкой в начале строки с точкой останова. Из появившегося контекстного меню выберите пункт *Удалить точку останова* (*Remove Breakpoint*), чтобы удалить точку останова.



Вы закончили сеанс отладки. Очень удобно иметь возможность выполнять код шаг за шагом. Можно отследить работу своей программы. Иногда можно таким образом обнаружить, что вы случайно указали программе выполнять код не в том порядке, в каком нужно, или использовали в операторе присваивания не ту переменную. Пошаговое выполнение программы позволяет определить, где именно программа начинает работать неправильно. Выполняя строки кода, можно отслеживать значения свойств и переменных, подводя к ним курсор мыши в окне редактора кода. Эти значения обновляются при выполнении строк кода.

Рассмотренные методики отладки становятся весьма полезными, когда код становится сложным. Сложный код сложнее отлаживать. Скоро вы узнаете, как управлять порядком выполнения кода с помощью операторов ветвления и циклов. Отладочные инструменты незаменимы для отслеживания выполнения этих операторов в программах.

Microsoft-CD

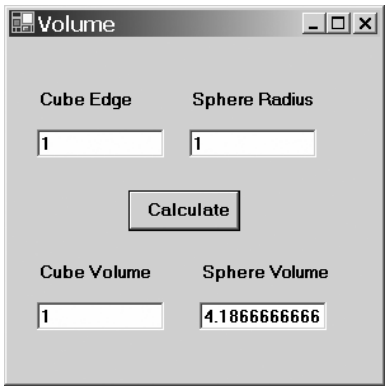
Задания для самостоятельного выполнения

7.7. В системе программирования Visual Basic .NET потренироваться в использовании навыков отладки на примере проекта «Отладка». Готовый проект содержится в самораспаковывающемся архиве Отладка.exe. Вы должны записать в текстовом файле, в документе Word или просто на бумаге, ответы на три вопроса:

1. Какое значение вы ввели в текстовое поле на форме?
2. Какое конечное значение у переменной `AnswerOne`?
3. Какое конечное значение у переменной `AnswerTwo`?

Код в форме намеренно сделан запутанным и сложным, чтобы вы не пытались угадать, как программа получает ответы. Поместите точку останова в коде и отслеживайте его выполнение шаг за шагом. Чтобы узнать значение переменной, подведите к ней курсор мыши после того, как строка кода выполнится.

7.8. В системе программирования Visual Basic .NET создать проект для вычисления объемов куба и сферы. Создайте форму вроде этой:



Длина ребра куба вводится в первом текстовом поле, а радиус сферы — во втором.

Из курса геометрии возьмем формулы для вычислений:

- Объем куба — это длина его ребра в третьей степени (умноженная сама на себя 3 раза).
- Объем сферы — это дробь 4/3, умноженная на число π и на радиус в третьей степени. В качестве значения π можно использовать 3,14 или `Math.PI`.

Дополнительное задание. Сможете ли вы подобрать такие значения длины ребра и радиуса, которые давали бы одинаковые объемы, хотя бы с точностью в 4 десятичных разряда?

Если вы хорошо разбираетесь в математике, можете попробовать добавить на форму еще одну часть, которая вычисляет длину ребра куба и радиус сферы по заданным объемам.

Тест по теме «Операции»

1. 

Что используется для изменения порядка выполнения операций?

- ❍ *Скобки*
- ❍ *Точки*
- ❍ *Амперсенды*
- ❍ *Плюсы*

2. 

Какая операция используется для конкатенации строк?

- ❍ `And`
- ❍ `Not`
- ❍ `&`
- ❍ `=`

3. 

Каким будет результат операции `Variable = "5" & "5"`?

- ❍ `55`
- ❍ `10`
- ❍ `25`
- ❍ `5`

4. 

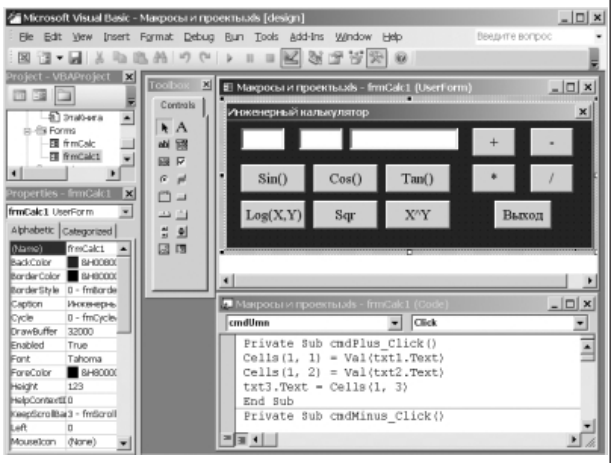
Результатом какой логической операции является `True`?

- ❍ `2 * 2 = 5`
- ❍ **`Not`** `(3 * 3 = 9)`
- ❍ `2 * 2 = 5` **`And`** `3 * 3 = 9`
- ❍ `2 * 2 = 5` **`Or`** `3 * 3 = 9`

Глава 8

Ветвление: неполная форма

Microsoft В 1994 году на основе системы программирования Visual Basic была создана система программирования для прикладных программ, которая получила название Visual Basic for Applications (VB для приложений). Первый вариант VBA 1.0 появился в составе MS Office 4.0, а в состав MS Office 2000 вошла система VBA 6.0, которая используется в шести приложениях — Word, Excel, PowerPoint, Access, Outlook и Frontpage.



8.1. Булева логика

Компьютерные программы хорошо умеют использовать булеву логику. В этой логике ответы на все вопросы всегда или «да» (`True`) или «нет» (`False`) — нет никаких «возможно», «похоже» или «почти». Вот несколько примеров ее использования.

```
4 = 4
```
4 действительно равно 4? Да, `True`.

```
3 = 1 + 3
```
3 действительно равно 1 + 3? Нет, `False`.

```
3 = (6 + 12)/(1 + 5)
```
3 действительно равно (6 + 12)/(1 + 5)? Да, `True`.

Как видите, в каждом примере ответ — или «да», или «нет». Именно так программа принимает решения. Если ответ «да» (`True`), программа делает одно. Если ответ «нет» (`False`) — другое.

Как и в математике, в булевой логике есть свои операции. Но эти операции работают не с числами, а только с булевыми операндами, т. е. со значениями `True` и `False`. Они позволяют обрабатывать значения `True` и `False` специальными способами. С их помощью можно объединять булевы выражения, составляя более сложные выражения.

Есть три булевых (логических) операции: `AND`, `OR` и `NOT`. Операции `AND` и `OR` обрабатывают два операнда. Операция `NOT` обрабатывает один операнд. Все эти операции возвращают результат булевого типа (`Boolean`). У каждой из этих операций есть правила, определяющие ее результат в зависимости от значений операндов. Давайте посмотрим на несколько булевых задач, использующих эти операции, чтобы понять, как работают операции.

Операция логического умножения AND

```
3 = 1 + 2 AND 2 = 1 + 1
```

Результат — `True`.

Здесь у нас есть два утверждения, объединенных операцией `AND`. Сначала мы вычисляем выражение 3=1+2. Оно правильно, т. е. `True`. Потом вычисляем выражение 2=1+1, которое тоже правильно — `True`. Затем мы проверяем выражение `True AND True`, которое по правилам выполнения операции `AND` тоже дает ответ `True`: если оба операнда операции `AND` равны `True`, то ответ этой операции — тоже `True`.

```
3 = 1 + 2 AND 5 = 2 + 2
```

Результат — `False`.

Опять у нас есть два выражения, объединенных операцией `AND`. Сначала мы вычисляем 3=1+2, получая результат `True`. Затем вычисляем 5=2+2 — равенство не выполняется, значит, результат — `False`. Затем вычисляем выражение `True AND False`, которое по правилам операции `AND` даст результат `False`: если хотя бы один операнд оператора `AND` равен `False`, результат выполнения операции будет тоже `False`.

Вам понадобится некоторое время, чтобы запомнить правила выполнения логических операций, но постепенно вы к ним привыкнете. К счастью, можно перечислить все возможные сочетания операндов операции `AND` в простой таблице. Их не обязательно запоминать; можете просто обращаться к этой таблице, когда вам будет нужно.

Левый операнд	Правый операнд	Левый операнд AND правый операнд (результат операции)
True	True	True
True	False	False
False	True	False
False	False	False

Операция логического сложения OR

```
3 = 1 + 2 OR 5 = 2 + 2
```
Результат — True.

Сначала мы вычисляем 3=1+2, результат — True. Затем вычисляем 5=2+2, результат — False. Затем вычисляем True OR False. Согласно правилам операции OR мы получим результат True: если хотя бы один операнд операции OR равен True, то результат операции — тоже True. Все возможные сочетания операндов для операции OR перечислены в таблице:

Левый операнд	Правый операнд	Левый операнд OR правый операнд (результат операции)
True	True	True
True	False	True
False	True	True
False	False	False

Операция логического отрицания NOT

Операция NOT обрабатывает только один операнд. Она просто преобразует значение False в True, а значение True — в False. Вот примеры использования операции NOT.

NOT(3 = 1 + 2) Результат — False.
3 = 1 + 2 — результат True. NOT True — результат False.

NOT(5 = 2 + 2) Результат — True.
5 = 2 + 2 — результат False. NOT False — результат True.

Операнд	NOT операнд (результат операции)
True	False
False	True

Microsoft-CD

Задания для самостоятельного выполнения

8.1. В системе программирования Visual Basic .NET создать проект «Логические операции», который строит таблицы истинности логических операций. Готовый проект содержится в самораспаковывающемся архиве Логические_операции.exe.

8.2. Операции сравнения

Компьютеры могут решать очень сложные логические (булевы) задачи, но они не умеют их составлять. Это должны делать программисты, которые создают программы, решающие логические задачи. Когда код выполняется, программа решает задачу. Ответом будет `True` или `False`. Программа «принимает решения», в зависимости от того, выполняются какие-то условия (`True`) или не выполняются (`False`).

Вы уже знаете, что знак «равно» обозначает операцию присваивания, но он может обозначать и операцию сравнения. В примерах, которые мы рассмотрели раньше, знак «равно» задает вопрос «Равна ли левая сторона правой? Да или нет, `True` или `False`?» Есть и другие операции сравнения, позволяющие выяснить, больше левая сторона правой (>) или меньше (<). Все эти операции задают вопрос, ответ на который — `True` или `False`. Примеры:

```
3 < 4
```
Задается вопрос: 3 меньше, чем 4?
Да, `True`.

```
2 > 5
```
Задается вопрос: 2 больше, чем 5?
Нет, `False`.

Вот список часто используемых операций сравнения:

Операция сравнения	Значение	Вопрос
=	Равно	Левая сторона равна правой?
>	Больше	Левая сторона больше правой?
<	Меньше	Левая сторона меньше правой?
>=	Больше или равно	Левая сторона больше или равна правой?
<=	Меньше или равно	Левая сторона меньше или равна правой?
<>	Не равно	Левая сторона не равна правой?

Возможно, вы раньше не встречались с операциями >=, <= и <>. В каждой такой операции объединены две операции сравнения, объединенные логической операцией OR. Например, >= значит: левая сторона больше правой или (OR) левая сторона равна правой? Операция <> значит: левая сторона меньше правой или левая сторона больше правой? Иными словами, операция <> спрашивает, не равны ли друг другу левая и правая стороны?

Вот несколько примеров использования операций сравнения в булевых выражениях. Возможно, вам понадобится возвращаться назад и смотреть в таблицу операции OR, чтобы понять, как работает каждое выражение.

```
3 <= 3
```
3 < 3 есть False. 3 = 3 есть True.
```
False OR True = True.
```
Значит, результат операции «3 меньше или равно 3» — True.

```
2 >= 3
```
2 > 3 есть False. 2 = 3 есть False.
```
False OR False = False
```
Значит, результат операции «2 больше или равно 3» — False.

С помощью операций сравнения строки можно сравнивать так же, как и числа.

```
"ABC" < "DEF"
```
"ABC" меньше "DEF"? Да, `True`.

```
"DEF" >= "ABC"
```
"DEF" больше или равно "ABC"? Да, `True`.

```
3 <> 4
```
3 < 4 есть `True`. 3 > 4 есть `False`.
`True OR False = True`.
Значит, результат операции «3 не равно 4» — `True`.

```
3 <> 3
```
3 < 3 есть `False`. 3 > 3 есть `False`.
`False OR False = False`.
Значит, результат операции «3 не не равно 3» — `False`, т. е. 3 равно 3.

Во всех этих примерах мы сравнивали числа. Но с помощью операций сравнения можно сравнивать и текстовые строки, и просто булевы значения, например, так:

```
"ABC" = "DEF"
```
"ABC" равно "DEF"? Нет, `False`.

`True` = `True`
`True` равно `True`? Да, `True`.

`True` = `False`
`True` равно `False`? Нет, `False`.

8.3. Оператор If...Then

Задать в программе вопрос, ответ на который — да, `True`, или нет, `False`, можно с помощью булевой логики: операций сравнения и логических (булевых) операций. А как реагировать на полученный ответ? В зависимости от ответа можно выполнять какие-то действия или нет. Если ответ `True`, программа что-то делает. Если ответ `False`, она ничего не делает. Вот примеры принятия решения в программе.

Если установлен флажок `Super Size`, показать большую картинку.

Расшифруем подробнее:

Флажок `Super Size` установлен? Ответ `True`. Показать большую картинку.

Флажок Super Size установлен? Ответ False. Не показывать большую картинку.

Если значение переменной TotalLitres больше 0, то вычислить цену бензина.

Подробнее:

Значение переменной TotalLitres > 0? Ответ True. Вычислить цену бензина.

Значение переменной TotalLitres > 0? Ответ False. Не вычислять цену бензина.

В примерах задаются вопросы, на которые можно ответить True или False. В этих вопросах используются операции сравнения. Ответы на вопросы — всегда True или False. Если ответ — True, то выполняются какие-то действия. Если ответ False, то никакие действия не выполняются.

Вы заметили, что в примерах используются слова «если» и «то»? Если условие выполняется (True), то что-то происходит. Если условие не выполняется (False), ничего не происходит. Если вы хотите, чтобы программа принимала решения, используйте оператор «если...то» (If...Then). В Visual Basic .NET синтаксис этого оператора такой:

```
If Условие Then
    Последовательность операторов
End If
```

Вот пример кода на Visual Basic .NET:

```
If myAge = 15 Then
    MessageBox.Show("I am 15.")
End If
```

Слово If — ключевое слово Visual Basic .NET, поэтому оно выделено полужирным шрифтом. Условие идет сразу после слова If. В условии с помощью операции сравнения задается вопрос, на который дается ответ True или False. Значение переменной myAge равно 15?

Слово Then идет сразу после условия. Это тоже ключевое слово Visual Basic.NET, поэтому оно выделено полужирным шрифтом. После слова Then идет

код, который должен выполняться, если выполняется условие (ответ на вопрос — `True`). Этот код записан с отступом для удобства восприятия. Если условие выполняется (`True`), программа выводит окно сообщения с текстом «I am 15.» Слова `End If` после условного кода — это тоже ключевые слова. Слова `End If` завершают оператор `If...Then`.

Попробуем применить эти операторы! Создайте новое приложение Windows и назовите его «If_Then». Поместите на форму текстовое поле и кнопку. Выполните по кнопке двойной щелчок, чтобы отредактировать обработчик ее нажатия. Добавьте в обработчик следующий код:

```
Dim myAge As Integer
myAge = 15
If myAge = 15 Then
    TextBox1.Text = "I am 15."
End If
```

Постройте и запустите проект. Нажмите кнопку. Код объявляет переменную `myAge` и присваивает ей значение `15`. После ключевого слова `If` задается вопрос: `myAge` равно `15`? Ответ — `True`, поскольку мы только что присвоили переменной `myAge` значение `15`. Поскольку ответ `True`, выполняется код после ключевого слова `Then`. В текстовом поле появляется текст «I am 15.»

А теперь изменим код с `myAge = 15` на:

```
myAge = 14
```

Еще раз постройте и запустите проект. Нажмите кнопку. Что произойдет? Ничего? На этот раз `myAge` присвоено значение `14`. После ключевого слова `If` задается вопрос: `myAge` равно `15`? Теперь ответ — `False`, и код после ключевого слова `Then` не выполняется. Ничего не происходит. Текст в текстовом поле не изменяется.

Давайте разберем еще один пример. Добавьте на форму флажок и кнопку. Создайте обработчик нажатия второй кнопки:

Все операторы условного кода между `If...Then... End If` лучше выделять отступом. Это сделает код более понятным. Даже если весь код, выполняющийся по условию, — это только один оператор, его стоит выделить отступом. По умолчанию Visual Studio .NET автоматически выделяет такой код отступами.

```
If CheckBox1.Checked =
True Then
  MessageBox.Show
  ("Danger!")
End If
```

Microsoft-CD

```
If CheckBox1.Checked = True Then
    TextBox1.Text="I am checked."
End If
```

Постройте и запустите проект. Установите флажок и нажмите вторую кнопку. Условие выполняется — `True` (вы установили флажок), и текст в текстовом поле изменится на «I am checked.»

Еще раз запустите проект. На этот раз нажмите вторую кнопку, не устанавливая флажок. Что произойдет? Ничего. На этот раз условие не выполняется — `False` (флажок не установлен), и текст не изменится. Ваша программа приняла решение (изменять ли текст) исходя из условия (установлен ли флажок).

Задания для самостоятельного выполнения

8.2. В системе программирования Visual Basic .NET создать проект «If_Then», который описан в параграфе. Готовый проект содержится в самораспаковывающемся архиве If_Then.exe.

8.4. Множественные условия

Иногда программам необходимо принимать множество решений одно за другим. Например, нужно проверить, установлен ли флажок, выбрана ли позиция переключателя, затем сравнить два числа, и проверить, оставил ли пользователь пустым текстовое поле. Все это нужно сделать в одной программе. Как? К счастью, не слишком сложно. В коде можно использовать столько операторов `If...Then`, сколько нужно. Как и другие операторы, они выполняются в том порядке, в каком встречаются в коде. Давайте создадим приложение, демонстрирующее использование нескольких операторов `If...Then`.

Предположим, что в программе есть флажок CheckBox1 с надписью *Тревога*. Если флажок установлен, программа изменяет цвет фона (BackColor) формы на красный, текст в текстовом поле — на «Пожар» и выводит окно сообщения с текстом «Опасность! Опасность!» Все это можно сделать с помощью трех отдельных операторов If...Then. Каждый из этих операторов будет проверять, установлен ли флажок Alarms. Если да (True), то будет выполняться соответствующий код.

Создайте новое приложение Windows и назовите его «Пожарная тревога». Добавьте на форму флажок, текстовое поле и кнопку. Измените значение свойства Text кнопки Button1 на Три условия. Выполните двойной щелчок по кнопке, чтобы отредактировать обработчик ее нажатия. Добавьте в обработчик следующий код:

```
If CheckBox1.Checked = True Then
    Form.ActiveForm.BackColor = System.Drawing._
    Color.Red
End If
If CheckBox1.Checked = True Then
    TextBox1.Text = "Пожар"
End If
If CheckBox1.Checked = True Then
    MessageBox.Show("Опасность! Опасность!")
End If
```

Постройте и запустите проект. Установите флажок и нажмите кнопку с надписью *Три условия*. Операторы If...Then выполнятся один за другим. Первый оператор If...Then изменит цвет фона формы на красный. Второй оператор изменит текст в текстовом поле на «Пожар». Третий оператор выведет окно сообщения с текстом «Опасность! Опасность!» Во всех операторах If...Then в примере «Пожарная тревога» проверялось одно и то же условие. Это было условие

```
If CheckBox1.Checked = True Then
```

Но в блоках If...Then выполнялись разные операторы. Вы тоже считаете, что один и тот же код повторяется слишком много раз? Есть ли лучшее решение? В Visual Basic .NET и большинстве современных языков программирования в одном блоке If...Then можно использовать множество операторов.

Так что код в приложении «Пожарная тревога» можно переписать, упростив его за счет удаления лишних блоков. Откройте приложение «Пожарная тревога». Добавьте на форму еще одну кнопку. Измените ее текст на «Одно условие». Выполните по ней двойной щелчок, чтобы отредактировать обработчик ее нажатия. Добавьте в обработчик следующий код:

```
If CheckBox1.Checked = True Then
    Form.ActiveForm.BackColor = System.Drawing._
    Color.Red
    TextBox1.Text = "Пожар"
    MessageBox.Show("Опасность! Опасность!")
End If
```

Постройте и запустите проект. Установите флажок и нажмите кнопку с надписью *Одно условие*. Выполнится один оператор If...Then. Мы получим ответ True, и все операторы в блоке If...Then будут выполнены один за другим. В программе выполняются те же действия, что и раньше. Однако на этот раз условие проверяется только один раз. Такой код легче понимать и писать.

Используя последовательности операторов If...Then, можно выбирать разные варианты в коде (устанавливать переключатели).

Рассмотрим пример.

Создайте новое приложение Windows и назовите его «Выбор цвета». Поместите на форму три переключателя и кнопку. С помощью окна *Свойства*

(*Properties*) задайте следующие значения свойств. Измените значение свойства Text у RadioButton1 на Red, у RadioButton2 — на Green, а у RadioButton3 — на Blue. У кнопки Button1 измените значение свойства Text на Установить цвет. Установите свойство Checked у RadioButton1 в значение True. Выполните двойной щелчок по кнопке Button1, чтобы отредактировать обработчик ее нажатия. Добавьте в него такой код:

```
Dim myColor As System.Drawing.Color
If RadioButton1.Checked = True Then
    myColor = System.Drawing.Color.Red
End If
If RadioButton2.Checked = True Then
    myColor = System.Drawing.Color.Green
End If
If RadioButton3.Checked = True Then
    myColor = System.Drawing.Color.Blue
End If

Form.ActiveForm.BackColor = myColor
```

Постройте и запустите проект. Установите один из переключателей и нажмите кнопку с надписью *Установить цвет*. Форма и кнопки приобретут выбранный цвет. Выберите другой переключатель и нажмите кнопку с надписью *Установить цвет*. Цвет формы и кнопок станет другим!

Посмотрите на код. Последовательность операторов If...Then позволяет выбирать разные варианты ответа на вопрос: «Какой переключатель выбран?» Когда код выполняется, программа постепенно определяет, какая позиция выбрана. Определив, какая позиция выбрана, программа выполняет соответствующий оператор, устанавливающий цвет.

Microsoft-CD

Задания для самостоятельного выполнения

8.3. В системе программирования Visual Basic .NET создать проект «Пожарная тревога», который описан в параграфе. Готовый проект содержится в самораспаковывающемся архиве Пожарная_тревога.exe.

8.4. В системе программирования Visual Basic .NET создать проект «Выбор цвета», который описан в параграфе. Готовый проект содержится в самораспаковывающемся архиве Выбор_цвета.exe.

8.5. Булевы операции в коде

Мы уже разобрались, как с помощью булевых операций объединять выражения для решения сложных логических задач. Давайте разберем пример, в котором операция AND используется для одновременной проверки выполнения двух условий.

Модифицируем приложение «Выбор цвета», добавив в него флажок, который позволит отключать изменение цвета. С помощью операции AND создадим составное условие, которое будет проверять установку переключателей и состояние флажка.

Откройте приложение «Выбор цвета». Добавьте на форму флажок. Измените значение его свойства Text на Цвет существует. Отредактируйте код обработчика нажатия кнопки Button1. Замените код в нем на следующий:

```
Dim myColor As System.Drawing.Color
If RadioButton1.Checked = True And CheckBox1._
Checked = True Then
    myColor = System.Drawing.Color.Red
End If
If RadioButton2.Checked = True And_
CheckBox1.Checked = True Then
    myColor = System.Drawing.Color.Green
End If
```

Помните, операцию AND используют, если проверяют, равны ли оба операнда True. Если нужно проверить, равен ли хотя бы один операнд True, используют операцию OR.

```
If RadioButton3.Checked = True And CheckBox1._
Checked = True Then
    myColor = System.Drawing.Color.Blue
End If

Form.ActiveForm.BackColor = myColor
```

Постройте и запустите проект. Нажмите кнопку с надписью *Установить цвет*. Ничего не произойдет! Цвет не изменится. А теперь установите флажок с надписью *Цвет существует* и еще раз нажмите кнопку с надписью *Установить цвет*. Форма и элементы управления станут красными. Вот что происходит в программе. В условие в блоке If...Then мы добавили следующий код:

```
And CheckBox1.Checked = True
```

Это дополнительное условие. Теперь, чтобы цвет изменился, должны выполняться оба условия. Например:

```
RadioButton2.Checked = True And CheckBox1._
Checked = True
```

Оба этих выражения должны быть равны True. Помните, True AND True = True. Если хотя бы одно из выражений, объединяемых операцией AND, равно False, то результат тоже будет равен False, и цвет не изменится.

Задания для самостоятельного выполнения

Microsoft-CD

8.5. В системе программирования Visual Basic .NET создать проект «Выбор цвета-2», который описан в параграфе. Готовый проект содержится в самораспаковывающемся архиве Выбор_цвета_2.exe.

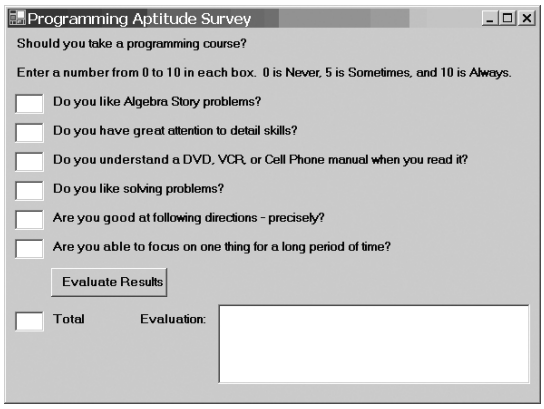
8.6. В системе программирования Visual Basic .NET создать проект «Выбор подарка», который поможет покупателям выбрать подарки. Готовый проект содержится в самораспаковывающемся архиве Выбор_подарка.exe.

1. Создайте форму и поместите на нее выпадающий список `Combobox`. Задайте для его свойства `Text` значение «Список подарков». Заполните коллекцию `Items` названиями подарков.
2. Поместите на форму текстовое поле. Сделайте его многострочным.
3. При выборе подарков в выпадающем списке, в текстовое поле должны быть выведены названия, как минимум, двух предметов.
4. Текст выбранного пункта содержится в свойстве `SelectedItem`.
5. Если вы хотите, чтобы два предмета выводились в разные строки в текстовом поле, поместите между ними специальную константу Visual Basic, например так:

   ```
   TextBox1.Text = "A" & vbNewLine & "B".
   ```

   Разумеется, вместо `A` и `B` нужно выводить названия пунктов выпадающего списка.

8.7. В системе программирования Visual Basic .NET создать проект «Проверка», который осуществит проверку склонности к программированию. Создайте форму, похожую на эту:



При нажатии кнопки с надписью *Evaluate Results* сложите числа из 6 текстовых полей. Поскольку содержимое текстового поля — строка, а не число, ее нужно будет преобразовать в число. Visual Basic .NET позволяет делать это с использованием функции `Val()`.

Если результат больше 40, выведите в текстовом поле сообщение, что учащийся — прекрасный кандидат для прохождения курса программирования.

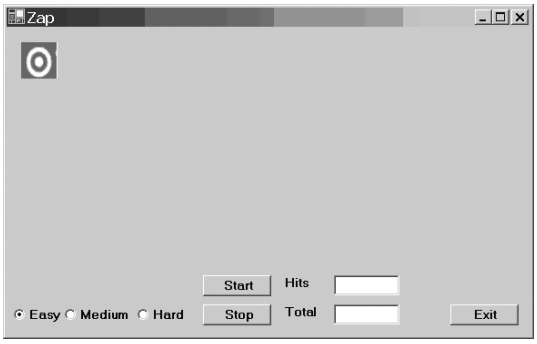
Если результат в пределах 20–39, нужно сообщить, что успешность обучения не гарантируется, но можно попробовать.

Если результат меньше 20, то предложите ученику выбрать другое занятие.

А каков ваш собственный результат?

8.8. В системе программирования Visual Basic .NET создать проект «Игра», в котором нужно попасть по появляющейся и исчезающей мишени.

1. Измените размеры формы: высоту (`Height`) на `250`, а ширину (`Width`) на `500`. Назовите форму `Zap`. Добавьте на нее элементы управления, чтобы она выглядела так, как на рисунке:



2. Цель игры — щелкнуть по движущейся мишени. Код будет двигать мишень, чтобы по ней было трудно попасть курсором. Кнопка с надписью *Start* инициализирует и запускает игру. Кнопка с надписью *Stop* заверша-

ет игру. Кнопка с надписью *Exit* закрывает программу. Три переключателя позволяют выбирать уровень сложности. Добавьте элемент `PictureBox` для размещения рисунка мишени.

3. Рисунок мишени, мишень.gif, хранится на диске. Выберите этот рисунок для отображения в элементе `PictureBox`. Данный элемент служит для отображения картинок. Для свойства `SizeMode` выберите значение `StretchImage`.
4. Программа использует элемент `Timer`, чтобы заставить код выполняться через определенные интервалы времени. Создав форму, найдите элемент `Timer` в окне *Область элементов* (*Toolbox*) и выполните по нему двойной щелчок. Элемент `Timer` является частью формы, но невидим пользователю.
5. Объявите переменную типа `Integer` для подсчета удачных попаданий.
6. Объявите переменную типа `Integer` для подсчета общего количества попыток.
7. В обработчике нажатия кнопки с надписью *Start* создайте код, который будет делать следующее:
   - Сбрасывать в 0 значения переменных-счетчиков попаданий и попыток.
   - Устанавливать для свойства `Enable` элемента управления `Timer` значение `True`.
   - Если выбрана позиция переключателя `RadioButton1`, устанавливать для свойства `Interval` элемента `Timer` значение `900`, а высоты и ширины рисунка — `40`.
   - Если выбрана позиция переключателя `RadioButton2`, устанавливать для свойства `Interval` элемента `Timer` значение `850`, а высоты и ширины рисунка — `35`.

- Если выбрана позиция переключателя RadioButton3, устанавливать для свойства Interval элемента Timer значение 800, а высоты и ширины рисунка — 30.

8. В обработчике нажатия кнопки с надписью *Stop*, установите для свойства Enable элемента Timer значение False.
9. В обработчике нажатия кнопки с надписью *Exit* завершите программу командой End.
10. В обработчик щелчка по элементу PictureBox добавьте код, который добавит единицу к значению счетчика Hits и отобразит это значение в текстовом поле «Hits».
11. В обработчик события Timer (сделайте двойной щелчок по этому элементу, чтобы создать обработчик), добавьте следующий код, чтобы мишень двигалась. Он подразумевает, что размер формы равен 250 × 550 единиц:

```
Dim MyRandomGenerator As System.Random
MyRandomGenerator = New System.Random
Dim RandomX As Integer
Dim RandomY As Integer
RandomX = MyRandomGenerator.Next(1, 550)
RandomY = MyRandomGenerator.Next(1, 250)
PictureBox1.SetBounds(RandomX, RandomY,_
Me.Width, Me.Height, System.Windows._
Forms.BoundsSpecified.Location)
```

12. После этого кода добавьте в обработчик код, добавляющий единицу к значению счетчика попыток и помещающий это значение в TextBox «Total».
13. В начале кода обработчика события Timer сделайте цвет фона формы серым. При попадании в мишень цвет фона должен стать красным.

Тест по теме «Ветвление: неполная форма»

1. 

После какого ключевого слова в операторе `If...Then` размещено условие?

- ❍ `End If`
- ❍ `Then`
- ❍ `If`
- ❍ `Else`

2. 

Что означает операция сравнения <>?

- ❍ *Больше или равно*
- ❍ *Меньше или равно*
- ❍ *Меньше*
- ❍ *Не равно*

3. 

Когда логическая операция `And` возвращает значение `True`?

- ❍ *Если равно `True` значение левого или правого операнда*
- ❍ *Если значения обоих операндов равны `True`*
- ❍ *Если значения обоих операндов не равны `True`*
- ❍ *При любом значении операндов*

4. 

Когда логическая операция `OR` возвращает значение `False`?

- ❍ *Если равно `False` значение левого или правого операнда*
- ❍ *Если значения обоих операндов равны `False`*
- ❍ *Если значения обоих операндов не равны `False`*
- ❍ *При любом значении операндов*

Глава 9

Ветвление: полная форма

Microsoft В 1995 году была создана операционная система для персональных компьютеров с графическим интерфейсом Windows 95, которая содержала утилиты для работы в Интернете. Графический интерфейс этой системы используется практически без изменений в последующих версиях операционных систем для персональных компьютеров Windows 98/Me.

9.1. Вложенные операторы If...Then

Внутрь оператора If…Then можно поместить еще один оператор If…Then. Второй оператор If…Then будет выполняться, только если условие в первом выполняется. Второй оператор находится внутри первого. Говорят, что оператор If…Then вложен внутрь другого оператора.

Создайте новое приложение Windows и назовите его «Флажки». Добавьте на форму два флажка и кнопку. Измените значение свойства Text кнопки на Флажки установлены?. Выполните двойной щелчок по кнопке, чтобы отредактировать обработчик ее нажатия. Добавьте в обработчик следующий код:

```
If CheckBox1.Checked = True Then
    If CheckBox2.Checked = True Then
        MessageBox.Show("Оба флажка_
        установлены.")
  End If
End If
```

Постройте и запустите приложение. Установите первый флажок и нажмите кнопку с надписью *Флажки установлены?*. Что произойдет? Ничего. Теперь установите второй флажок и снова нажмите кнопку с надписью *Флажки установлены?*. На этот раз появится окно с сообщением «Оба флажка установлены.» Как видите, в первой строке кода есть условие:

```
If CheckBox1.Checked = True Then
```

За этим условием идет оператор, который будет выполняться, только если выполняется условие. Это еще один оператор If…Then, вложенный в первый. Во вложенном операторе If…Then есть свое условие:

```
If CheckBox2.Checked = True Then
```

Можно вкладывать друг в друга столько операторов `If…Then`, сколько захотите, но больше трех-четырех вложенных операторов будет трудно понять, и в них трудно будет находить ошибки. Обязательно делайте отступы, используя вложенные операторы.

В этот оператор вложен следующий:

```
MessageBox.Show("Оба флажка установлены.")
```

Так что же происходит? Если условие в первом операторе выполняется (флажок `CheckBox1` установлен), то выполняется вложенный оператор. Это еще один оператор `If…Then`. Если условие во втором операторе тоже выполняется (флажок `CheckBox2` установлен), выполняется следующий вложенный оператор, и на экран выводится окно сообщения. Оба условия должны выполняться, иначе окно сообщения не будет выводиться. И `Checkbox1`, и `Checkbox2` должны быть установлены.

Обратите внимание на то, что `End If` для первого оператора `If…Then` идет после `End If` для второго `If…Then`. С помощью вложения можно совмещать множество операторов `If…Then`, чтобы какой-то код выполнялся только при выполнении всех условий.

Microsoft-CD

Задания для самостоятельного выполнения

9.1. В системе программирования Visual Basic .NET создать проект «Флажки», описанный в параграфе. Готовый проект содержится в самораспаковывающемся архиве Флажки.exe.

9.2. Противоположные условия

До сих пор действия в условном операторе выполнялись при выполнении условия. Если условие не выполнялось, ничего не происходило. А что, если нам нужно выполнить какие-то действия и в том случае, если условие не выполняется?

Ну, во-первых, можно написать еще один оператор `If…Then` с противоположным условием и поместить в него нужные строки кода. Давайте попробуем!

Создайте новое приложение Windows и назовите его «IfThenOtherwise». Добавьте на форму два флажка и кнопку. Измените значение свойства `Text` кнопки на `IfThen`. Поместите в обработчик ее нажатия следующий код:

```
If CheckBox1.Checked = True Then
    Form.ActiveForm.BackColor = System.Drawing._
    Color.Red
End If

If CheckBox1.Checked = False Then
    Form.ActiveForm.BackColor = System.Drawing._
    Color.Blue
End If
```

Постройте и запустите проект. Нажмите кнопку с надписью *IfThen*. Форма станет синей. Установите флажок `CheckBox1` и еще раз нажмите кнопку с надписью *IfThen*. Форма станет красной.

Как работает этот код? В первом операторе `If…Then` условие проверяет, установлен ли флажок: `CheckBox1.Checked = True`. Если условие выполняется, то цвет фона (`BackColor`) формы изменяется на красный. Второй оператор `If…Then` содержит условие, проверяющее, сброшен ли флажок: `CheckBox1.Checked = False` (противоположное условие). Если это условие выполняется, то цвет фона формы меняется на синий. Да, в коде есть повторения, но он работает! Его недостаток — присутствие двух почти одинаковых операторов. Такой подход отнимает лишнее время и может стать причиной ошибок.

Microsoft-CD

Задания для самостоятельного выполнения

9.2. В системе программирования Visual Basic .NET создать проект «IfThenOtherwise», описанный в параграфе. Готовый проект содержится в самораспаковывающемся архиве IfThenOtherwise.exe.

Использование двух операторов If…Then с противоположными условиями отнимает время и может привести к ошибкам. Вместо этого используйте новый вид оператора, If…Then…Else, чтобы упростить ваш код.

9.3. Оператор If...Then...Else

В языке Visual Basic .NET есть альтернативная возможность, избавляющая от необходимости повторять строки кода и упрощающая его понимание. Эта альтернатива — оператор If…Then…Else. Этот оператор требует только одного условия, но позволяет выбирать, какие операторы выполнять при выполнении условия, а какие — при невыполнении. Оператор If…Then…Else используется вместо двух одинаковых операторов If…Then с противоположными условиями. Вот его синтаксис:

```
If Условие Then
    Последовательность операторов 1
Else
    Последовательность операторов 2
End If
```

Как работает этот код? Сначала проверяется условие. Если оно выполняется, то выполняется Последовательность операторов 1. В противном случае выполняется Последовательность операторов 2.

Создадим программу «If-Then-Else» с помощью оператора If…Then…Else и докажем, что она работает точно так же, как и с двумя отдельными операторами If…Then с противоположными условиями.

Добавьте на форму еще одну кнопку. Измените значение ее свойства Text на If-Then-Else. В обработчик ее нажатия введите следующий код:

```
If CheckBox1.Checked = True Then
    Form.ActiveForm.BackColor = System.Drawing.
    Color.Red
Else
    Form.ActiveForm.BackColor = System.Drawing.
    Color.Blue
End If
```

Постройте и запустите проект. Нажмите кнопку с надписью *If-Then-Else*. Форма станет синей. Установите флажок CheckBox1 и еще раз нажмите

Обратите внимание на то, что операторы после `Then` и `Else` записаны с отступом. Visual Studio .NET делает это автоматически — оформленный таким образом код легче читать, и он выглядит аккуратнее.

кнопку с надписью *If-Then-Else*. Форма станет красной. Программа делает с помощью одного оператора то же самое, что раньше делала с помощью двух! Код проверяет, выполняется ли условие `CheckBox1.Checked = True`. Если да, то выполняется первый оператор и форма становится красной. В противном случае выполняется второй оператор и форма становится синей.

Давайте разберем еще один пример с `If…Then…Else`. Создадим программу «If-Then-Else-2», поместив в каждую часть оператора `If…Then…Else` не по одному оператору, а по несколько. В операторе `If…Then…Else` после `If…Then` и `Else` можно размещать сколько угодно строк кода.

Введите код обработчика нажатия кнопки `Button1`:

```
If CheckBox1.Checked = True Then
    Form.ActiveForm.BackColor = System.Drawing._
    Color.Red
    MessageBox.Show("I'm red.")
    TextBox1.Text = "I’m red.
Else
    Form.ActiveForm.BackColor = System.Drawing._
    Color.Blue
    MessageBox.Show("I'm blue.")
    TextBox1.Text = "I'm blue."
End If
```

Постройте и запустите проект. Нажмите кнопку с надписью *If-Then-Else*, а затем установите флажок `CheckBox1` и еще раз нажмите кнопку с надписью *If-Then-Else*.

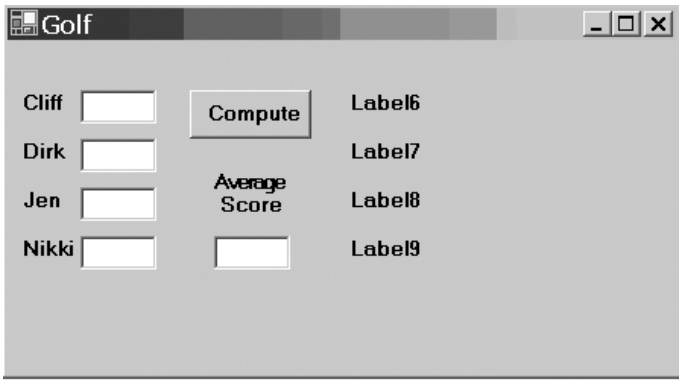
Microsoft-CD

Задания для самостоятельного выполнения

9.3. В системе программирования Visual Basic .NET создать проект «If-Then-Else», описанный в параграфе. Готовый проект содержится в самораспаковывающемся архиве If_Then_Else.exe.

9.4. В системе программирования Visual Basic .NET создать проект «If-Then-Else-2», описанный в параграфе. Готовый проект содержится в самораспаковывающемся архиве If_Then_Else_2.exe.

9.5. В системе программирования Visual Basic .NET создать проект «Гольф». Создайте форму вроде этой:



Когда программа запускается, свойства `Text` надписей должны быть пустыми, а пользователь должен ввести в текстовые поля количества очков, набранных участниками. При нажатии кнопки с надписью *Compute* должно вычисляться среднее количество очков. Поскольку содержимое текстового поля — строка, а не число, ее придется преобразовывать в число. Visual Basic .NET позволяет сделать это с помощью функции `Val()`, находящей числовое значение строки, которое можно присвоить переменной, например `X`.

```
X = Val(TextBox1.Text)
```

Для каждого игрока, если его число очков больше среднего, выведите сообщение вроде «Хороший результат» в соответствующей надписи, в противном случае выведите сообщение «Тренируйтесь».

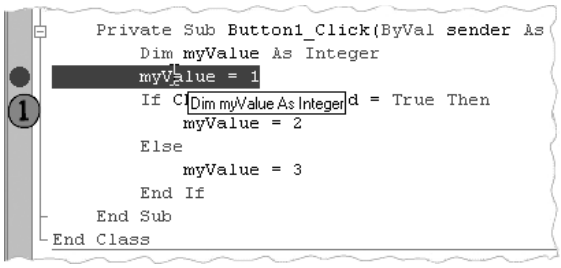
9.4. Пошаговое выполнение If

А теперь воспользуемся отладочными средствами Visual Studio .NET и пошагово выполним код в операторе `If…Then…Else`. Таким образом мы сможем увидеть, как он выполняется внутри программы.

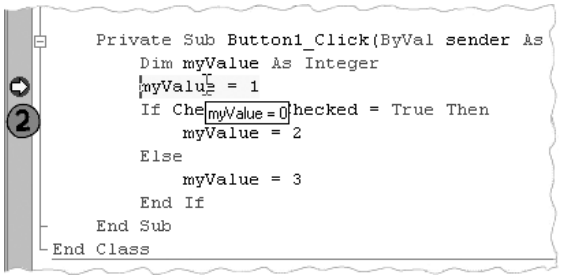
Создайте новое приложение Windows и назовите его «Step-In-If». Поместите на форму одну кнопку и один флажок. Установите для свойства `Checked` флажка значение `True`. Когда приложение запустится, флажок будет установлен. Выполните двойной щелчок по кнопке, чтобы отредактировать обработчик ее нажатия. Добавьте в обработчик следующий код:

```
Dim MyValue As Integer
MyValue = 1
If CheckBox1.Checked = True Then
    MyValue = 2
Else
    MyValue = 3
End If
```

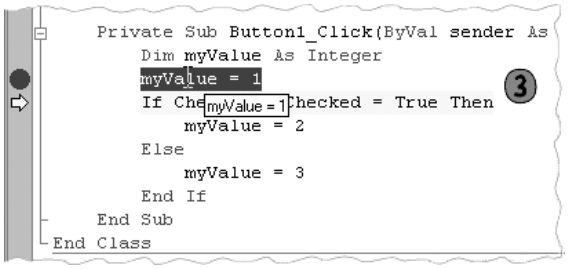
1. Установите точку останова в первой строке кода (`MyValue = 1`), выполнив двойной щелчок в серой области слева от этой строки или щелкнув правой кнопкой по этой строке и выбрав из контекстного меню пункт *Вставить точку останова* (*Insert Breakpoint*).



2. Постройте и запустите проект. Когда откроется форма, нажмите кнопку. Код выполнится до строки с точкой останова.



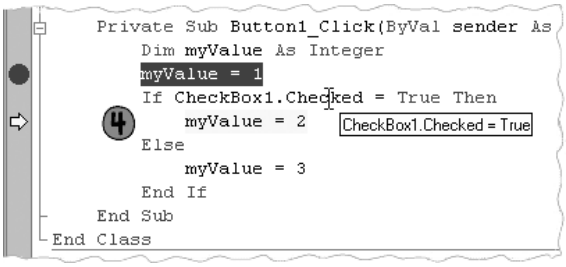
3. Нажмите клавишу {F11}, чтобы выполнить текущую строку кода `MyValue = 1`. Подсветится следующая строка.



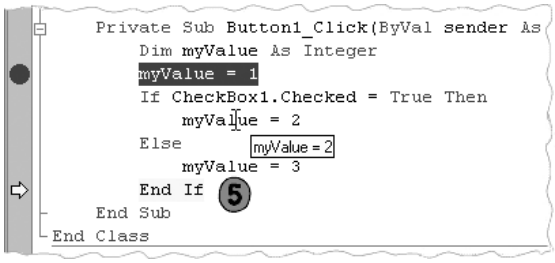
4. Еще раз нажмите клавишу {F11}, чтобы выполнить строку кода

```
If CheckBox1.Checked = True Then
```

Поскольку флажок `CheckBox1` установлен, будет подсвечена следующая строка кода — `MyValue = 2`.



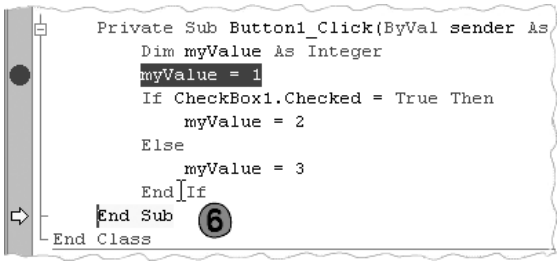
5. Нажмите клавишу {F11}, чтобы выполнить строку кода `MyValue = 2`. Будет подсвечена строка `End If`.



6. Нажмите клавишу {F11}, чтобы выполнить строку кода

```
End If
```

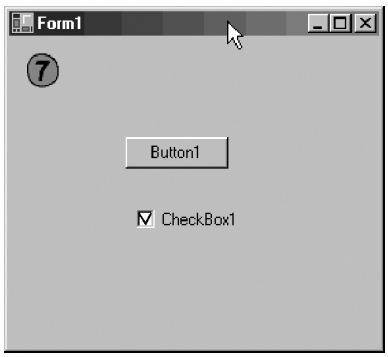
Будет подсвечена следующая строка.



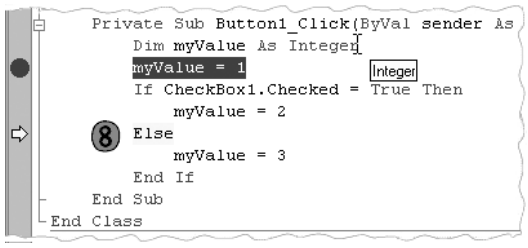
7. Нажмите клавишу {F11}, чтобы выполнить строку кода

```
End Sub
```

Опять появится форма.



8. Сбросьте флажок `CheckBox1` и нажмите кнопку. Повторите пошаговое выполнение оператора, нажимая клавишу {F11} и наблюдая, какие строки кода будут подсвечиваться и выполняться. На этот раз выполнится оператор после `Else`, т. е. строка `MyValue = 3`.



Вот и все! Теперь вы видели, как выполняется оператор `If…Then…Else` при работе программы!

Задания для самостоятельного выполнения

Microsoft-CD

9.6. В системе программирования Visual Basic .NET создать проект «Step-In-If». Провести пошаговое выполнение проекта так, как это описано в параграфе. Готовый проект содержится в самораспаковывающемся архиве Step_In_If.exe.

9.5. Операторы If в C# и J#

В языках C# и J# есть условные операторы, работающие точно так же, как `If…Then` и `If…Then…Else` в языке Visual Basic .NET. Но их синтаксис немного другой. Вот пример из C#:

```
int MyAge;
MyAge=15;
if (MyAge==15)
{
    MessageBox.Show("I am 15.");
    MessageBox.Show("I am still 15.");
}
```

Сначала мы объявляем целочисленную переменную MyAge и присваиваем ей начальное значение. Затем мы начинаем условный оператор с ключевого слова if — обратите внимание, что все буквы в нем строчные. За этим словом идет условие MyAge==3. Условие помещено в скобках. Вы, наверное, обратили внимание на два знака «равно» подряд. В языке C# так обозначается операция сравнения (в Visual Basic .NET она обозначается одним знаком). Операторы, которые должны выполняться при выполнении условия, помещены в фигурные скобки — {...}. Чтобы код было удобнее читать, некоторые программисты помещают фигурные скобки в отдельных строках. Заметьте, что каждый оператор заканчивается точкой с запятой. Кроме того, ключевое слово Then вообще не используется!

Код на языке J# будет точно таким же, как и на C#.

Давайте разберем оператор If…Then…Else на языке C#. Посмотрите, что есть общего между этими операторами в C# и Visual Basic .NET и чем они отличаются.

```
int MyAge;
MyAge=15;
if (MyAge==16)
{
    MessageBox.Show("I am 15.");
    MessageBox.Show("I am still 15.");
}
else
{
    MessageBox.Show("I am 16.");
    MessageBox.Show("I am still 16.");
}
```

Условие снова помещено в скобки. Заметьте, что на этот раз здесь два блока операторов — один после условия и один после else. Каждый блок помещен в фигурные скобки. Заметьте, что каждый оператор

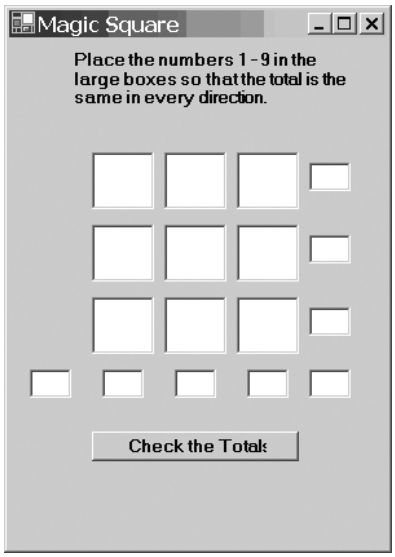
заканчивается точкой с запятой. Ключевое слово `Then` не используется.

В J# синтаксис оператора `If…Then…Else` точно такой же, как и в C#.

Задания для самостоятельного выполнения

Microsoft-CD

9.7. В системе программирования Visual Basic .NET создать проект «Магический квадрат». Создайте такую форму, как показано на рисунке, или, если жаль тратить время, используйте самораспаковывающийся архив Магический_квадрат.exe.



Создайте программный код, который числа от 1 до 9 размещает в массиве размером 3×3 так, чтобы сумма в каждой строке, каждом столбце и каждой диагонали была одинаковой.

При нажатии кнопки с надписью *Check the Totals* должны вычисляться суммы трех строк, трех столб-

цов и двух диагоналей, результаты должны отображаться в полях от TextBox10 до TextBox17 (от левого нижнего угла до верхнего правого). Поскольку содержимое текстового поля — строка, а не число, ее придется преобразовывать в число. Visual Basic .NET позволяет сделать это с использованием функции Val(), которая находит числовое значение строки.

Каждая сумма должна быть равна 15. Сумма чисел в девяти текстовых полях должна быть равна 45.

9.6. Булевы операции и операции сравнения в C# и J#

Вы должны знать еще кое-что о различиях между языком Visual Basic .NET и языками C# и J# при использовании операторов If…Then и If…Then…Else. Во-первых, булевы операции обозначаются по-другому. Вместо And, Or и Not, C# и J# используют &&, || и !:

Булева операция	Visual Basic.NET	C#	J#
И	And	&&	&&
Или	Or	||	||
Не	Not	!	!

Язык C#. Вот пример использования булевых операторов И (&&) и Не (!) в C#:

```
bool isBilled=true;
bool isLate=false;
bool isDone;
isDone = isBilled && !isLate;
MessageBox.Show(isDone.ToString());
```

Обратите внимание на ключевые слова `bool` (тип `Boolean`), `true` и `false` — они все набраны строчными буквами. Переменные `isBilled`, `isLate` и `isDone` объявлены как `bool` (тип `Boolean`). Обратите внимание на операцию И (`&&`) и операцию Не (`!`) в C#. Поскольку C# и J# не выполняют автоматических преобразований значений различных типов в строковые, используется метод `ToString`.

Язык J#. Вот такой же код на J#:

```
boolean isBilled=true;
boolean isLate=false;
boolean isDone;
isDone=isBilled && !isLate;
MessageBox.Show(System.Convert.ToString(isDone));
```

Код выглядит почти так же, как код на C#, но есть и некоторые отличия. Обратите внимание на то, как объявляется переменная `boolean` (тип `Boolean`). Все переменные — `isBilled`, `isLate` и `isDone` — объявлены как `boolean`. В коде на J# также по-другому указывается преобразование значения типа `Boolean` в строку — с помощью метода `System.Convert.ToString`.

Некоторые операции сравнения в языках C# и J# обозначаются не так, как в Visual Basic .NET. Вы уже видели, что проверка равенства обозначается == вместо =. Кроме того, проверка неравества в C# и J# обозначается != вместо <>. Вот таблица с обозначениями операций сравнения:

Операция сравнения	Visual Basic .NET	C#	J#
Равно	=	==	==
Больше чем	>	>	>
Меньше чем	<	<	<
Больше или равно	>=	>=	>=
Меньше или равно	<=	<=	<=
Не равно	<>	!=	!=

Посмотрите на код на языках J# или C#, который использует оператор сравнения «не равно»:

```
int MyAge;
MyAge=15;
if (MyAge!=16)
{
    MessageBox.Show("I am not 16.");
}
```

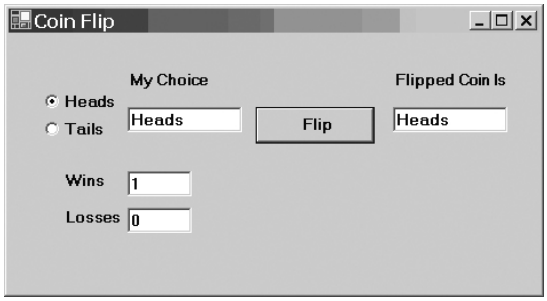
Когда выполняется этот код, выводится окно с сообщением «I am not 16.».

Как видите, различий между операторами `If…Then` и `If…Then…Else` в Visual Basic .NET и C# или J# немало, но есть и общие черты! Эти операторы делают то же самое, просто записываются по-другому.

Microsoft-CD

Задания для самостоятельного выполнения

9.8. В системе программирования Visual Basic .NET создать проект «Бросок монеты», имитирующий на компьютере броски монеты. Создайте такую форму, как показано на рисунке, или, если жаль тратить время, используйте самораспаковывающийся архив Бросок_монеты.exe.



Объявите переменные типа `Integer` для подсчета выигрышей и проигрышей. Прибавляйте едини-

цы к их значениям и отображайте эти значения в соответствующих текстовых полях.

В программном коде в зависимости от того, какая выбрана позиция переключателя, отображайте в `TextBox1.Text` или «Heads», или «Tails».

При нажатии кнопки с надписью *Flip* генерируйте случайное число с помощью такого кода:

```
Dim MyRandomGenerator As System.Random
MyRandomGenerator = New System.Random
Dim RanNum As Integer
 ' Generate random value between 0 and 2 - not
 including 2.
RanNum = MyRandomGenerator.Next(0, 2)
```

Если случайное число равно 0, то `TextBox2.Text` должно быть равно «Heads», иначе оно должно быть равно «Tails».

Если `TextBox1.Text` равно `TextBox2.Text`, то прибавьте 1 к количеству выигрышей, иначе прибавьте 1 к количеству проигрышей.

Отобразите количества выигрышей и проигрышей в соответствующих полях.

Добавьте поля для подсчета процентов выигрышей и проигрышей.

Тест по теме «Ветвление: полная форма»

1. 

Как рекомендуется записывать вложенные операторы `If`?

- ❍ *Помещать в квадратные скобки*
- ❍ *Помещать в фигурные скобки*
- ❍ *Выделять жирным шрифтом*
- ❍ *Выделять отступами*

2. 

Если оператор `If` вложен в другой оператор `If`, то сколько должно быть операторов `End If`?

- ❍ *0*
- ❍ *1*
- ❍ *2*
- ❍ *3*

3. 

Если условие в операторе `If` не выполняется, то какой код будет выполняться?

- ❍ *Код после оператора* `Then`
- ❍ *Код после оператора* `Else`
- ❍ *Никакой код не будет выполняться*
- ❍ *Будут выполняться оба кода*

4. 

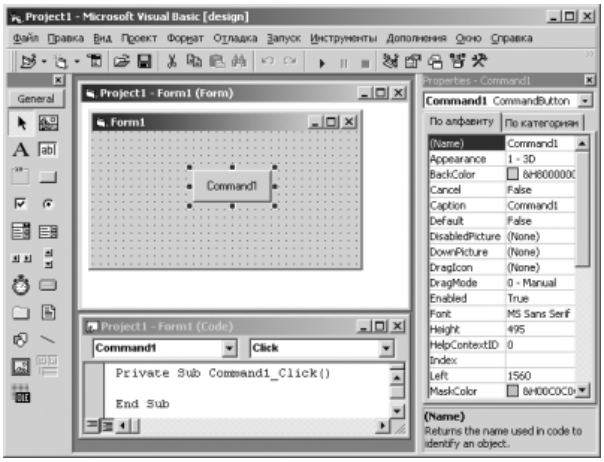
Сколько строк кода может следовать за операторами `Then` или `Else`?

- ❍ *0*
- ❍ *1*
- ❍ *2*
- ❍ *Сколько угодно*

Глава 10

Циклы со счетчиком

Microsoft В 1996 году был создан язык объектно-ориентированного программирования Visual Basic 5.0, а в 1998 году — Visual Basic 6.0. Эти языки получили широкое распространение, так как могли использоваться как программистами-профессионалами для разработки коммерческих приложений, так и начинающими программистами в процессе обучения.



10.1. Циклы For...Next

Циклы используются для многократного выполнения одних и тех же операторов кода. Иногда программе нужно повторять какие-то действия раз за разом, пока она не выполнит их нужное количество раз. Поэтому во всех языках программирования поддерживаются циклы. Циклы очень полезны. Во-первых, они уменьшают объем кода, который вам нужно писать. Повторяющиеся действия записываются в цикле только один раз. Во-вторых, они уменьшают вероятность ошибок.

Вот пример полезности циклов в программах. Предположим, что вам нужно сложить все числа от 1 до 100. Это можно сделать, написав одно длинное выражение, например:

```
Dim TotalCount As Integer
TotalCount = 0
TotalCount = 1 + 2 + 3 + 4 'и так далее
```

Это очень неудобно!

А как насчет вот такого?

```
Dim TotalCount As Integer
TotalCount = 0
TotalCount = TotalCount + 1
TotalCount = TotalCount + 1
TotalCount = TotalCount + 1
TotalCount = TotalCount + 1
'и так далее
```

Тоже плохо. Эту строку пришлось бы повторить 100 раз! А если бы нужно было просуммировать числа от 1 до 1000? Как бы вы сделали это?

Оператор For...Next. Вот здесь-то нам и пригодятся циклы. Как мы уже говорили, циклы есть во всех языках программирования, и они позволяют исполнять один и тот же код множество раз. Часто

вы знаете заранее, сколько раз нужно выполнить код. В нашем примере нужно просуммировать все числа от 1 до 1000, поэтому код нужно выполнить 1000 раз. Если вы знаете, сколько раз должен выполниться код, используйте цикл с предварительно заданным числом повторений.

В Visual Basic .NET, цикл с предварительно заданным числом повторений создается с помощью оператора `For…Next`. Вот синтаксис такого цикла:

```
Dim Счетчик As Integer
For Счетчик = НачальноеЗначение To Конечное
Значение
    Последовательность операторов
Next
```

Обратите внимание на то, что `For`, `Next` и `To` — это ключевые слова. Они выделены синим цветом и начинаются с заглавных букв. `Счетчик` (счетчик цикла) — переменная, которую нужно объявлять как целочисленную. Ее нужно объявить, прежде чем использовать в цикле `For…Next`. (Для нее можно задать любое имя.) Переменная `счетчик` отслеживает, сколько раз выполнился цикл. К значению этой переменной можно обращаться внутри цикла. `Начальное значение` и `конечное значение` должны быть целыми числами, целочисленными переменными, или выражениями целочисленного типа, например 3 + 1 или *X* + 1. При каждом выполнении цикла значение переменной `счетчик` увеличивается на 1. Когда значение счетчика становится больше, чем `конечное значение`, выполнение цикла завершается.

Давайте напишем код, в котором используется цикл `For…Next`. Для начала мы создадим цикл, который выполнится два раза и будет выводить значение переменной-счетчика при каждом выполнении. Создайте новое приложение Windows и назовите его «For-Next». Поместите на форму кнопку. Выполните по ней двойной щелчок, чтобы отредактировать

Лучше выделять код между операторами `For` и `Next` с помощью отступа. Делая отступы в коде, вы быстрее разберетесь, какие операторы будут повторно выполняться в цикле. Код с отступами проще читать и отлаживать.

обработчик ее нажатия. Добавьте в обработчик следующий код:

```
Dim LoopCounter As Integer
For LoopCounter = 1 To 2
    MessageBox.Show(LoopCounter)
Next
```

Постройте и запустите проект. Нажмите кнопку. Появится сообщение с цифрой 1, текущим значением переменной `LoopCounter`. Нажмите кнопку *OK*. Появится сообщение с цифрой 2, текущим значением переменной `LoopCounter`. Нажмите кнопку *OK*. Выполнение цикла закончится, поэтому больше сообщений не будет.

Вот как это работает. В коде вы объявили переменную `LoopCounter` типа `Integer`. В операторе `For` вы проинициализировали `LoopCounter` значением 1 и задали для ее значения верхний предел 2. В цикле будут выполняться операторы, помещенные между `For` и `Next`. При каждом выполнении цикла значение `LoopCounter` увеличивается на 1. В окне сообщения будет выведено текущее значение переменной `LoopCounter` (сначала 1, потом 2). Когда значение `LoopCounter` достигнет 3, выполнение цикла завершится и окно сообщения выводиться больше не будет. Конец цикла!

По умолчанию значение счетчика увеличивается на 1 при каждом выполнении цикла. Можно сделать так, чтобы оно увеличивалось на бóльшую величину, используя ключевое слово `Step`, как в этом примере:

```
Dim LoopCounter As Integer
For LoopCounter = 0 To 100 Step 5
      MessageBox.Show(LoopCounter)
Next
```

Здесь значение переменной `LoopCounter` увеличивается на 5 при каждом выполнении цикла. В ок-

нах сообщений будут выводиться значения 0, 5, 10, 15, и т. д.

Microsoft-CD

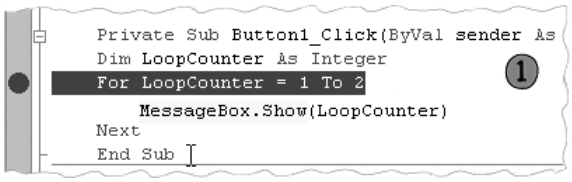
Задания для самостоятельного выполнения

10.1. В системе программирования Visual Basic .NET создать проект «For-Next», описанный в параграфе. Готовый проект содержится в самораспаковывающемся архиве For_Next.exe.

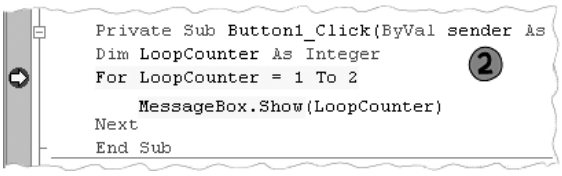
10.2. Пошаговое выполнение цикла For...Next

Давайте изучим цикл `For...Next`, который вы только что написали, с помощью отладочных инструментов Visual Studio .NET. Мы будем пошагово выполнять цикл. Вот что для этого нужно сделать:

1. Поставьте точку останова в строке кода, содержащей оператор `For`. Постройте и запустите приложение.



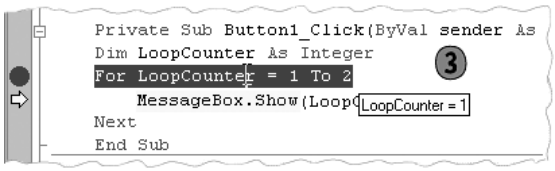
2. Откроется форма. Нажмите кнопку `Button1`. Код выполнится до точки останова. Строка с точкой останова будет подсвечена.



3. Нажмите клавишу {F11}. Подсвеченная строка кода выполнится.

```
For LoopCounter = 1 To 2
```

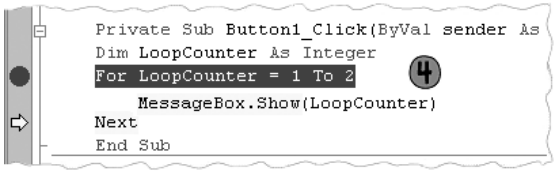
Подведите курсор мыши к переменной `LoopCounter` в строке `For LoopCounter = 1 To 2`. Ее значение равно 1.



4. Нажмите клавишу {F11}. Подсвеченная строка кода выполнится.

```
MessageBox.Show(LoopCounter)
```

В окне сообщения будет цифра 1. Нажмите кнопку *OK*, чтобы закрыть окно сообщения.



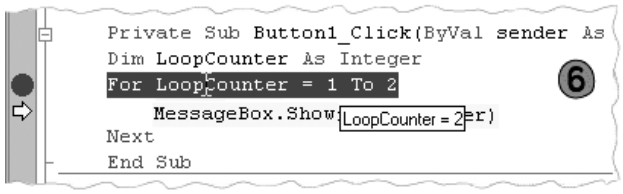
5. Нажмите клавишу {F11}. Подсвеченная строка кода выполнится.

```
Next
```

6. Первое выполнение цикла `For` закончено. Нажмите клавишу {F11}. Подсвеченная строка кода выполнится.

```
For LoopCounter = 1 To 2
```

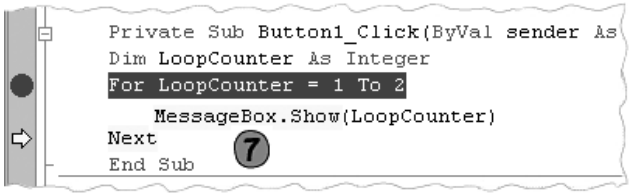
Подведите курсор к переменной `LoopCounter` в операторе `For`. Значение переменной будет равно 2.



7. Нажмите клавишу {F11}. Подсвеченная строка кода выполнится.

```
MessageBox.Show(LoopCounter)
```

В окне сообщения будет цифра 2. Нажмите кнопку *OK*, чтобы закрыть окно сообщения.



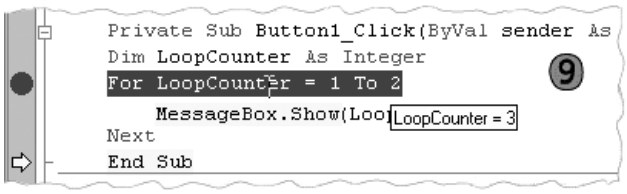
8. Нажмите клавишу {F11}. Подсвеченная строка кода выполнится.

```
Next
```

9. Выполнение цикла закончено. Нажмите клавишу {F11}. Подсвеченная строка выполнится.

```
For LoopCounter = 1 To 2
```

Подведите курсор к переменной `LoopCounter` в операторе `For`. Значение переменной будет равно 3.



10. Нажмите клавишу {F11}. Подсвеченная строка кода выполнится.

```
End Sub
```

Форма появится опять.

Вы только что пошагово выполнили код в цикле `For…Next`. Вы увидели, как этот цикл выполняется в программе!

Microsoft-CD

Задания для самостоятельного выполнения

10.2. В системе программирования Visual Basic .NET провести пошаговое выполнение проекта «For-Next» так, как это описано в параграфе. Готовый проект содержится в самораспаковывающемся архиве For_Next.exe.

10.3. Проекты с использованием For...Next

Сложение чисел от 1 до 1000. С помощью цикла `For…Next` можно складывать большое количество чисел. Создайте новое приложение Windows и назовите его «Сложение». Поместите на форму кнопку и измените значение ее свойства `Text` на `Сложить`. Выполните по ней двойной щелчок, чтобы отредактировать обработчик ее нажатия. Добавьте в обработчик следующий код:

```
Dim Counter As Integer
Dim Total As Integer = 0
For Counter = 1 To 1000
    Total = Total + Counter
Next
MessageBox.Show(Total)
```

Когда цикл `For…Next` заканчивает выполняться, выполняется первая строка кода после оператора `Next`. В нашем примере это строка `MessageBox.Show(Total)`.

Постройте и запустите проект. Нажмите кнопку с надписью *Сложить*. Появится окно сообщения с суммой, полученной в результате сложения всех чисел от 1 до 1000. Вы написали код, и программа вычислила ответ быстрее, чем вы успели вытащить калькулятор!

Как этот код работает? Сначала мы объявили две целочисленных переменных: `Counter` и `Total`. `Counter` отслеживает, сколько раз выполнился цикл. В `Total` хранится сумма сложенных чисел. В операторе `For` начальное значение `Counter` задано равным 1, а конечное значение — равным 1000. Это потому, что мы хотим выполнить цикл 1000 раз, суммируя числа от 1 до 1000. Оператор, который нужно повторять много раз, прибавляет текущее значение `Counter` к значению `Total` и сохраняет результат в `Total`. При каждом выполнении цикла значение `Counter` прибавляется к значению `Total` (0 + 1 = 1, 1 + 2 = 3, 3 + 3 = 6, 6 + 4 = 10, и т. д.). Когда `Counter` достигает значения 1000, выполнение цикла заканчивается. Выполняется строка кода после оператора `Next`. В этой строке вызывается окно сообщения, в котором выводится значение `Total`.

Сложение строк. Помните, внутрь цикла `For…Next` можно помещать какой угодно код. Можно автоматически составлять длинные текстовые строки, выводить снова и снова раздражающие сообщения или изменять свойства в зависимости от значения счетчика цикла. Вот еще один пример использования цикла.

Создайте новое приложение Windows и назовите его «Сложение строк». Поместите на форму кнопку и измените значение ее свойства `Text` на `Сложить строки`. Выполните по ней двойной щелчок, чтобы отредактировать обработчик ее нажатия. Добавьте в обработчик следующий код:

```
Dim Message As String
Dim i As Integer = 0
Message = "СТРОКА."
For i = 1 To 5
    MessageBox.Show(Message)
    Message = Message & vbNewLine & Message
Next
```

Постройте и запустите проект. Нажмите кнопку с надписью *Сложить строки*. В окне сообщений будет выведено сообщение «СТРОКА.». В окне сообщений щелкните пять раз по кнопке *OK*, цикл выполнится пять раз. Обратите внимание на то, во сколько раз увеличилось сообщение.

Цикл и вложенный условный оператор. Создайте проект «Цвет формы». Поместите на форму кнопку и измените значение свойства `Text` этой кнопки на `Установить цвет`. Выполните двойной щелчок по ней, чтобы отредактировать обработчик ее нажатия. Добавьте в обработчик следующий код:

```
Dim i As Integer = 0
For i = 2 To 6
    If i < 3 Or i > 5 Then
      MessageBox.Show(i)
      Form1.ActiveForm.BackColor = System._
      Drawing.Color.Red
    Else
      MessageBox.Show(i)
      Form1.ActiveForm.BackColor = System._
      Drawing.Color.Blue
End If
Next
```

Постройте и запустите проект. Нажмите кнопку с надписью *Установить цвет*. В окнах сообщения

будет выводиться текущее значение счетчика цикла. Заметьте, что на этот раз минимальное значение счетчика равно 2, а максимальное — 6. Оператор `If…Then…Else` внутри цикла `For…Next` решает, как раскрасить форму в зависимости от значения счетчика цикла. В цикле выводится окно сообщения, которое позволяет отслеживать значение счетчика цикла. Кроме того, вывод окна сообщения замедляет работу цикла, чтобы можно было заметить изменение цвета формы.

Microsoft-CD

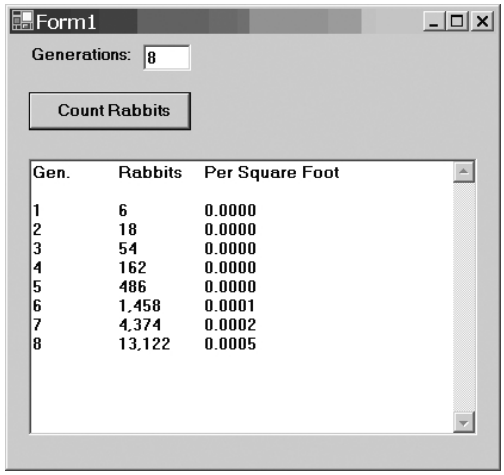
Задания для самостоятельного выполнения

10.3. В системе программирования Visual Basic .NET создать проект «Сложение», описанный в параграфе. Готовый проект содержится в самораспаковывающемся архиве Сложение.exe.

10.4. В системе программирования Visual Basic .NET создать проект «Сложение строк», описанный в параграфе. Готовый проект содержится в самораспаковывающемся архиве Сложение_строк.exe.

10.5. В системе программирования Visual Basic .NET создать проект «Цвет формы», описанный в параграфе. Готовый проект содержится в самораспаковывающемся архиве Цвет_формы.exe.

10.6. В системе программирования Visual Basic .NET создать проект «Подсчет кроликов», вычисляющий как будет расти популяция кроликов в каждом поколении. Создайте такую форму, как показано на рисунке, или, если жаль тратить время, используйте самораспаковывающийся архив Подсчет_кроликов.exe.



Представим себе, что есть остров, на котором для кроликов достаточно еды и нет хищников. Предположим, что на остров поместили пару кроликов и что в каждом их выводке будет четыре кролика. Тогда после первого поколения будет 2 + 4 = 6 кроликов. После двух поколений будет 6 + 12 = 18 кроликов.

Указывайте номер поколения, количество кроликов, и количество кроликов на квадратный фут площади, полагая, что остров имеет площадь в одну квадратную милю, а в миле 5280 футов.

Предоставьте пользователю возможность указывать, для скольких поколений кроликов выполнять расчет.

Чтобы результаты было легче читать, используйте константы Visual Basic `vbTab` и `vbNewLine`.

Чтобы форматировать вывод, используйте функцию `Format`, например:

```
VariableX = Format(VariableX, "###, ###, ###,
###, ##0")
```

или

```
VariableY = Format(VariableY, ###, ##0.0000")
```

10.4. Вложенные циклы

Знаете ли вы, что можно поместить один цикл `For…Next` внутрь другого? Внутренний цикл `For…Next` будет выполняться раз за разом при каждом выполнении внешнего цикла `For…Next`. Так сколько же раз выполнится код внутри внутреннего цикла? Давайте рассмотрим пример, чтобы определить ответ на этот вопрос.

Создайте новое приложение Windows и назовите его «Цикл в цикле». Поместите на форму кнопку. Выполните по ней двойной щелчок, чтобы отредактировать обработчик ее нажатия. Добавьте в обработчик следующий код:

```
Dim i As Integer
Dim k As Integer
For i = 1 To 3
    Dim Total As Integer = 0
    For k = 1 To 4
      Total = Total + 1
    Next
Next
MessageBox.Show("Total= " & Total)
```

Постройте и запустите проект. Нажмите кнопку. Появится окно сообщения с текстом «Total = 12». Это общее количество выполнений кода внутри внутреннего цикла. Посмотрите на код. Вы заметили, что 12 — это 3 (количество выполнений внешнего цикла) умножить на 4 (количество выполнений внутреннего цикла)? Правильно! При каждом выполнении внешнего цикла внутренний цикл исполняется 4 раза. Поскольку внешний цикл выполнился 3 раза, внутренний цикл выполнился $4 \cdot 3 = 12$ раз.

Задания для самостоятельного выполнения

Microsoft-CD

10.7. В системе программирования Visual Basic .NET создать проект «Цикл в цикле», подсчитывающий количество повторений тела вложенного цикла. Готовый проект содержится в самораспаковывающемся архиве Цикл_в_цикле.exe.

10.5. Выход из циклов

Иногда нужно прервать выполнение цикла до того, как значение счетчика достигнет верхнего предела. Это можно сделать с помощью оператора `Exit For`. Данный оператор помещается внутрь оператора `If…Then` внутри цикла. Условие в операторе `If…Then` указывает коду, когда нужно прервать выполнение цикла. Давайте напишем код, в котором будет использоваться оператор `Exit For`.

Создайте новое приложение Windows и назовите его «Выход из цикла». Поместите на форму кнопку. Измените значение свойства `Text` этой кнопки на `Выйти из цикла`. Выполните по ней двойной щелчок, чтобы отредактировать обработчик ее нажатия. Добавьте в обработчик следующий код:

```
Dim i As Integer = 0
For i = 1 To 5
    MessageBox.Show("i inside= " & i)
    If i = 3 Then
      Exit For
    End If
Next
MessageBox.Show("i outside= " & i)
```

Постройте и запустите проект. Нажмите кнопку с надписью *Выйти из цикла* несколько раз. Будут появляться окна сообщения со значениями счетчика цикла. Когда это значение станет равным 3, условие в блоке `If…Then` выполнится и выполнится оператор `Exit For`. Выполнение цикла завершится без выполнения оставшихся повторов. Обратите внимание, что окна с сообщением «i inside = 4» нет. Когда цикл завершится, выполнится код после оператора `Next`. Появится окно с сообщением «i outside = 3».

Вот еще один пример использования `Exit For`. Мы изменим программу «Выход из цикла». Добавьте на форму флажок и еще одну кнопку. Измените значение свойства `Text` кнопки на `Проверить`. Выполните двойной щелчок по новой кнопке, чтобы

отредактировать обработчик ее нажатия. Измените код обработчика:

```
Dim Index As Integer
For Index = 1 To 5
    If Index = 4 Then
      CheckBox1.Checked = True
    End If
    If CheckBox1.Checked = True Then
      MessageBox.Show("Exit on: " & Index)
      Exit For
    End If
Next
```

Постройте и запустите проект. Нажмите кнопку с надписью *Проверить*. Цикл `For…Next` будет выполняться, пока значение переменной `Index` не станет равным 4. Затем свойство `Checked` флажка `CheckBox1` будет установлено в `True`. Поскольку `Checkbox1` будет установлен, будет выведено окно сообщения со значением переменной `Index` и выполнится оператор `Exit For`. Выполнение цикла завершится.

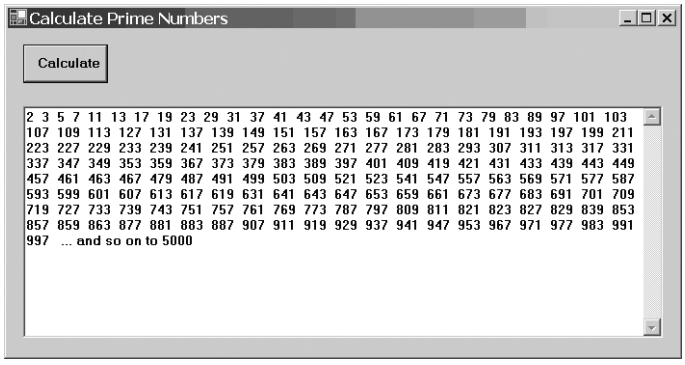
Еще раз нажмите кнопку с надписью *Проверить*, не сбрасывая флажок `CheckBox1`. Что произойдет на этот раз? Оператор `Exit For` будет выполнен при `Index = 1`.

Задания для самостоятельного выполнения

Microsoft-CD

10.8. В системе программирования Visual Basic .NET создать проект «Выход из циклов», описанный в параграфе. Готовый проект содержится в самораспаковывающемся архиве Выход_из_циклов.exe.

10.9. В системе программирования Visual Basic .NET создать проект «Простые числа», находящий все простые числа от 2 до 5000. Создайте такую форму, как показано на рисунке, или, если жаль тратить время, используйте самораспаковывающийся архив Простые_числа.exe.



Простое число — это число, которое нацело (без остатка) делится только на 1 и на само себя. Первые простые числа — это 2, 3, 5, 7. Числа 4, 6, 8, и 9 — не простые, потому что они нацело делятся, например, на 2 или, в случае 9, на 3.

При нажатии кнопки с надписью *Calculate* должны вычисляться простые числа до 5000. Они должны выводиться в текстовом поле.

Поместите цикл `For` внутрь другого цикла `For`. Внешний цикл должен перебирать числа от 2 до 5000. Внутренний цикл должен проверять, делится ли число на все числа от 2 до самого этого числа минус 1.

Вам, вероятно, понадобится переменная для того, чтобы помечать, является ли проверяемое в данный момент число простым. Ее значение можно установить равным `True` перед запуском начального цикла, а если окажется, что число не простое, изменить значение на `False`.

Внутри внутреннего цикла следует проверять, делится ли число нацело на значение переменной-счетчика внутреннего цикла. Если да, значит, оно не простое.

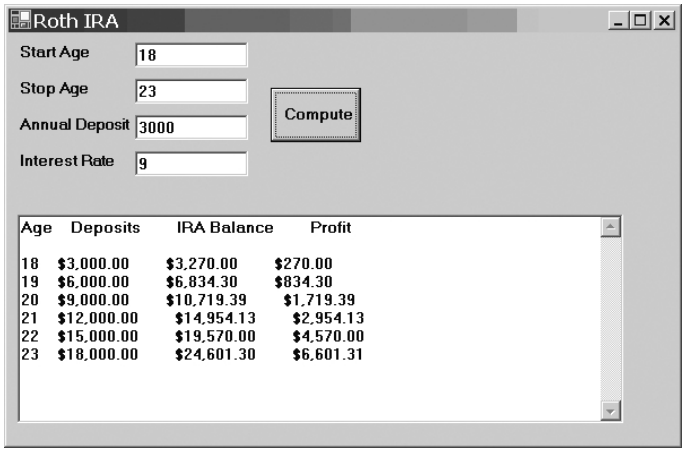
Предположим, что у нас есть числа X и Y. Как узнать, есть ли остаток от деления X / Y? Есть несколько способов. Поскольку мы проверяем ваше

умение обращаться с циклами, а не ваше знание математики, мы покажем вам простой способ. Это способ с использованием операции `Mod`. Предположим, что у нас есть переменная `Remainder` (остаток):

```
Remainder = X Mod Y
```

Если по завершении внутреннего цикла для следящей переменной все еще установлено значение `True`, значит, проверяемое число — простое. Между выводимыми простыми числами в текстовом поле лучше вставлять пробелы.

10.10. В системе программирования Visual Basic .NET создать проект «Пенсия», который будет вычислять суммы, накопившиеся на пенсионном счету. Создайте такую форму, как показано на рисунке, или, если жаль тратить время, используйте самораспаковывающийся архив Пенсия.exe.



На форме указывается возраст, в котором вы начинаете откладывать деньги, возраст, в котором вы прекращаете это делать, сумма, откладываемая каждый год, и процент по вкладу.

Мы изучали циклы, и похоже, здесь очень удобно их применить. Начальный и конечный возраст разумно использовать как пределы для цикла. Не забудьте, что в текстовых полях содержатся строки, а не числа, и нужно применять функцию `Val()` для их преобразования.

Программа должна выводить возраст, сумму на счету по каждому году, сумму вкладов и прибыль, т. е. разность суммы на счету и суммы вкладов.

Для форматирования вывода используйте функцию `Format`, например:

```
VariableX = FormatCurrency(VariableX)
```

Подсказка 1. Добавляйте годовую сумму вклада, прежде чем вычислять проценты.

Подсказка 2. Сумма на счету с учетом прироста — это сумма на нем, умноженная на (1 + процент) (например, для 9% сумму надо умножить на 1,09).

10.6. Циклы в C# и J#

Как и во всех современных языках программирования, в C# и J# тоже есть циклы! В C# и J# есть аналоги цикла Visual Basic .NET `For…Next`, но код выглядит по-другому.

Цикл на языке C#. Вот как выглядит такой код в C#:

```
for (int Counter = 1; Counter < 4; Counter++)
{
MessageBox.Show(Counter.ToString());
}
```

Этот код выглядит не так, как оператор `For…Next`, но делает то же самое. В C# нет оператора `Next` и ключевое слово `for` пишется строчными буквами. Код, который должен выполняться в цикле, поме-

щается в фигурные скобки. Код в скобках после `for` управляет выполнением цикла.

Объявление переменной-счетчика цикла и инициализация ее начальным значением:

```
int Counter = 1;
```

Условное выражение, определяющее, когда завершится цикл:

```
Counter < 4;
```

Увеличение значения счетчика на 1:

```
Counter++
```

Это выражение можно записать как:

```
Counter = Counter + 1
```

Цикл на языке J#. Вот такой же код на J#. Он выглядит так же, за исключением того, что J# использует функцию `System.Convert.ToString` для преобразования значения `Counter` из целого числа в строку для вывода в окне сообщения.

```
for (int Counter = 1; Counter < 4; Counter++)
{
MessageBox.Show(System.Convert.ToString(Counter));
}
```

Хотя код выглядит не так, как оператор `For…Next` в Visual Basic .NET, он имеет тот же смысл — позволяет многократно исполнять один и тот же код.

Тест по теме «Циклы со счетчиком»

1. 

Какое слово не является ключевым в операторе цикла со счетчиком?

- ❍ `For`
- ❍ `Next`
- ❍ `Counter`
- ❍ `To`

2. 

Сколько раз будет выполнено тело цикла во вложенном цикле со счетчиком, если внешний цикл повторится 2 раза, а внутренний — 5 раз?

- ❍ *2*
- ❍ *5*
- ❍ *7*
- ❍ *10*

3. 

Преждевременный выход из цикла реализуется с помощью оператора

- ❍ `Exit For`
- ❍ `End If`
- ❍ `End Sub`
- ❍ `Exit`

4. 

Каким будет значение переменной `Counter` после завершения цикла со счетчиком `For Counter = 1 To 2`?

- ❍ *0*
- ❍ *1*
- ❍ *2*
- ❍ *3*

Глава 11

Циклы с условием

Microsoft В 2003 году была создана операционная система Windows XP для серверов, рабочих станций и персональных компьютеров. Операционная система обеспечивает информационную безопасность при работе в локальной сети и в Интернете (использует файловую систему NTFS, позволяет устанавливать политики безопасности для рабочих групп и доменов и пр.). Windows XP объединила достоинства пользовательских операционных систем Windows 95/98/Me и профессиональных Windows NT/2000.

11.1. Циклы Do While...Loop

Есть множество операций, которые нужно повторять, пока что-то не произойдет. Это «что-то» представляет собой условие прекращения процесса. Код в цикле с неопределенным количеством повторений выполняется неизвестное заранее число раз. Он перестает выполняться, когда выполняется (или перестает выполняться) какое-то условие. Есть два вида таких циклов: `do while` и `do until`. Цикл `do while` выполняется, пока выполняется условие. Цикл `do until` выполняется до тех пор, пока не выполнится условие. Используя циклы с неопределенным числом повторений, нужно уметь выбирать требуемый вид.

Цикл `Do While…Loop` выполняет блок кода раз за разом, пока выполняется заданное условие. Как только условие перестает выполняться, выполнение цикла завершается. Условие, управляющее циклом `Do While…Loop`, является его частью. Вот синтаксис цикла `Do While… Loop` в Visual Basic.NET:

```
Do While (Условие)
    Последовательность операторов
Loop
```

Обратите внимание на то, что `Do`, `While` и `Loop` — это ключевые слова. Они выделяются синим цветом. Условие заключено в скобки. Условием может быть любое булево выражение, результатом вычисления которого является `True` или `False`, например `X < 4`. Код, который вы хотите выполнить в цикле, помещается между операторами `Do While` и `Loop`. В начале каждого выполнения цикла проверяется условие. Если оно выполняется, то код в цикле выполняется. Если условие не выполняется, начинает выполняться код после цикла.

Теперь мы покажем вам пример цикла `Do While… Loop`. Создайте новое приложение Windows и назовите его «Do-While-Loop». Поместите на форму

Код между операторами `Do While` и `Loop` лучше выделять с помощью отступа. Если это сделать, будет легко понять, какой код будет выполняться в цикле. При этом упростится чтение и отладка кода.

кнопку. Выполните по кнопке двойной щелчок, чтобы отредактировать обработчик ее нажатия. Вставьте в обработчик следующий код:

```
Dim WhileValue As Integer
WhileValue = 0
Do While (WhileValue < 2)
    MessageBox.Show(WhileValue)
    WhileValue = WhileValue + 1
Loop
```

Постройте и запустите проект. Нажмите кнопку на форме. Появится окно сообщения с числом 0. Нажмите кнопку *OK*. Появится новое окно сообщения с числом 1. Нажмите кнопку *OK*. Вы вернулись к форме. Больше окон сообщений не появится.

Как работает этот код? Мы объявили переменную `WhileValue`. Изначально ей присвоено значение 0. При каждом выполнении цикла значение `WhileValue` увеличивается на 1. В начале цикла оператор `Do While` проверяет, выполняется ли условие `WhileValue < 2`. Если да, то выводится окно сообщения и значение `WhileValue` увеличивается. Если нет, выполнение цикла завершается. Цикл выполнится дважды, прежде чем условие перестанет выполняться. Первый раз в окне сообщения отобразится значение 0. Во второй раз — 1. При третьем выполнении оператора `WhileValue = 2`. Условие $2 < 2$ не выполняется. Цикл завершается, и окно сообщения не появляется.

Microsoft-CD

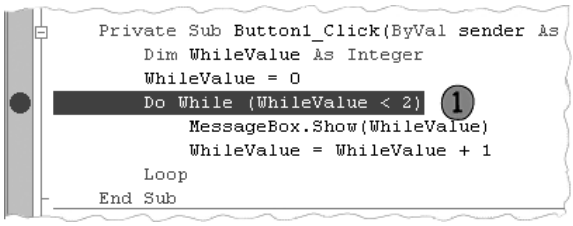
Задания для самостоятельного выполнения

11.1. В системе программирования Visual Basic .NET создать проект «Do-While-Loop», описанный в параграфе. Готовый проект содержится в самораспаковывающемся архиве Do_While_Loop.exe.

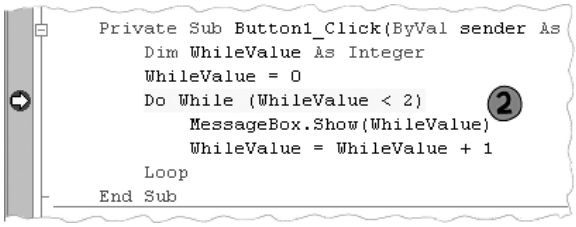
11.2. Пошаговое выполнение цикла Do While...Loop

Давайте воспользуемся отладочными инструментами Visual Studio .NET и отследим шаг за шагом выполнение цикла в программе «Do-While-Loop».

1. В окне *Код* (*Code*) установите точку останова на операторе `Do While`.



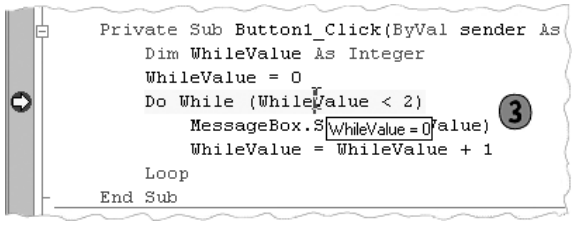
2. Запустите проект. Нажмите кнопку на форме. Код выполнится до точки останова и будет подсвечена строка с этой точкой.



3. Нажмите клавишу {F11}, чтобы выполнить подсвеченную строку кода.

   `Do While` `(WhileValue < 2)`

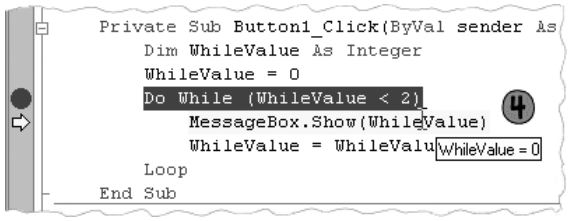
   Подведите курсор к переменной `WhileValue`. Ее значение равно 0.



4. Нажмите клавишу {F11}, чтобы выполнить подсвеченную строку кода.

```
MessageBox.Show(WhileValue)
```

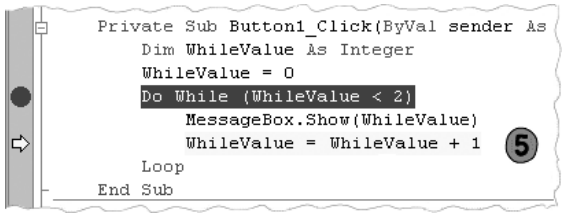
Появится окно сообщения с цифрой 0. Нажмите кнопку *OK*.



5. Нажмите клавишу {F11}, чтобы выполнить подсвеченную строку кода.

```
WhileValue = WhileValue + 1
```

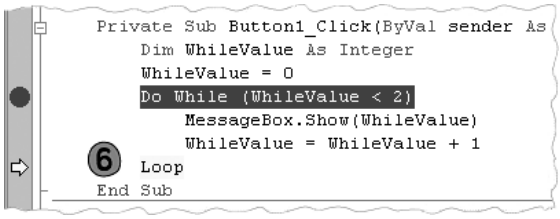
Значение `WhileValue` увеличится на 1.



6. Нажмите клавишу {F11}, чтобы выполнить подсвеченную строку кода.

```
Loop
```

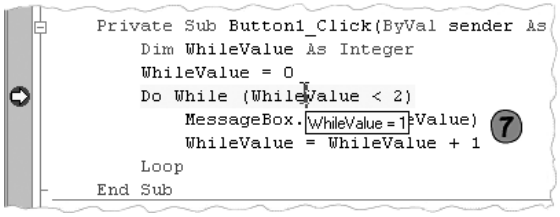
Цикл выполнится еще раз.



7. Нажмите клавишу {F11}, чтобы выполнить подсвеченную строку кода.

```
Do While (WhileValue < 2)
```

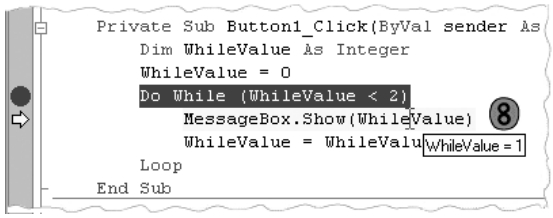
Подведите курсор к переменной `WhileValue`. Теперь ее значение равно 1.



8. Нажмите клавишу {F11}, чтобы выполнить подсвеченную строку кода.

```
MessageBox.Show(WhileValue)
```

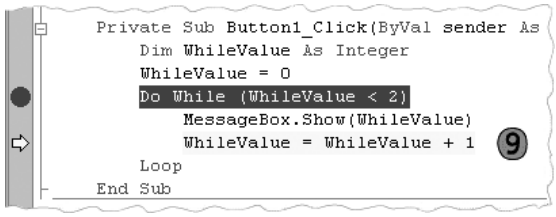
Появится окно сообщения с цифрой 1. Нажмите кнопку *OK*.



9. Нажмите клавишу {F11}, чтобы выполнить подсвеченную строку кода.

```
WhileValue = WhileValue + 1
```

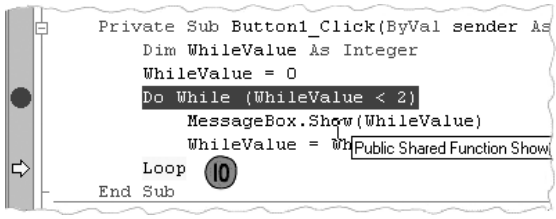
Значение WhileValue увеличится еще на 1.



10. Нажмите клавишу {F11}, чтобы выполнить подсвеченную строку кода.

Loop

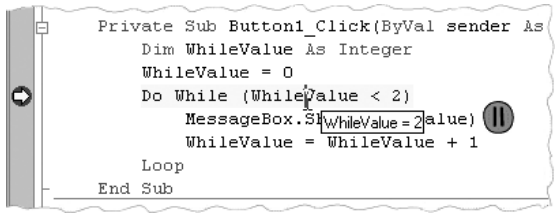
Цикл начнет выполняться снова.



11. Нажмите клавишу {F11}, чтобы выполнить подсвеченную строку кода.

Do While (WhileValue < 2)

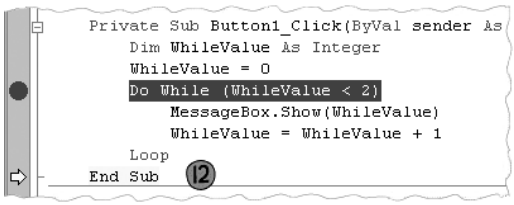
Подведите курсор к переменной WhileValue. Ее значение равно 2. Условие WhileValue < 2 перестало выполняться, поэтому выполнение цикла завершается. Окно сообщения больше не появляется, и значение WhileValue не увеличивается.



12. Нажмите клавишу {F11}, чтобы выполнить подсвеченную строку кода.

End Sub

Выполнение обработчика завершается. Снова отображается форма.



Теперь вы видели, как выполняется цикл `Do While…Loop`.

Microsoft-CD

Задания для самостоятельного выполнения

11.2. В системе программирования Visual Basic .NET создать проект «Do-While-Loop». Провести пошаговое выполнение проекта так, как это описано в параграфе. Готовый проект содержится в самораспаковывающемся архиве Do_While_Loop.exe.

11.3. Циклы Do Until...Loop

Цикл `Do Until…Loop` выполняет блок операторов кода, пока какое-то условие не станет истинным. Это условие может быть любым булевым выражением, результатом вычисления которого будет `True` или `False`. Вот синтакис цикла `Do Until…Loop` в Visual Basic .NET:

```
Do Until (Условие)
    Последовательность операторов
Loop
```

Слова Do, Until и Loop — ключевые. Они выделены синим цветом. Истинность условия проверяется в начале выполнения цикла. Если условие ложно, то выполняется код в цикле. Когда условие в операторе Do, Until становится истинным, цикл завершается.

Создайте новое приложение Windows и назовите его «Do-Until-Loop». Поместите кнопку на форму. Выполните двойной щелчок по этой кнопке, чтобы отредактировать обработчик ее нажатия. Добавьте в обработчик следующий код:

```
Dim UntilValue As Integer
UntilValue = 0
Do Until (UntilValue > 1)
    MessageBox.Show(UntilValue)
    UntilValue = UntilValue + 1
Loop
```

Постройте и запустите проект. Нажмите кнопку. Появится окно сообщения с цифрой 0. Нажмите *OK*. Появится окно сообщения с цифрой 1. Нажмите *OK*. Выполнение цикла закончится, и снова появится форма.

Как работает цикл Do Until…Loop? Сначала мы объявили переменную UntilValue и проинициализировали ее значением 0. Значение переменной UntilValue используется в условии, чтобы определить, должен ли цикл выполняться снова. При каждом выполнении цикла значение переменной UntilValue увеличивается на 1. В операторе Do Until содержится оператор сравнения, проверяющий, больше ли значение UntilValue, чем 1. При первой проверке значение UntilValue равно 0. Условие ложно, и выполняется код внутри цикла. Выводится окно сообщения, и значение UntilValue увеличивается на 1. При второй проверке значение UntilValue равно 1. Условие по-прежнему ложно, и код в цикле выполняется еще раз. Но при третьей проверке значение UntilValue равно 2. Условие ложно (2 > 1), и выполнение цикла заканчивается.

Microsoft-CD

Задания для самостоятельного выполнения

11.3. В системе программирования Visual Basic .NET создать проект «Do-Until-Loop», описанный в параграфе. Готовый проект содержится в самораспаковывающемся архиве Do_Until_Loop.exe.

11.4. Проекты с использованием Do...Loop

А теперь давайте разберем примеры с циклами `Do While…Loop` и `Do Until…Loop`, в которых используются булевы выражения, не зависящие напрямую от значения переменной-счетчика. Первый пример использует цикл `Do While…Loop` с булевым выражением, проверяющим, больше ли значение переменной нуля или равно ему. Если значение переменной станет меньше нуля, выполнение цикла завершится.

Создайте новое приложение Windows и назовите его «Do-While-Loop-2». Поместите на форму кнопку. Измените значение свойства `Text` этой кнопки на `Вычислить`. Выполните двойной щелчок по ней, чтобы отредактировать обработчик ее нажатия. Добавьте в него следующий код:

```
Dim Index As Integer = 0
Dim Controller As Integer = 167
Do While (Controller >= 0)
    Index = Index + 1
    Controller = Controller - (23 * Index)
Loop
MessageBox.Show(Controller)
```

Постройте и запустите проект «Do-While-Loop-2». Нажмите кнопку с надписью *Вычислить*. Появится окно сообщения с числом –63. Как этот код работает? Мы объявили две целочисленные переменные `Index` и `Controller` и задали их начальные значения. Значение переменной `Index` увеличивается

на 1 при каждом выполнении цикла. При каждом выполнении цикла для переменной Controller вычисляется новое значение — по более сложной формуле. Это значение вычисляется путем умножения значения Index на 23 и вычитания результата умножения из старого значения переменной Controller. При каждом выполнении цикла значение переменной Controller будет уменьшаться. Сложно навскидку сказать, сколько раз цикл выполнится, прежде чем значение переменной Controller станет меньше 0, и цикл завершится. Но это ничего не значит, поскольку для булева выражения важно значение только переменной Controller, а не переменной Index.

Во втором примере мы используем цикл Do While…Loop с булевым выражением, которое проверяет, выбран ли переключатель RadioButton3. При каждом выполнении цикла выбирается новый переключатель.

В проекте «Do-While-Loop-2» добавьте на форму еще одну кнопку. Измените значение свойства Text этой кнопки на While. Добавьте на форму три переключателя. Выполните двойной щелчок по новой кнопке, чтобы отредактировать обработчик ее нажатия. Добавьте в обработчик такой код:

```
Dim Index As Integer = 0
RadioButton1.Checked = True
Do While (RadioButton3.Checked = False)
    Index = Index + 1
    If Index = 1 Then
      RadioButton1.Checked = True
    End If
    If Index = 2 Then
      RadioButton2.Checked = True
    End If
    If Index = 3 Then
      RadioButton3.Checked = True
    End If
Loop
MessageBox.Show(Index)
```

Постройте и запустите приложение. Обратите внимание на то, что выбрана позиция переключателя RadioButton1. Нажмите кнопку с надписью *While*. Код выберет позицию переключателя RadioButton3, и в окне сообщения будет выведено значение переменной Index. Если у вас очень хорошая реакция или очень старый компьютер, вы можете заметить, как на мгновение была выбрана позиция переключателя RadioButton2.

Вот что происходит. Мы объявляем переменную Index, чтобы отслеживать, сколько раз выполнится код в цикле. Однако эта переменная не используется в булевом выражении, управляющем циклом Do While…Loop. Вместо этого Index используется для перебора переключателей. Булево выражение в цикле Do While…Loop проверяет, выбран ли переключатель RadioButton3. Как только условие RadioButton3.Checked = True выполняется, цикл завершается.

В третьем примере мы используем оператор Do Until…Loop. Булево выражение сравнивает значение свойства Text текстового поля со значением строковой переменной, которое изменяется с каждой итерацией цикла. Добавьте текстовое поле на форму проекта «Do-Until-Loop-2». Оставьте значение TextBox1 свойства Text текстового поля. Добавьте на форму кнопку и измените значение свойства Text этой кнопки на Сравнить тексты. Выполните двойной щелчок по новой кнопке, чтобы отредактировать обработчик ее нажатия. Добавьте в обработчик следующий код:

```
Dim MatchText As String = ""
Dim Index As Integer = 0
Do Until (MatchText = TextBox1.Text)
    Index = Index + 1
    If Index = 2 Then
      MatchText = "Text"
    End If
    If Index = 3 Then
      MatchText = "Box1"
```

```
    End If
    If Index = 4 Then
      MatchText = "TextBox1"
    End If
Loop
MessageBox.Show("При значении счетчика " &_
Index & " тексты равны")
```

Постройте и запустите приложение. Нажмите кнопку с надписью *Сравнить тексты*. В появившемся окне сообщения будет показано значение переменной `Index`, при котором тексты совпали.

В коде используется оператор `Do Until…Loop`. Переменная `Index` используется только для определения значения переменной `MatchText`. Булево выражение сравнивает значение переменной `MatchText` и значение свойства `Text` текстового поля `TextBox1`. При четвертом выполнении кода в цикле значение `Index` будет равно 4, и `MatchText = "TextBox1"`. Булево выражение станет истинным, поскольку `"TextBox1" = "TextBox1"`, и выполнение цикла завершится.

А что, если свойству `Text` текстового поля `TextBox1` присвоить значение `Текст`? Сможете ли вы добавить в код еще один оператор `If..Then`, чтобы код все равно работал?

Microsoft-CD

Задания для самостоятельного выполнения

11.4. В системе программирования Visual Basic .NET создать проект «Do-While-Loop-2», описанный в параграфе. Готовый проект содержится в самораспаковывающемся архиве Do_While_Loop_2.exe.

11.5. Циклы с постусловием

В предыдущих параграфах циклы `Do While…Loop` и `Do Until…Loop` использовались с предусловием,

т. е. условие располагалось в начале цикла. Однако можно использовать циклы `Do While…Loop` и `Do…Loop Until` и с постусловием, т. е. чтобы условие располагалось в конце цикла. В таких циклах операторы внутри циклов будут всегда выполняться хотя бы один раз. Потом будет проверяться выполнение условия. Вот пример оператора `Do…Loop While` с постусловием:

```
Dim WhileValue As Integer
WhileValue = 0
Do
    MessageBox.Show(WhileValue)
    WhileValue = WhileValue + 1
Loop While (WhileValue < 2)
```

А вот пример оператора `Do…Loop Until`:

```
Dim UntilValue As Integer
UntilValue = 0
Do
    MessageBox.Show(UntilValue)
    UntilValue = UntilValue + 1
Loop Until (UntilValue > 1)
```

Операторы от `Do` и до `Loop` выполнятся как минимум один раз. Затем будет проверено выполнение условия, чтобы определить, должен ли цикл выполняться еще раз.

Задания для самостоятельного выполнения

Microsoft-CD

11.5. В системе программирования Visual Basic .NET создать проект «Циклы с постусловием», описанный в параграфе. Готовый проект содержится в самораспаковывающемся архиве Циклы_с_постусловием.exe.

11.6. Циклы в J# и C#

Давайте посмотрим на циклы с неопределенным количеством повторений в C# и J#.

Циклы в языке C#. На C# можно написать код, делающий то же, что и операторы `Do While…Loop` и `Do…Loop While` в Visual Basic .NET. В C# нет эквивалентов операторам `Do Until…Loop` или `Do…Loop Until`.

Вот как написать такой цикл (`while` и `loop` записываются строчными буквами) на C#. Здесь условие проверяется в начале выполнения цикла:

```
int WhileValue=0;
While (WhileValue<5)
{
    MessageBox.Show(WhileValue.ToString());
    WhileValue=WhileValue+1;
}
```

А вот код на C# для цикла `do…while loop` (`do`, `while` и `loop` записываются строчными буквами), истинность условия в котором проверяется после выполнения кода в цикле:

```
int WhileValue=0;
do
{
    MessageBox.Show(WhileValue.ToString());
    WhileValue=WhileValue+1;
}
while(WhileValue<5);
```

Циклы в языке J#. А теперь посмотрим на код на J#, который очень похож на код на C#. В J# нет эквивалентов операторам `Do Until…Loop` или `Do…Loop Until` из Visual Basic .NET. Вот как создается цикл (`while` и `loop` записываются строчными буквами) на J#. Здесь цикл проверяет истинность условия в начале выполнения:

```
int WhileValue=0;
while (WhileValue<5)
{
    MessageBox.Show(System.Convert.ToString
    (WhileValue));
    WhileValue=WhileValue+1;
}
```

Вот код на J# для цикла (do, while и loop записываются строчными буквами), проверяющего истинность условия после выполнения кода в цикле:

```
int WhileValue=0;
do
{
    MessageBox.Show(System.Convert.ToString
    (WhileValue));
    WhileValue=WhileValue+1;
}
while (WhileValue<5);
```

11.7. Выход из циклов

Из циклов Do While…Loop, Do Until…Loop, Do…Loop While и Do…Loop Until можно выйти с использованием оператора Exit Do. С помощью оператора If…Then проверяется истинность условия, управляющего выполнением оператора Exit Do. Если условие истинно, то выполняется оператор Exit Do и выполнение цикла завершается. Используйте Exit Do, если вы хотите завершить выполнение цикла до того, как его управляющее условие станет истинным (или ложным).

Давайте разберем пример выхода из цикла Do Until…Loop. Создайте новое приложение Windows и назовите его «Exit-Do». Поместите на форму кнопку. Выполните двойной щелчок по кнопке, чтобы отредактировать обработчик ее нажатия. Добавьте в обработчик следующий код:

```
Dim UntilValue As Integer
UntilValue = 0
Do Until (UntilValue > 10)
    MessageBox.Show(UntilValue)
    UntilValue = UntilValue + 1
    If UntilValue = 5 Then
      Exit Do
    End If
Loop
MessageBox.Show("Exit Value= " & UntilValue)
```

Постройте и запустите проект. Нажмите кнопку на форме. При каждом выполнении цикла значение переменной `UntilValue` будет увеличиваться на 1 и эти значения будут выводиться в окнах сообщений. Закрывайте каждое окно сообщения нажатием кнопки *OK*. Когда значение `UntilValue` достигнет 5, условие в операторе `If…Then` станет истинным. Будет выполнен оператор `Exit Do`. Выполнение цикла завершится, и начнет выполняться код за оператором `Do Until…Loop`. Этот код выведет на экран окно сообщения с текстом «Exit Value= 5». Нажмите в этом окне кнопку *OK*, чтобы вернуться на форму.

Точно такой же оператор `Exit Do` можно использовать для выхода из циклов `Do While…Loop` до того, как управляющее условие станет ложным.

Задания для самостоятельного выполнения

Microsoft-CD

11.6. В системе программирования Visual Basic .NET создать проект «Exit-Do», описанный в параграфе. Готовый проект содержится в самораспаковывающемся архиве Exit_Do.exe.

11.7. В системе программирования Visual Basic .NET создать проект «Мишень», в котором компьютер пытается попасть в мишень. Заготовка проекта хранится в самораспаковывающемся архиве Мишень.exe. Форма должна выглядеть примерно так:

В заготовке проекта содержится код для обработчика нажатий кнопки с надписью *New Game*, потому что в нем используются графические возможности, не рассматривающиеся в этом курсе. Вы должны написать код для обработчика нажатий кнопки с надписью *Play*.

Цель игры — подсчитать количество попыток, необходимых для попадания в маленькую точку в центре мишени. Размер этой точки — 1 пиксель. Поскольку размер рисунка 200 × 200 пикселей, координаты точки в центре — 100, 100.

Сгенерируйте случайное целое значение для переменной `X`, используемой для измерения горизонтальной координаты. Сгенерируйте случайное целое значение для переменной `Y`, используемой для измерения вертикальной координаты. Для генерации случайного числа в диапазоне 0–200 используется такой код:

```
Dim MyRandomGenerator As System.Random
MyRandomGenerator = New System.Random
Dim RanNum As Integer

' Генерируем случайное число от 0 до 200 - не
'включая 200.
RanNum = MyRandomGenerator.Next(0, 200)
```

Вы умеете использовать циклы с неопределенным количеством повторений, и здесь очень удобно применить такой цикл. Его выполнение закончится, когда X и Y одновременно станут равны 100.

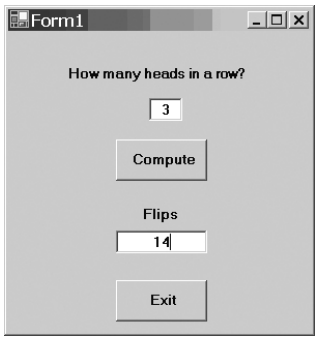
Считайте количество выполнений цикла. Результат выводите в текстовое поле `TextBox1`.

Вот код для отображения попадания на мишени:

```
g.DrawEllipse(MyPen, New Rectangle(X, Y, 1, 1))
```

11.8. В системе программирования Visual Basic .NET создать проект «Монета», имитирующий подбрасывание монеты. Вы когда-нибудь интересовались, сколько раз нужно подбросить монету, чтобы три раза подряд выпала решка? А четыре раза? А пять? Заго-

товка формы проекта хранится в самораспаковывающемся архиве Монета.exe. Форма должна выглядеть примерно так:



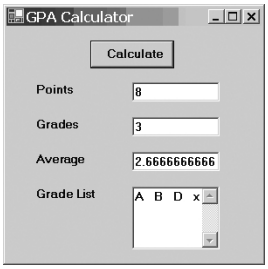
Код для генерации случайно выпадающих 0 или 1 такой:

```
Dim MyRandomGenerator As System.Random
MyRandomGenerator = New System.Random
Dim RanNum As Integer
' Генерируем случайное значение между 0 и 2 -
'не включая 2.
RanNum = MyRandomGenerator.Next(0, 2)
```

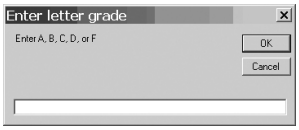
Предположим, что 1 — это решка, а 0 — орел. Продолжайте генерировать случайные числа, пока не получите столько единиц подряд, сколько указано в текстовом поле TextBox1. Если выпадает 0, начинайте сначала. Поместите количество сделанных попыток в TextBox2.

При тестировании лучше добавить в цикл оператор, который прекратит выполнение, если количество сделанных попыток станет больше какого-то очень большого числа.

11.9. В системе программирования Visual Basic .NET создать проект «Средняя оценка», вычисляющий среднюю оценку для последовательности оценок. Заготовка формы проекта хранится в самораспаковывающемся архиве Средняя_оценка.exe. Форма должна выглядеть примерно так:



Эта программа использует функцию `InputBox()`, которая выводит показанное ниже окно.



Это новая для вас возможность, поэтому ниже приведен код, который вам понадобится. Объявите переменную, в которой будет храниться введенная оценка:

```
Dim GradeIn As String = "None Entered"
```

Затем внутри цикла с неопределенным количеством повторений нужно поместить такой код:

```
GradeIn = InputBox("Enter A, B, C, D, or F",_
"Enter letter grade", "")
```

Пользователь сможет вводить столько оценок, сколько ему нужно. Когда он захочет закончить ввод, он должен либо не вводить никакой оценки, либо нажать кнопку с надписью *Cancel*. В любом из этих случаев переменная `GradeIn` будет пустой. Пустая строка обозначается двумя кавычками подряд, без пробела между ними.

Оценка A — это 4 балла, B — 3, C — 2, D — 1, а F — 0. Игнорируйте любые другие буквы или цифры. Пользователь может вводить как строчные, так и заглавные буквы.

Все введенные оценки должны отображаться в текстовом поле внизу окна для проверки.

Когда все оценки будут введены, вычислите среднюю оценку.

Тест по теме «Циклы с условием»

1. 

До каких пор продолжается выполнение цикла `Do…While Loop`?

- ❍ *Пока условие не станет истинным*
- ❍ *Пока условие не станет ложным*
- ❍ *Пока значение счетчика не достигнет максимального значения*
- ❍ *Бесконечно*

2. 

До каких пор продолжается выполнение цикла `Do…Until Loop`?

- ❍ *Пока условие не станет истинным*
- ❍ *Пока условие не станет ложным*
- ❍ *Пока значение счетчика не достигнет максимального значения*
- ❍ *Бесконечно*

3. 

Преждевременный выход из цикла реализуется с помощью оператора

- ❍ `Exit For`
- ❍ `End Do`
- ❍ `Exit Do`
- ❍ `Stop Do`

4. 

Где в цикле `Do…While Loop` размещается условие?

- ❍ *После ключевого слова `Do`*
- ❍ *После ключевого слова `While`*
- ❍ *После ключевого слова `Loop`*
- ❍ *Внутри тела цикла*

Глава 12

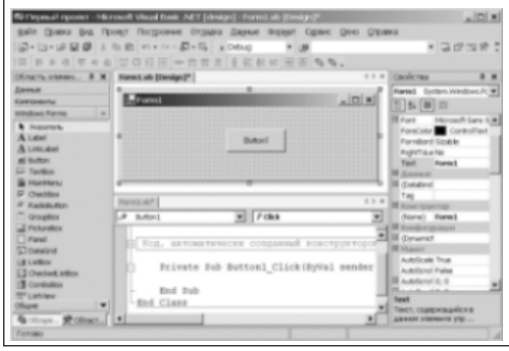
Подпрограммы и функции

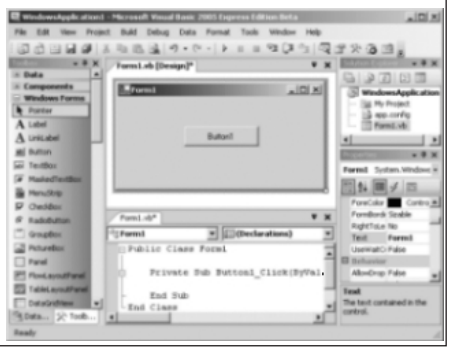
Microsoft

В 2003 году была создана система объектно-ориентированного программирования Visual Basic .NET 2003, а в 2005 году — система объектно-ориентированного программирования Visual Basic 2005 Express Edition. Visual Basic .NET является системой визуального объектно-ориентированного программирования на платформе .NET Framework — новой компьютерной платформе разработки приложений в операционной системе Windows.

Платформа .NET Framework предоставляет возможность создавать приложения на различных языках объектно-ориентированного программирования: Visual Basic .NET, J# (читается «Джей-шарп»), созданном на основе языка Java, C# (читается «Си-шарп»), созданном на основе языков C и C++.

Основным компонентом .NET Framework является общеязыковая среда выполнения программ (CLR — Common Language Runtime), которая обеспечивает компиляцию программ в два этапа. Сначала программа, написанная на объектно-ориентированном языке, компилируется в промежуточный код на языке Microsoft Intermediate Language, а затем — в машинный код.





12.1. Что такое подпрограммы?

Когда вы пишете программу, разделите ее на независимые части, выполняющие определенные задачи в приложении, и оформите каждую часть в виде подпрограммы. В вашем приложении может быть столько подпрограмм, сколько вам нужно. Незачем жадничать, создавая подпрограммы, — создайте столько, сколько хотите, но старайтесь писать подпрограммы, выполняющие четко определенные задачи.

Зачем нужны подпрограммы? Их использование делает код более понятным, его становится легче отлаживать, поскольку он делится на мало зависящие друг от друга части. Намного легче написать и отладить несколько небольших программ, чем одну большую программу. После отладки подпрограммы она готова к использованию и будет работать всегда, когда она вам понадобится.

Подпрограммы помогают сэкономить много времени, если вы пишете сложный код, к которому нужно обращаться из разных мест в программе. Их можно использовать много раз. Написав подпрограмму, вы можете использовать ее (обращаться к ней) из любого места программы.

Из подпрограмм можно вызывать другие подпрограммы.

Подпрограммы также позволяют разделить написание большой программы между несколькими программистами. Каждому программисту нужно будет написать определенный набор подпрограмм для приложения. Он будет отвечать за написание и отладку этих подпрограмм и за то, чтобы они правильно взаимодействовали с другими частями программы.

Подпрограммы часто используются для выполнения вычислений, форматирования и вывода информации, настройки пользовательского интерфейса, вывода приглашений к вводу данных и выполнения ввода/вывода.

12.2. Создание и вызов подпрограмм

Создание собственных подпрограмм для приложений — это еще одна часть программирования. Это немного похоже на создание форм — нужно сначала представить себе, что должна делать подпрограмма, а потом написать соответствующий код. Подпрограммы похожи на маленькие программы, и в них могут использоваться любые операторы, которые вы изучили раньше!

Создание подпрограммы. Сейчас мы напишем подпрограмму и рассмотрим, как использовать ее в приложении. Мы начнем с изучения синтаксиса подпрограмм, а потом познакомимся с тем, как создавать код для нее. Вот синтаксис подпрограммы:

```
Private Sub ИмяПодпрограммы()
    Последовательность операторов
End Sub
```

Обратите внимание на то, что `Private`, `Sub` и `End Sub` — это ключевые слова. Они выделены синим цветом. Ключевое слово `Private` означает, что к этой подпрограмме можно обращаться только из кода в данной форме. Подпрограмму можно назвать как угодно, но лучше выбрать понятное имя, чтобы можно было разобраться, для чего нужна эта подпрограмма. Обратите внимание на пару скобок после имени подпрограммы. Если подпрограмме нужно передать какую-то информацию, то переменные, в которых хранится эта информация, указываются между скобками. Вы уже видели это в обработчиках событий в предыдущих параграфах. Операторы кода подпрограммы помещаются между строками с операторами `Sub` и `End Sub`. Операторы выполняются по порядку. В подпрограммах можно использовать почти любые операторы.

Код в подпрограмме лучше выделить отступом, чтобы в дальнейшем было легче понять, какой код принадлежит к подпрограмме.

А теперь мы напишем подпрограмму и посмотрим, как вызвать ее из кода. Создайте новое приложение Windows и назовите его «Подпрограмма». Откройте окно редактора кода. Найдите строку

```
Код, автоматически созданный конструктором форм Windows
```

В следующую за ней строку добавьте такой код:

```
Private Sub MyMessage()
    MessageBox.Show("Это сообщение выведено_
    подпрограммой Sub MyMessage.")
End Sub
```

Как видите, при выполнении кода в подпрограмме `MyMessage` будет выведено окно сообщения с текстом «Это сообщение выведено подпрограммой Sub MyMessage.» Но как сделать так, чтобы этот код выполнился?

Вызов подпрограммы. Чтобы выполнить код внутри подпрограммы, нужно вызвать эту подпрограмму в коде программы. Когда подпрограмму вызывают, выполняется код в этой подпрограмме. Синтаксис вызова подпрограммы прост. Нужно просто указать имя подпрограммы и пару скобок за ним. Вот синтаксис вызова подпрограммы:

```
SubName()
```

А теперь давайте вызовем подпрограмму, которую вы только что написали.

В проекте «Подпрограмма» поместите на форму `Form1` кнопку. Измените значение свойства `Text` этой кнопки на `Сообщение`. Выполните двойной щелчок по кнопке, чтобы отредактировать обработчик ее нажатия. Добавьте в обработчик следующую строку кода:

```
MyMessage()
```

Постройте и запустите проект. Нажмите кнопку с надписью *Сообщение*. Появится окно сообщения с текстом «Это сообщение выведено подпрограммой Sub MyMessage.» Как оно появилось? Его отобразила строка кода из подпрограммы MyMessage. Когда эта подпрограмма была вызвана из обработчика нажатия кнопки, был выполнен код в подпрограмме.

А теперь давайте добавим в проект «Подпрограмма» еще одну подпрограмму. Она будет выводить другое сообщение. Откройте окно редактора кода и добавьте после оператора End Sub подпрограммы MyMessage такой код:

```
Private Sub YourMessage()
    MessageBox.Show("Это сообщение выведено_
    подпрограммой Sub YourMessage."
End Sub
```

На этот раз давайте вызовем подпрограмму YourMessage из MyMessage, а не из обработчика нажатия на кнопку. Отредактируйте код в подпрограмме MyMessage, чтобы он выглядел так:

```
Private Sub MyMessage()
    MessageBox.Show("Это сообщение выведено_
    подпрограммой Sub MyMessage.")
    YourMessage()
End Sub
```

Мы добавили вызов подпрограммы YourMessage в подпрограмму MyMessage. Постройте и запустите проект. Нажмите кнопку с надписью *Сообщение*. Появятся два окна сообщений. В первом будет текст «Это сообщение выведено подпрограммой Sub MyMessage.» Во втором будет текст «Это сообщение выведено подпрограммой Sub YourMessage.» Как работает этот код? Обработчик нажатия на кнопку вызывает подпрограмму MyMessage. Когда эта подпрограмма вызывается, выполняется код, содержащийся в ней. Подпрограмма MyMessage содержит две строки кода. В первой строке содержится оператор вывода окна

сообщения с текстом «Это сообщение выведено подпрограммой Sub MyMessage.» Во второй строке вызывается подпрограмма `YourMessage`, которая выводит окно сообщения с текстом «Это сообщение выведено подпрограммой `Sub YourMessage`.»

Вы создали две подпрограммы и вызвали одну из них из обработчика нажатия кнопки. А вторая подпрограмма вызывалась из первой!

Microsoft-CD

Задания для самостоятельного выполнения

12.1. В системе программирования Visual Basic .NET создать проект «Подпрограмма», описанный в параграфе. Готовый проект содержится в самораспаковывающемся архиве Подпрограмма.exe.

12.3. Подпрограммы с аргументами

Одна из ценных возможностей Visual Basic .NET и большинства остальных современных языков программирования — возможность передавать подпрограммам нужную информацию. Это очень удобно, потому что можно создавать подпрограммы, которые выдают разные результаты и выполняют разные действия в зависимости от того, какую информацию им передают. Фрагменты информации, которые передаются подпрограмме, называются аргументами. Когда вы пишете подпрограмму, которой хотите передавать аргументы, нужно определить, сколько будет этих аргументов и каких они будут типов.

Вот синтаксис написания подпрограммы с аргументами:

```
Private Sub SubName(ByVal ИмяАргумента1 As
ТипАргумента1, ByVal ИмяАргумента2 As
ТипАргумента2, ... ByVal + ИмяАргументаN As
ТипАргументаN)
    Последовательность операторов
End Sub
```

Здесь скобки не пустые. В них содержится список имен передаваемых подпрограмме аргументов и типов этих аргументов. Обратите внимание на то, что `ByVal` — ключевое слово. Оно выделено синим цветом. Типы аргументов — это обычные типы, например `Integer`, `String` и т. д. Обратите внимание, что аргументы отделяются друг от друга запятыми.

Скорее всего, реальный пример вы поймете гораздо быстрее, чем объяснения. Откройте окно редактора кода для проекта «Подпрограмма». После оператора `End Sub` подпрограммы `YourMessage` добавьте новую подпрограмму:

```
Private Sub GeneralMessage(ByVal InMessage As_
String)
    MessageBox.Show(InMessage)
End Sub
```

Этой подпрограмме нужно передавать аргумент типа `String`. Когда вызывается эта подпрограмма, она отображает окно сообщения с текстом, переданным ей в аргументе.

Давайте вызовем подпрограмму `GeneralMessage` и передадим ей аргумент типа `String`. Добавьте на форму проекта «Подпрограмма» вторую кнопку. Измените значение свойства `Text` этой кнопки на `Любое сообщение`. Выполните двойной щелчок по этой кнопке, чтобы отредактировать обработчик ее нажатия. Добавьте в него такой код:

```
GeneralMessage("Whatever message.")
GeneralMessage("Some other message.")
GeneralMessage("A different message.")
```

Постройте и запустите проект. Нажмите кнопку с надписью *Любое сообщение*. Будут выведены три окна сообщений с разными текстами. Как это получилось? Код в обработчике нажатия кнопки вызывает подпрограмму `GeneralMessage` три раза. При каждом вызове он передает ей в качестве аргумента новую строку. При первом вызове это строка

Переменные, объявленные в подпрограмме или функции с помощью оператора `Dim`, называются локальными переменными. Когда переменная объявляется в подпрограмме или функции, она обладает теми же свойствами, что и объявленная внутри обработчика, потому что обработчик тоже является подпрограммой. К этим переменным можно обращаться только во время выполнения подпрограммы или функции. По завершении их выполнения переменные теряют хранящиеся в них значения, и к ним нельзя больше обращаться.

«Whatever message». При втором вызове это строка «Some other message.» При третьем — «A different message». Когда выполняется код подпрограммы, аргумент передается этой подпрограмме и используется в операторе `MessageBox.Show`.

Давайте разберем еще один пример. На этот раз мы напишем подпрограмму, которая принимает в качестве аргументов два целых числа, складывает их и выводит результат сложения в окне сообщения. Добавьте новую подпрограмму `Adders` в проект «Подпрограмма». Эта подпрограмма будет выглядеть так:

```
Private Sub Adders(ByVal AddOne As Integer,
ByVal AddTwo As Integer)
Dim Total As Integer
    Total = AddOne + AddTwo
    MessageBox.Show(Total)
End Sub
```

Обратите внимание на то, что при вызове этой подпрограммы нужно передать ей два аргумента.

Добавьте на форму проекта «Подпрограмма» третью кнопку. Измените значение ее свойства `Text` на `Сложить числа`. Выполните двойной щелчок по новой кнопке, чтобы отредактировать обработчик ее нажатия. Добавьте в обработчик одну строку:

```
Adders(34, 57)
```

Постройте и запустите приложение. Нажмите кнопку с надписью *Сложить числа*. Появится окно сообщения с числом 91. Наш обработчик нажатия кнопки вызывает подпрограмму `Adders` и передает ей в качестве аргументов два целых числа, 34 и 57. Подпрограмма `Adders` выполняет три строки кода. Сначала она объявляет переменную `Total`. Затем она присваивает переменной `Total` значение суммы `AddOne` (34) и `AddTwo` (57). В аргументах `AddOne` и `AddTwo` находятся значения, переданные подпрог-

рамме при вызове. И наконец, подпрограмма выводит значение переменной `Total` **(91)** в окне сообщений.

Теперь вы можете писать собственные подпрограммы, вызывать их и передавать им информацию. Дальше вы узнаете, как писать функции. Функциям можно передавать информацию точно так же, как и подпрограммам, — с помощью аргументов. Однако в отличие от подпрограмм, функции могут возвращать какую-то информацию программе.

Microsoft-CD

Задания для самостоятельного выполнения

12.2. В системе программирования Visual Basic .NET создать проект «Подпрограмма» так, как это описано в параграфе. Готовый проект содержится в самораспаковывающемся архиве Подпрограмма.exe.

12.4. Создание и вызов собственных функций

Создание собственных функций. Главное различие между подпрограммами и функциями состоит в том, что функция возвращает значение, а подпрограмма — нет. Функция возвращает значение определенного типа. Когда создается функция, обязательно должен быть определен тип возвращаемого значения. Ниже приведен синтаксис создания функции, которая имеет определенный набор аргументов и возвращаемое значение:

```
Private Function ИмяФункции
(ByVal ИмяАгрумента1 As ТипАргумента1,
ByVal ИмяАгрумента2 As ТипАргумента2, ...
ByVal ИмяАргументаN As ТипАргументаN) As
ТипВозвращаемогоЗначения
    Последовательность операторов
    ИмяФункции = ВозвращаемоеЗначение
End Function
```

Обратите внимание, что слова Function и End Function являются ключевыми словами и выделяются синим цветом. Название функции должно быть содержательное. Список имен аргументов и типов аргументов расположен в круглых скобках после названия функции. Имена аргументов отделяются друг от друга запятой. Функция возвращает только одно значение. Вы должны определить тип возвращаемого функцией значения (например, как As Integer или As String) после списка аргументов.

Понять синтаксис функций поможет пример. Создайте новое приложение Windows и назовите его «Функция». Откройте окно редактора кода и найдите строку

```
Код, автоматически созданный конструктором форм
Windows
```

Сразу после нее вставьте такой код:

```
Private Function Multipliers(ByVal MultOne_
As Integer, ByVal MultTwo As Integer)_
As Integer
    Multipliers = MultOne * MultTwo
End Function
```

Эта функция называется Multipliers и принимает два целочисленных параметра — MultOne и MultTwo. Тип возвращаемого функцией значения указывается после ее списка аргументов. В данном случае это целое число (тип Integer). Код в функции состоит из единственной строки:

```
Multipliers = MultOne * MultTwo
```

Эта строка кода перемножает два аргумента и задает возвращаемое значение. Имя функции, Multipliers, используется как имя возвращаемого значения. Если этого не сделать, возникнет ошибка.

А теперь давайте вызовем эту функцию и используем ее возвращаемое значение в программе.

Вызов собственных функций. Один из способов вызова функции — поставить ее имя со списком аргументов справа от знака равенства в операторе присваивания. При этом тип возвращаемого функцией значения должен быть таким же, что и у переменной, стоящей слева от знака равенства.

Синтаксис вызова функции:

```
ПеременнаяДляВозвращаемогоЗначения =
ИмяФункции(Аргумент1, Аргумент2, ... АргументN)
```

Давайте вызовем функцию `Multipliers`, которую мы только что создали, и используем возвращаемое ею значение в коде. Поместите кнопку на форму `Form1` проекта «Функция». Измените значение свойства `Text` этой кнопки на `Умножить`. В обработчик ее нажатия поместите такой код:

```
Dim Product As Integer
Product = Multipliers(34, 57)
MessageBox.Show(Product)
```

Постройте и запустите проект. Нажмите кнопку с надписью *Умножить*. Появится окно сообщения с результатом перемножения 34 и 57 (1938). Как работает этот код? В обработчике нажатия кнопки объявляется переменная `Product` типа `Integer`. Код вызывает функцию `Multipliers` и передает ей аргументы 34 и 57. Функция `Multipliers` перемножает значения `MultOne` (34) и `MultTwo` (57) и помещает результат перемножения `MultOne * MultTwo` в переменную `Multipliers`, хранящую возвращаемое значение. Возвращаемое значение имеет тип `Integer`. В обработчике нажатия кнопки переменной `Product` типа `Integer` присваивается значение, возвращенное функцией `Multipliers`. Это значение и выводится в окне сообщения.

Давайте попробуем кое-что еще, раз у нас есть функция `Multipliers`. Добавьте на форму вторую

Переменные, объявленные в функции с помощью оператора `Dim`, будут локальными. Они доступны только в функции, пока выполняется ее код. Когда функция заканчивает выполняться, локальные переменные становятся недоступными.

кнопку. Измените значение свойства `Text` этой кнопки на `Умножить несколько раз`. В обработчик нажатия кнопки с надписью *Умножить несколько раз* добавьте такой код:

```
Dim Product As Integer
Product = Multipliers(Multipliers(2, 3),_
Multipliers(5, 7))
MessageBox.Show(Product)
```

Постройте и запустите проект. Нажмите кнопку с надписью *Умножить несколько раз*. Будет выведено число 210. На этот раз мы вызываем функцию `Multipliers` трижды. Сначала мы вызываем ее с аргументами 2 и 3. Возвращаемое ею значение (6) используется как первый аргумент при третьем вызове функции `Multipliers`. Во второй раз функция `Multipliers` вызывается с аргументами 5 и 7. Возвращаемое значение (35) используется как второй аргумент при третьем вызове функции `Multipliers`. При третьем вызове функции `Multipliers` ей передаются аргументы 6 (возвращаемое значение при первом вызове `Multipliers`) и 35 (возвращаемое значение при втором вызове `Multipliers`).

Видите, как можно вызвать функцию и использовать возвращаемое значение при вызове другой функции?

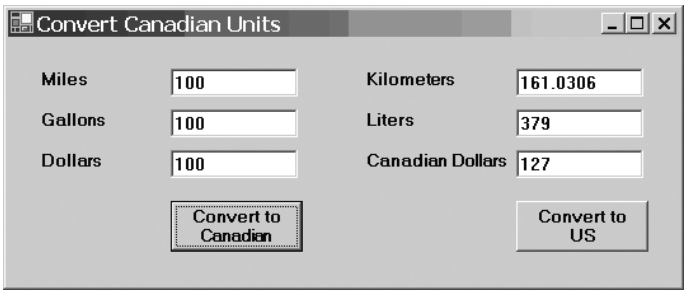
Microsoft-CD

Задания для самостоятельного выполнения

12.3. В системе программирования Visual Basic .NET создать проект «Функция», описанный в параграфе. Готовый проект содержится в самораспаковывающемся архиве Функция.exe.

12.4. В системе программирования Visual Basic .NET создать проект «Единицы измерения», позволяющий осуществлять перевод из американских единиц (длина, объем и денежная единица) в канадские и обратно. Заготовка проекта хранится в самораспако-

вывающемся архиве Единицы_измерения.exe. Форма должна выглядеть примерно так:



При нажатии кнопки с надписью *Convert to Canadian* выполняйте преобразование значений в левых текстовых полях в канадские единицы. При нажатии кнопки с надписью *Convert to US* выполняйте обратное преобразование.

Используйте такие коэффициенты пересчета:
1 миля = 0,621 километров;
1 галлон = 3,79 литров;
1 доллар = 1,27 канадских долларов.

Вы уже знаете, как создавать собственные функции и подпрограммы. В этой программе вам придется это делать. В заготовке проекта введена одна из 6 функций, напишите оставшиеся 5.

12.5. Встроенные функции

В платформе .NET есть огромное количество встроенных подпрограмм и функций для выполнения часто встречающихся задач. Эти функции тщательно отлажены и проверены, и они доступны для всех языков .NET. Если это возможно, используйте встроенные процедуры и функции, а не пишите свои собственные. Использование встроенных процедур и функций не только экономит ваше время, но и делает код более понятным для других людей.

Кроме того, замену многим встроенным процедурам и функциям вам написать не удастся. Эти процедуры обращаются к частям платформы .NET и классам Windows API, к которым вы обратиться не сможете.

Рассмотрим несколько часто используемых процедур и функций, встроенных в платформу .NET, к которым можно обратиться из программ на Visual Basic .NET. Возможно, они покажутся вам знакомыми — они использовались в заданиях в этом курсе. Мы использовали их, но не объясняли, а сейчас мы их объясним!

Некоторые функции для работы со строками. В платформе .NET есть множество функций, выполняющих разные действия со строками. Эти функции позволяют удалять символы, извлекать части строк, заменять строчные буквы на заглавные и т. д. Назначение большинства этих функций легко понять из примеров.

У всех функций работы со строками, которые мы будем рассматривать, синтаксис один и тот же. Их можно вызывать для любой строки. Некоторым из них нужны аргументы, некоторым нет. Большинство этих функций возвращают строки, функция `Length()` возвращает целое число. Вот общий синтаксис вызова встроенных функций работы со строками в Visual Basic .NET. Здесь `возвращаемое значение` и `строка` — переменные:

```
Возвращаемое значение = Строка.ИмяСтроковой
Функции (Аргумент1, Аргумент2, ... АргументN)
```

Одна из самых полезных функций — функция `Length`. Она возвращает целое число — длину строки. Вот пример:

```
MyText = "TextBox1"
myLength = MyText.Length 'возвращает 9, длину
’"TextBox1"
```

Функции `ToLower` и `ToUpper` заменяют все буквы на строчные и на заглавные соответственно, например:

```
MyText = "TextBox1"
MyCaps = MyText.ToUpper 'возвращает "TEXTBOX1"

MyText = "TextBox1"
mySmalls = MyText.ToLower 'возвращает
'"textbox1"
```

Иногда нужно гарантировать, что в какой-то текстовой строке нет пробелов ни в начале, ни в конце. Проще говоря, если такие пробелы есть, их нужно удалить. Для этого можно воспользоваться функцией `Trim`:

```
MyText = " TextBox1"
myClean = MyText.Trim 'возвращает "TextBox1"

MyText = " TextBox1 "
MyCleaner = MyText.Trim 'возвращает "TextBox1"
```

Еще одна полезная функция — `Substring`. Эта функция возвращает часть большей строки. Вызывая функцию `Substring`, ей нужно передать в качестве аргументов два целых числа (тип `Integer`). Первый аргумент — номер символа в строке, с которого начинается нужная часть. Второй аргумент — длина этой нужной части. Заметьте, что номер первого символа в строке — 0, а не 1. Номер второго символа — 1, а не 2.

```
MyText = "TextBox1"
MySub = MyText.Substring(0, 4)'возвращает
'"Text", нужная часть начинается с позиции 0 и
'ее длина равна 4

MyText = "TextBox1"
MySub = MyText.Substring(1, 2)'возвращает "ex",
'нужная часть начинается с позиции 1 и ее длина
'равна 2
```

Наверно, вам не терпится увидеть эти функции в работе, так что давайте создадим маленькое приложение, в котором они будут использоваться! Создайте новое приложение Windows и назовите его «Встроенные функции». Поместите на форму два текстовых поля. Очистите значение свойства `Text` текстового поля `TextBox2`. Установите для свойства `Multiline` этого текстового поля значение `True`. Задайте для свойства `Scrollbars` этого текстового поля значение `Vertical`. Растяните текстовое поле `TextBox2`, чтобы его высота была почти такой же, как высота формы. Поместите на форму кнопку. В обработчик нажатия этой кнопки вставьте такой код:

```
Dim MyText As String
Dim TempText As String = ""
MyText = TextBox1.Text
TempText = TempText & MyText
TempText = TempText & vbNewLine &_
MyText.ToLower
TempText = TempText & vbNewLine &_
MyText.ToUpper
TempText = TempText & vbNewLine & MyText.Trim
TempText = TempText & vbNewLine &_
MyText.Substring(0, 4)
TempText = TempText & vbNewLine &_
MyText.Substring(1, 2)
TempText = TempText & vbNewLine & MyText.Length
TextBox2.Text = TempText
```

Постройте и запустите проект. Нажмите кнопку. В `TextBox2` будут выведены результаты выполнения каждой функции для свойства `Text` текстового поля `TextBox1`. Измените текст в `TextBox1` и снова нажмите кнопку.

Генератор случайных чисел. Во многих программах нужно генерировать случайные числа. Особенно часто это требуется, например, в играх, в которых события происходят с какой-то вероятностью, — например, при бросании костей или подбра-

сывании монетки. В платформе .NET Framework есть класс System.Random, в котором имеются функции, используемые для генерирования случайных чисел. Мы напишем код, который имитирует бросание игрального кубика. Добавьте еще одну кнопку на форму проекта «Встроенные функции». Измените значение свойства Text этой кнопки на Случайное число. Добавьте в обработчик ее нажатия такой код:

```
Dim myRandomGenerator As System.Random
Dim myRandomInteger As Integer
myRandomGenerator = New System.Random
myRandomInteger = myRandomGenerator.Next(1, 6)
MessageBox.Show(myRandomInteger)
```

Постройте и запустите приложение. Нажмите кнопку с надписью *Случайное число*. В окне сообщения будет выведено случайное число от 1 до 6. Еще раз нажмите кнопку. Будет выведено еще одно случайное число от 1 до 6.

Как работает этот код? Сначала мы объявляем переменную myRandomGenerator типа System.Random. Потом мы объявляем переменную myRandomInteger типа Integer. Чтобы сгенерировать случайное число от 1 до 6, мы вызываем метод Next() переменной myRandomGenerator. Вызывая метод Next(), мы передаем ему аргументы 1 и 6 как границы диапазона, в котором должно находиться нужное нам случайное число. Метод Next() генерирует это число, и оно присваивается переменной myRandomInteger. Последняя строка кода выводит сгенерированное случайное число в окне сообщения.

Функции преобразования. В Visual Basic .NET есть две функции преобразования, которые могут пригодиться при работе с числами. Это функция Val и функция Int. Сначала давайте разберемся с функцией Val. Эта функция преобразует строку в число. Если в строке есть дробная часть, функция Val возвращает значение типа Double. Если дроб-

ной части в строке нет, функция `Val` возвращает значение типа `Integer`. Эта функция часто используется, чтобы преобразовать значение свойства `Text` текстового поля в число, которое можно использовать в расчетах. Вот несколько примеров ее применения:

```
Dim MyInt As Integer
MyInt = Val("123") + 123 'возвращает 246
Dim MyDouble As Double
MyDouble = Val("123.22") + 123 'возвращает
’246.22
```

Функция `Int` возвращает целую часть числа, т. е. часть слева от десятичной точки. Посмотрите на примеры ее использования, чтобы понять, как она работает:

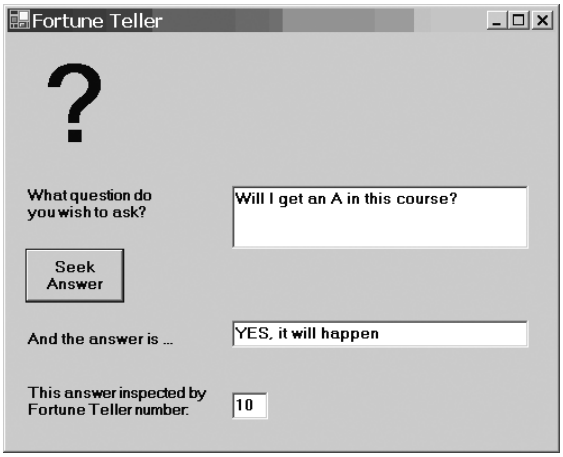
```
Dim MyInt As Integer
MyInt = Int(123) 'возвращает 123
MyInt = Int(123.45) 'возвращает 123
```

Microsoft-CD

Задания для самостоятельного выполнения

12.5. В системе программирования Visual Basic .NET создать проект «Встроенные функции», описанный в параграфе. Готовый проект содержится в самораспаковывающемся архиве Встроенные_функции.exe.

12.6. В системе программирования Visual Basic .NET создать проект «Предсказатель», который будет выдавать «предсказания» — отвечать на вопросы. Заготовка проекта хранится в самораспаковывающемся архиве Предсказатель.exe. Форма должна выглядеть примерно так:

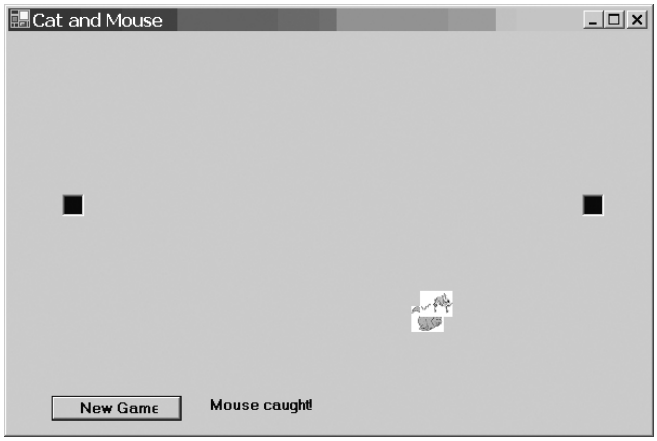


При нажатии кнопки с надписью *Seek Answer* используйте функцию `SubString`, чтобы подсчитать количество гласных букв в вопросе (A, E, I, O, U). Не забывайте, что важен регистр: «a» и «A» — это разные буквы. Кроме того, убедитесь, что введен именно вопрос. Вероятно, вам понадобится использовать цикл.

Помните, что функция `Length` возвращает количество символов в строке. Номера символов начинаются с 0, так что в строке из 10 символов номера этих символов — 0–9. Помните, что в Visual Basic .NET вместо числа всегда можно использовать переменную.

Исходя из количества гласных в вопросе, генерируйте разные ответы. Используйте свое воображение. Ответы «Yes» и «No» годятся, но это скучно.

12.7. В системе программирования Visual Basic .NET создать проект «Кот и мышь», в котором кот будет гоняться за мышью. Заготовка проекта хранится самораспаковывающимся архиве Кот_и_мышь.exe. Форма должна выглядеть примерно так:



Предполагайте, что ширина формы равна 600 пикселям, а высота — 300 пикселям.

Перемещайте кота и мышь в нескольких случайных направлениях, изменяя местоположение рамок рисунков, в которых показаны их изображения. Для этого прибавляйте случайные положительные и отрицательные числа к текущим значениям координат `X` и `Y`.

На форме есть элемент `Timer`, который генерирует события каждую четверть секунды. В обработчике этих событий и нужно выполнять перемещение кота и мыши. Это нельзя сделать напрямую. Чтобы переместить рамку рисунка, нужно вычислить новые координаты `X` и `Y` (числа), а затем воспользоваться такой строкой кода:

```
Mouse.Location = New Point(X, Y)
```

Переместив кота и мышь, проверьте:

1) не попала ли мышь в норку;
2) не поймал ли кот мышь;
3) не нужно ли мыши изменить направление движения, потому что поблизости кот;
4) не вышли ли кот или мышь за пределы экрана.

Кот отображается в рамке рисунка `Cat`, а не в границах элемента управления `PictureBox1`.

Мышь отображается в рамке рисунка `Mouse`, а не в границах элемента управления `PictureBox2`.

Если кот ближе, чем в 20 пикселях от мыши (расстояние отмеряется от верхнего левого угла рамки рисунка), то мышь поймана. Если мышь ближе, чем в 20 пикселях от любого черного текстового поля (расстояние отмеряется от верхнего левого угла рамки рисунка и верхнего левого угла текстового поля), то мышь убежала в норку. В любом из этих случаев нужно остановить таймер.

Если мышь ближе, чем в 40 пикселях от кота, она изменяет направление движения.

Координаты первой норки — (50, 150). Координаты второй норки — (530, 150).

В шаблоне есть еще несколько вещей, которые помогут вам написать эту программу.

Функция `HowFar` возвращает расстояние между двумя точками, если вы передадите ей координаты `X` и `Y` этих точек. Например:

```
Distance = HowFar(Cat.Location.X,_
Cat.Location.Y, Mouse.Location.X,_
Mouse.Location.Y)
```

Функция `NewChg` возвращает случайное число, которое может быть положительным или отрицательным. Это число обозначает расстояние, на которое должны переместиться кот или мышь. Например:

```
CatXChg = NewChg()
```

В обработчике нажатия кнопки с надписью *New Game* задайте случайные начальные позиции для кота и мыши. Присвойте координатам `X` и `Y` начальные значения. Запустите таймер.

Подсказка. Многие программисты предпочитают написать небольшую часть кода, проверить ее, затем написать еще небольшую часть и снова проверить и т. д. Эту программу лучше всего писать в такой последовательности:

1. Поместите кота и мышь.
2. Переместите кота.
3. Проверьте, не вышел ли кот за границы окна.
4. Переместите мышь.
5. Проверьте, не вышла ли мышь за границы окна.
6. Проверьте, не убежала ли мышь в норку.
7. Проверьте, не поймал ли кот мышь.
8. Проверьте, близко ли мышь от кота.

Отслеживайте, в каком направлении двигаются кот и мышь. Разворачивайте их изображения, если кот или мышь начинают двигаться в другом направлении. Используйте для этого примерно такой код:

```
Cat.Image.RotateFlip(RotateFlipType.Rotate180FlipX)
```

Определяйте, в каком направлении мышь от кота, а затем изменяйте направление движения кота, чтобы кот двигался к мыши. Например, если координата `X` кота равна 450, а мыши — 250, то кот должен двигаться в направлении уменьшения координаты.

12.6. Функции в J# и C#

В C# и J# тоже можно создавать собственные функции. Эквивалента подпрограммам в J# и C# нет. Кроме того, ключевое слово `Function` тоже не используется. Функции объявляются с помощью специального синтаксиса.

Функции в языке C#. Давайте посмотрим на код в C#, объявляющий функцию с двумя аргументами. Эта функция возвращает значение типа `Integer`.

```
private static int Adders_C(int oneAdd, int
twoAdd)
{
    int intSum=0;
    intSum = oneAdd + twoAdd;
    return intSum;
}
```

Заметьте, что мы не используем здесь ключевое слово `Function`, как в Visual Basic .NET. Ключевые слова `private` и `static` показывают, что эту функцию можно вызвать только из кода в данной форме. Ключевое слово `int` (после `static`) указывает тип возвращаемого значения (в нашем случае `Integer`). Два аргумента, используемые функцией, указаны в скобках после имени функции `Adders_C`. Оба аргумента, `oneAdd` и `twoAdd`, принадлежат к типу `int`. Обратите внимание на ключевое слово `return`. Оно указывает, какое значение должна вернуть функция (в нашем случае `intSum`).

Функция `Adders_C` в коде на C# вызывается так (точно так же, как в Visual Basic .NET):

```
int theSum=0;
theSum = Adders_C(3, 4);
MessageBox.Show(System.Convert.ToString(theSum));
```

Функции в языке J#. В коде на J# функции объявляются так же, как и в коде на C#. Посмотрите на этот код на J# и сравните его с кодом на C#:

```
private static int Adders_J(int oneAdd, int
twoAdd)
{
    int intSum=0;
    intSum = oneAdd + twoAdd;
    return intSum;
}
```

В коде на J# функция вызывается точно так же, как и в коде на C#. Посмотрите на этот код на J#:

```
{
    int theSum=0;
    theSum = Adders_J(3, 4);
    MessageBox.Show(System.Convert.ToString
    (theSum));
}
```

Тест по теме «Подпрограммы и функции»

1. 

Какое утверждение истинно?

- ❍ *И функция, и подпрограмма возвращают значение.*
- ❍ *Функция возвращает значение, а подпрограмма нет.*
- ❍ *Функция может иметь только один аргумент.*
- ❍ *Подпрограмма может иметь только один аргумент.*

2. 

Как выбрать фрагмент `"BC"` **из строки** `X = "ABCD"`**?**

- ❍ `X.Substring(1, 2)`
- ❍ `X.Substring(2, 3)`
- ❍ `X.Substring(2, 2)`
- ❍ `X.Substring(1, 3)`

3. 

Как вызвать подпрограмму `SubroutineName`**?**

- ❍ `Start SubroutineName()`
- ❍ `Go To SubroutineName()`
- ❍ `Execute SubroutineName()`
- ❍ `SubroutineName()`

4. 

Как преобразовать число в виде строки в число в виде числовой переменной?

- ❍ `X.ToNumber`
- ❍ `Convert(X)`
- ❍ `Val(X)`
- ❍ `Int(X)`

Оглавление