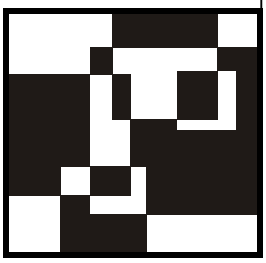
ГЛАВА 6



Отладка и обработка ошибок в программе

6.1. Типы ошибок

Если вы выполняли все задания для самостоятельной работы, то, наверное, уже заметили, что при написании программного кода допустить ошибку очень просто. Одна из задач разработчика — найти такие ошибки и устранить их (или обеспечить перехват ошибок времени выполнения и нормальную работу приложения даже в случае возникновения этих ошибок).

Все ошибки можно разделить на три большие группы:

- *синтаксические* (неправильно написан оператор, имя переменной и т. п.). Такие ошибки не требуют больших усилий по их поиску и исправлению. Многие синтаксические ошибки "отлавливаются" редактором кода VBA еще в процессе ввода кода. Об обнаружении других ошибок сообщается в ходе компиляции и запуска программы. При этом компилятор VBA выдает информацию о том, в какой строке кода обнаружена ошибка и в чем она заключается. Рекомендуется проверить данную строку по справке VBA;
- *логические*. В ходе выполнения программа ведет себя не так, как вы планировали. Главное здесь — найти причину неправильного поведения программы. Обычно для выявления и исправления ошибок такого типа предназначены приемы отладки *(см. разд. 6.2)*;
- *ошибки времени выполнения* (run-time error). Они возникают, когда в процессе выполнения программа столкнулась с проблемой, решить которую она не в состоянии (файл с таким именем уже существует, возник конфликт записей при вставке в базу данных, произведена попытка записать информацию на переполненный диск и т. п.). Заранее предугадать, какая именно неприятность может случиться, очень сложно. Во многом квалификация программиста определяется тем, как он умеет предугадывать

возможности возникновения ошибок времени выполнения и обеспечивать их перехват и обработку.

Если программа делается "для себя" (для автоматизации работы того пользователя, который пишет эту программу), то очень часто перехват ошибок времени выполнения вообще не предусматривается. Возникла ошибка — ничего страшного: открыли программу в отладчике, посмотрели, отчего возникла ошибка, и "исправились". Но если программа пишется для передачи другим пользователям (особенно не очень квалифицированным), то на реализацию обработки ошибок времени выполнения обычно уходит больше времени, чем на создание самой логики программы.

6.2. Приемы отладки. Окна *Immediate*, *Locals* и *Watch*

6.2.1. Тестирование

Главный способ обеспечения безошибочной работы программы — это ее тестирование. При создании крупных программных продуктов на их тестирование часто уходит не меньше времени, чем на создание. Поскольку в наших условиях рассчитывать на то, что тестировать вашу программу будет профессиональный тестер, не приходится, проверять ее придется вам самим. Приведу некоторые советы по тестированию:

- попытайтесь запустить программу при работе с большим количеством документов или когда не открыто ни одного документа;
- посмотрите, как работает программа, когда окно документа развернуто, свернуто или размер его изменен;
- проверьте, как работает программа, когда выделены разные элементы или группы элементов;
- если предусматривается ввод информации, попробуйте специально передать программе неверные значения. Например, если программа ожидает числовых значений, попробуйте ввести строковое значение, значение даты или оставить поле пустым;
- попробуйте прервать работу программы в самый неподходящий момент и потом вновь запустить ее;
- проверьте, как ведет себя программа, когда пропадает сеть, заканчивается свободное место на диске, заканчивается бумага в принтере и т. п.;
- проверьте работу программы под разными версиями Office и операционных систем (в том числе англоязычных и локализованных);

- попробуйте до запуска программы и во время ее работы переставлять системную дату и время, устанавливая самые невероятные значения.

Если есть возможность, всегда рекомендуется немного поработать, выполняя обязанности пользователя, для которого создается программа.

Мне очень нравится "диверсионный" подход при тестировании программ. Представьте себе, что вы — вредитель и диверсант, у которого цель — вывести программу из строя. Потом опробуйте те способы, которые вам пришли в голову. Если способ оказался удачным, придумайте для него защиту. Как ни удивительно, но реальная работа пользователей с вашей программой будет очень похожа на действия таких диверсантов.

6.2.2. Переход в режим паузы

Один из самых важных приемов в ходе отладки программы — возможность вовремя остановиться в ходе выполнения, чтобы просмотреть значения переменных, вмешаться в ход выполнения программы вручную, просмотреть, что возвращает оператор или функция и т. п.

Программу в режим паузы можно перевести следующими способами:

- с самого начала запустить программу в режиме пошагового выполнения (меню **Debug | Step Into** или клавиша <F8>). В этом случае программа будет переходить в режим паузы после выполнения каждого оператора;
- установить в программе точку останова *(breakpoint)*. Это можно сделать, поставив указатель на нужной строке и в меню **Debug** выбрав **Toggle Breakpoint** (или нажав клавишу <F9>). Строка с точкой останова будет помечена коричневым цветом, и точка такого же цвета появится на рамке слева от строки. Второй вариант — просто щелкнуть мышью по рамке слева от строки. Снятие точки останова — повторить то же самое действие еще раз. При запуске программа автоматически остановится на первой точке останова;
- к сожалению, точки останова не сохраняются после закрытия документа. Если нужно запомнить место остановки между сеансами отладки, то нужно просто впечатать в это место строку с единственной командой `Stop`. Программа в ходе выполнения автоматически остановится на этой строке, например:

```
n1 = 10
n2 = 5
Stop
nResult = n1/n2
```

- если программа не хочет завершаться (например, у вас выполняется бесконечный цикл), в ходе ее выполнения можно нажать кнопку **Break** на па-

нели инструментов **Standard**, воспользоваться меню **Run | Break** или просто нажать клавиши <Ctrl>+<Break>;

- еще одна возможность приостановить выполнение программы — воспользоваться контролируемым выражением (в окне **Watches**). Об этом — в *разд. 6.2.6*.

В любом случае выполнение будет приостановлено в выбранном вами месте программы, и следующий оператор, который должен быть выполнен, будет выделен желтым цветом (рис. 6.1).

Что делать дальше, рассказано в следующем разделе.

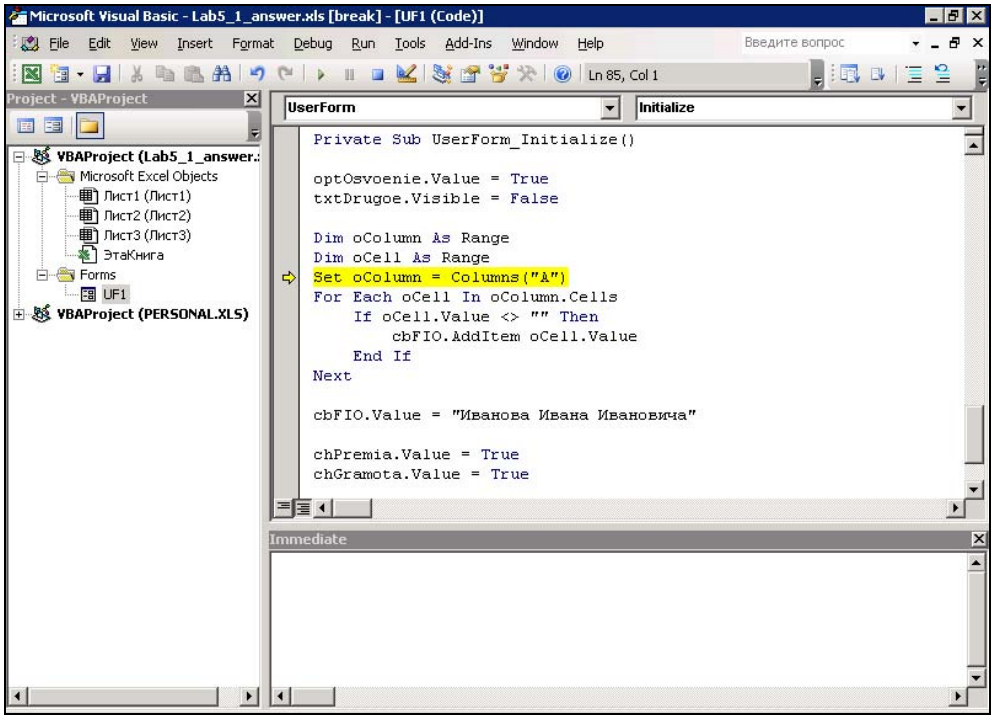


Рис. 6.1. Выполнение программы приостановлено

6.2.3. Действия в режиме паузы

В режим паузы обычно входят для того, чтобы предпринять какие-то действия. В этом режиме можно сделать следующее:

- продолжить выполнение в пошаговом режиме. Для этого можно воспользоваться меню **Debug | Step Into** или нажать клавишу <F8>. При этом бу-

дет выполнена текущая строка (помеченная желтым цветом), и выделится следующая строка вашего программного кода. Если была выделена строка, в которой происходит вызов процедуры или функции, то мы "шагнем" *(Step Into)* внутрь этой процедуры;

- если в строке программы происходит вызов какой-то процедуры, которая уже отлажена и проблем вызвать не должна, то у вас есть возможность выполнить ее без остановок и перейти к следующему оператору. Для этой цели используется команда меню **Debug | Step Over** (или <Shift>+<F8>);
- если вы все-таки зашли в процедуру, выполнили там необходимые действия и хотите быстро довести ее выполнение до конца, то в вашем распоряжении — меню **Debug | Step Out** (или <Ctrl>+<Shift>+<F8>);
- если вы хотите исполнить код не пошагово, а фрагментами (например, чтобы не проводить пошаговое выполнение больших циклов), вы можете щелкнуть правой кнопкой мыши по нужной строке кода и в контекстном меню выбрать **Run to Cursor** (альтернатива — воспользоваться той же командой в меню **Debug** или нажать <Ctrl>+<F8>). Программа пройдет вперед на выбранную вами строку и вновь остановится;
- если вам нужно "перепрыгнуть" через какой-то фрагмент кода, вызывающий ошибку (т. е. просто пропустить его без выполнения), можно воспользоваться командой меню **Debug | Set Next Statement** (или нажать <Ctrl>+<F9>), а затем перетащить желтую стрелку слева от кода вниз (или вверх) на нужную строку. В последних версиях Office в режиме останова можно сразу перетаскивать желтую стрелку, не вызывая команды. Альтернативный способ пропуска команд — выделить ненужный блок и при помощи кнопки **Comment Block** панели инструментов **Edit** его закомментировать (но тогда потом придется снимать комментарии);
- если в процессе просмотра кода вы пролистали код достаточно далеко и вам хочется вернуться к месту остановки (строка, выделенная желтым) без долгих поисков, то в вашем распоряжении команда **Show Next Statement**;
- чтобы просто продолжить выполнение программы после остановки, можно нажать клавишу <F5> или воспользоваться командой меню в меню **Run | Continue** (она появится вместо команды **Run**). Прекратить выполнение программы можно при помощи команды **Reset** в том же меню или клавишами <Alt>+<F4>. Кроме того, в вашем распоряжении одноименные кнопки на панели инструментов **Standard**.

Чаще всего переход в режим остановки нужен, чтобы просмотреть текущие значения переменных или исправить код. Текущие значения переменных можно просмотреть при помощи окон **Immediate**, **Locals** или **Watch** (о них будет рассказано в следующих разделах), а можно воспользоваться подсказ-

ками в самом коде. Для того чтобы получить информацию о текущем значении переменной, достаточно навести на нее указатель мыши — рис. 6.2 (если у вас остался включен по умолчанию параметр **Auto Data Tips** в окне **Options**, меню **Tools | Options**).

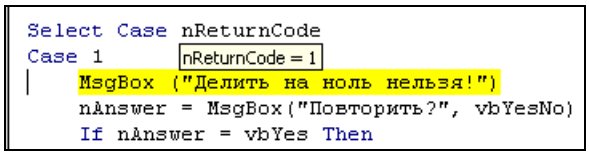


Рис. 6.2. Просмотр текущих значений переменных

Чтобы просмотреть информацию об области видимости и типе данных переменной, необходимо установить на нее курсор ввода текста и в меню **Edit** выбрать **Quick Info** (можно нажать клавиши <Ctrl>+<I>).

Все возможности редактирования кода в режиме отладки остаются. Если вы поняли, что переменной (свойству объекта) присвоено неверное значение, то вы можете исправить соответствующую строку, вернуться назад при помощи **Set Next Statement** (<Ctrl>+<F9>) и продолжить выполнение с исправленным значением.

Но возможностей подсказок в коде нам не всегда достаточно. Когда переменных очень много, или, например, к проблемам может привести значение свойства, которое мы даже не назначали, или во многих других ситуациях нам могут потребоваться специализированные окна **Immediate**, **Locals** и **Watch**.

6.2.4. Окно *Immediate*

Это мое любимое средство отладки. Окно **Immediate** предназначено для немедленного *(immediate)* выполнения программного кода, вызвать его можно через меню **View** или клавишами <Ctrl>+<G>. В этом окне можно:

- просматривать и изменять значения переменных и свойств объекта. Посмотреть значения переменных можно, набрав в окне **Immediate**:

```
Print nResult
Print oDoc.FullName
```

или еще проще:

```
?nResult
?oDoc.FullName
```

В данном случае `Print()` — это метод объекта `Debug`. Вывод в окно **Immediate** можно произвести при помощи этого объекта и просто из кода программы:

```
Debug.Print nResult
```

Преимуществом этого метода перед обычным `MsgBox()` является то, что при работе не в отладочном окружении (т. е. когда ваша программа уже эксплуатируется пользователями) все вызовы методов объекта `Debug` просто игнорируются (у этого объекта есть еще один метод `Assert()` — переход по условию).

Изменение значений переменных и свойств в окне **Immediate** производится точно так же, как в коде программы;

- вызывать процедуры и функции вашей программы или методы объектов — точно так же, как в коде программы. Microsoft рекомендует перед вставкой в программу проверять потенциально опасный код (например, который может привести к зависанию системы) в этом окне;
- а можно использовать это окно просто как калькулятор, вводя там выражения вида:

```
Print 25*115
```

Пример окна **Immediate** с некоторыми выполненными действиями представлен на рис. 6.3.

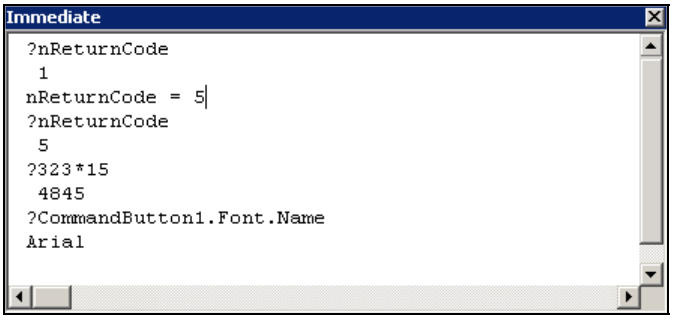


Рис. 6.3. Пример использования окна **Immediate**

Чтобы не печатать в окне **Immediate** выражения и имена переменных, которые уже есть в коде, можно воспользоваться перетаскиванием их в окно **Immediate** из окна редактора кода с нажатой клавишей <Ctrl> (чтобы происходило копирование).

6.2.5. Окно *Locals*

Очень часто бывает так, что вам нужно просмотреть значения всех переменных и свойств объектов, чтобы определить недопустимые, и сразу же их поменять. В этом случае возиться с каждым свойством/переменной в окне **Immediate** неэффективно. В этой ситуации гораздо удобнее использовать окно **Locals** (меню **View | Locals Window**). В этом окне выводятся значения всех переменных и свойств объектов, доступных в настоящий момент (пример окна приведен на рис. 6.4).

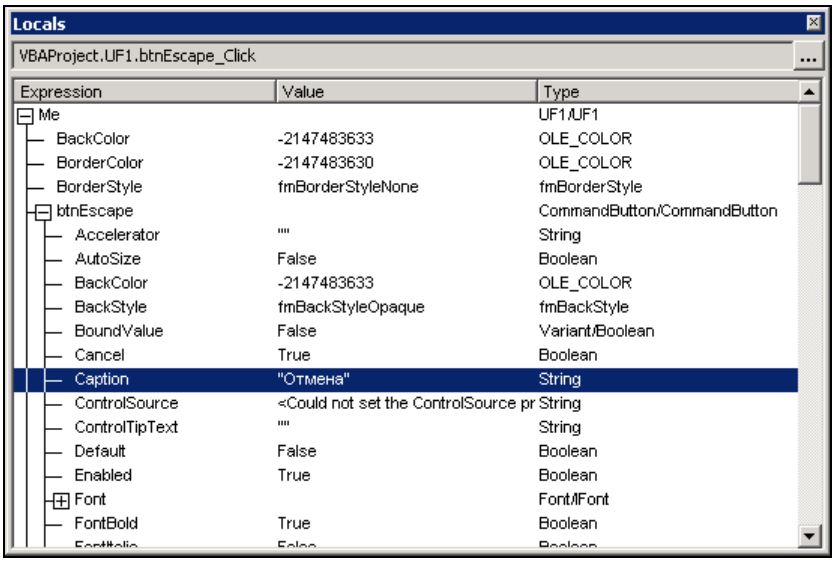


Рис. 6.4. Пример использования окна **Locals**

Чтобы поменять значение переменной или свойства, необходимо выделить нужную строку в окне **Locals**, а потом в этой строке аккуратно курсором выделить в столбце **Value** значение. Поверх старого значения можно впечатать новое. Если значение должно быть строковым, то при печати заключите его в кавычки (как в коде), а если это значение даты — то в символы решетки (#).

Через окно **Locals** можно также менять значения элементов массивов и коллекций.

6.2.6. Окно *Watches*

Окно **Watches** (его можно открыть так же, как и остальные, при помощи меню **View**) позволяет контролировать ход выполнения программы, производя "наблюдение" в соответствии с заданными вами условиями. При работе с ним

вам вначале потребуется определить значение, которое послужит сигналом к тому, чтобы вмешаться в ход выполнения программы. Это значение называется контролируемым выражением. Оно может быть совсем простым, например, `nResult = 10`, а может быть сложным, таким как:

```
InStr(oDoc.Fullname, "Document") <> 0
```

Создать контролируемое выражение можно так:

- щелкнуть по переменной, свойству или выражению в окне редактора кода правой кнопкой мыши и в контекстном меню выбрать **Add Watch**. Откроется диалоговое окно **Add Watch** для задания данного выражения;
- воспользоваться командой **Add Watch** в меню **Debug**;
- воспользоваться командой **Quick Watch** в меню **Debug**. В этом случае в окно **Watch** будет помещено автоматически сгенерированное выражение в зависимости от того, в каком месте кода находился у вас курсор ввода. В качестве условия для "срабатывания" в него будет помещено текущее значение переменной или свойства. Это контролируемое выражение в случае необходимости можно будет отредактировать;
- просто перетащить выражение из кода в окно **Watches**.

В любом случае откроется окно **Add Watch**. В нем вы должны написать (или дописать) контролируемое выражение, чтобы оно возвращало `True` или `False`, как и в конструкции `If...Else`, выбрать "область действия" данного контролируемого выражения (в виде процедуры или модуля), а также принять главное решение: что делать в ходе наблюдения. Вариантов у вас три:

- **Watch Expression** — ничего не делать (просто менять значение в столбце **Value** в окне **Watches**);
- **Break When Value Is True** — переводить программу в режим паузы, если контролируемое выражение "сработало" (его значение стало равно `True`);
- **Break When Value Changes** — переводить программу в режим паузы, если значение контролируемого выражения изменилось.

Окно **Watches** позволяет отследить происходящее в вашей программе даже в самых тяжелых случаях, когда понять, почему все работает именно так, а не иначе, очень сложно.

6.3. Перехват и обработка ошибок времени выполнения

Самое тяжелые для разработчика ошибки — это ошибки времени выполнения, которые могут возникнуть по самым разным причинам: пользователь

ввел недопустимое значение, файл с таким именем уже существует, сервер баз данных отказывается вставлять введенные пользователем значения, разорвано сетевое соединение и т. п. При возникновении ошибок времени выполнения обычно работа приложения аварийно завершается, а пользователю выдается встроенное сообщение, которое он вряд ли сможет расшифровать. Поэтому одна из самых трудоемких задач при создании программы на VBA — предусмотреть, какие ошибки могут возникнуть при работе пользователя и реализовать их обработку.

Общий принцип обработки ошибки выглядит так:

1. Перед опасным кодом (сохранение или открытие файла, возможность деления на ноль и т. п.) помещается команда:

   ```
   On Error GoTo метка_обработчика_ошибки
   ```

   например:

   ```
   Dim a As Integer, b As Integer, c As Integer
   On Error GoTo ErrorHandlerDivision
   c = a / b
   ```

2. Далее в коде программы помещается метка обработчика ошибки и программный код обработки:

   ```
   ErrorHandlerDivision:
       MsgBox "Ошибка при делении"
   ```

3. Поскольку в такой ситуации код обработчика ошибки будет выполняться даже в том случае, если ошибки не было, есть смысл поставить перед меткой обработчика команду `Exit Sub` (если это подпроцедура) или `Exit Function` (если это функция). Полный код нашей мини-программы может выглядеть так:

   ```
   Private Sub UserForm_Click()
       Dim a As Integer, b As Integer, c As Integer
       On Error GoTo ErrorHandlerDivision
       c = a / b
       Exit Sub
       ErrorHandlerDivision:
           MsgBox "Ошибка при делении"
   End Sub
   ```

   Как правило, если есть возможность исправить ошибку, то в обработчике ошибок ее исправляют (или предоставляют такую возможность пользователю), если нет — то выдают пользователю сообщение с объяснением и прекращают работу программы.

4. После выполнения кода обработчика ошибки вам нужно будет сделать выбор: либо продолжить выполнение той процедуры, в которой возникла ошибка, либо прекратить ее выполнение и передать управление вызвавшей ее процедуре. В вашем распоряжении три варианта:
   - еще раз выполнить оператор, вызвавший ошибку (если обработчик ошибки может определить характер возникший проблемы). Для этого достаточно в обработчик ошибок вставить команду `Resume`;
   - пропустить оператор, вызвавший ошибку. Для этой цели можно использовать команду `Resume Next`;
   - продолжить выполнение с определенного места в программе. Для этого используется команда `Resume` *`метка`*. Синтаксис работы с меткой — такой же, как в операторе `GoTo`.

Отметим еще несколько моментов, которые связаны с обработкой ошибок:

- чтобы вернуться в нормальный режим работы после прохождения опасного участка кода (отменить обработку ошибок), можно воспользоваться командой:

```
On Error GoTo 0
```

- в вашем распоряжении имеется также команда `On Error Resume Next`. Она предписывает компилятору просто игнорировать все возникающие ошибки и переходить к выполнению следующего оператора. На практике очень часто перед выполнением опасного оператора используется эта команда, а затем при помощи конструкции `Select Case` проверяется номер возникшей ошибки (через свойства объекта `Err`), и в зависимости от этого организуется дальнейшее выполнение программы.

Рассмотрим чуть подробнее специальный объект `Err`. У этого объекта есть два главных свойства и два метода.

- `Number` — это свойство показывает номер ошибки. Обычно оно и проверяется в обработчике ошибок, чтобы выяснить, какая именно ошибка возникла. Если номер ошибки равен 0, то ошибки не было.
- `Description` — текстовое описание возникшей ошибки. Именно оно по умолчанию возвращается пользователю (хотя пользователь вряд ли в нем что-либо поймет). Скорее это информация для разработчика.
- `Clear()` — этот метод очищает объект `Err` от старой информации об ошибках. То же самое делает и команда `On Error GoTo 0`.
- `Raise()` — позволяет сгенерировать ошибку в программе, передав ей номер и описание. Очень полезная возможность для проверки поведения программы, если смоделировать ситуацию с реальной ошибкой трудно.

Надо сказать, что обработка ошибок — это очень надежный, но и очень ресурсоемкий метод работы. Если в вашей программе есть возможность обойтись без генерации и перехвата ошибок (например, проверять вводимое пользователем значение при помощи встроенных функций), то лучше так и делать. В то же время наличие обработчиков ошибок, чтобы справляться с действительно аварийными ситуациями, — это большой плюс вашей программе (а зачастую и просто необходимость).

Задание для самостоятельной работы 6: Перехват ошибок времени выполнения

Подготовка:

1. Создайте новый файл Excel и сохраните его как C:\ErrorHandling.xls.
2. В ячейку A1 этого файла введите значение `"Результат деления:"`.
3. Щелкните правой кнопкой мыши по любой панели инструментов или меню и в открывшемся списке доступных панелей инструментов выберите **Элементы управления**.
4. На панели инструментов **Элементы управления** нажмите кнопку **Режим конструктора** (верхняя левая кнопка) и в этом режиме поместите на лист Excel новую кнопку. Для этого нужно щелкнуть по объекту **Кнопка** на панели инструментов **Элементы управления** и на листе определить местонахождение и размеры этой кнопки.
5. Щелкните по созданной вами кнопке правой кнопкой мыши и в контекстном меню выберите **Свойства**. Определите для нее свойства по вашему усмотрению. Выглядеть лист с кнопкой в итоге может, например, так, как показано на рис. 6.5.
6. В режиме конструктора щелкните по кнопке правой кнопкой мыши и в контекстном меню выберите **Исходный текст**. Откроется редактор кода Visual Basic с созданной процедурой для события `Click` данной кнопки. Поместите в него следующий код:

```
Private Sub CommandButton1_Click()
    Dim nNum1 As Integer
    Dim nNum2 As Integer
    Dim nResult As Integer
    nNum1 = InputBox("Введите первое число")
    nNum2 = InputBox("Введите второе число")
    nResult = nNum1 / nNum2
    Range("B1").Value = nResult
End Sub
```

7. Вернитесь на ваш лист Excel, выйдите из режима конструктора (щелкнув по кнопке **Выход из режима конструктора** на панели инструментов **Элементы управления**) и нажмите на созданную вами на листе кнопку. Убедитесь, что если вводить допустимые значения для делимого и делителя, то код работает правильно и выводит результат деления в ячейку B2.

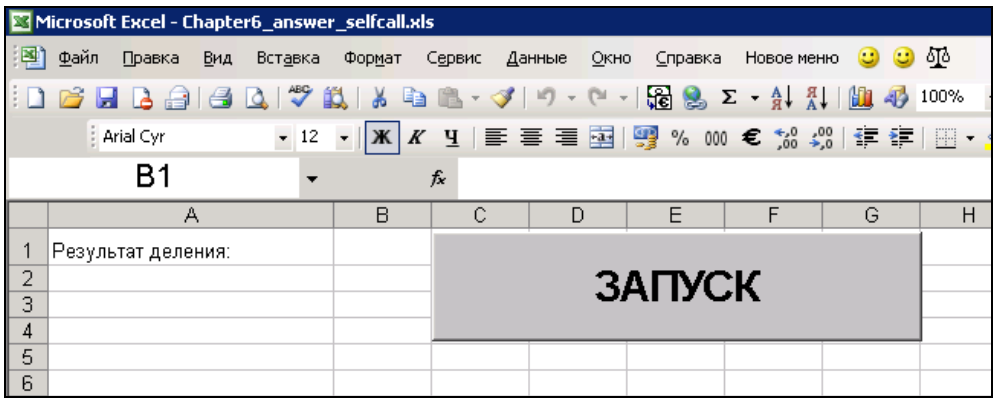


Рис. 6.5. Интерфейс вашей программы

ЗАДАНИЕ:

Измените эту программу таким образом, чтобы обеспечить защиту от ввода пользователем недопустимых значений (например, строковых значений делимого и делителя или значения делителя, равного 0).

Примечание

Не обязательно оставлять код по приему значений от пользователя и выполнению деления в обработчике события `Click` вашей кнопки. Этот код при желании можно перенести во внешние процедуры или функции.

Ответ к заданию 6

Существует множество вариантов решения этой задачи. Примеры приведены далее.

Так, например, может выглядеть решение с обработкой кода ошибки и повторным вызовом функцией самой себя:

```
Option Explicit
Private Sub CommandButton1_Click()
    Call subPrepare
End Sub
```

```
Public Sub subPrepare()
    Dim nReturnCode As Integer
    Dim nAnswer As Integer

    nReturnCode = fDiv()

    Select Case nReturnCode
    Case 1
        MsgBox ("Делить на ноль нельзя!")
        nAnswer = MsgBox("Повторить?", vbYesNo)
        If nAnswer = vbYes Then
            Call subPrepare
        Else
            Application.Quit
        End If
    Case 2
        MsgBox ("Нужно число!")
        nAnswer = MsgBox("Повторить?", vbYesNo)
        If nAnswer = vbYes Then
            Call subPrepare
        Else
            Application.Quit
        End If
    Case 3
        MsgBox ("Неизвестная ошибка")
        nAnswer = MsgBox("Повторить?", vbYesNo)
        If nAnswer = vbYes Then
            Call subPrepare
        Else
            Application.Quit
        End If
    End Select
End Sub

Function fDiv()
    On Error Resume Next
    Dim nNum1 As Integer
    Dim nNum2 As Integer
    Dim nResult As Integer

    nNum1 = InputBox("Введите первое число")
    nNum2 = InputBox("Введите второе число")

    nResult = CInt(nNum1) / CInt(nNum2)
```

```
    Select Case Err.Number
    Case 0
        Range("B1").Value = nResult
        fDiv = 0
    Case 11
        fDiv = 1
    Case 13
        fDiv = 2
    Case Else
        fDiv = 3
    End Select
End Function
```

А вот решение с обработкой кода ошибки и циклом:

```
Private Sub CommandButton1_Click()
    Dim nNum1 As Variant
    Dim nNum2 As Variant
    Dim nResult As Integer
    Dim nError As Integer

    Do
        nNum1 = InputBox("Введите первое число:")
        On Error Resume Next
        nError = CInt(nNum1)
        If Err.Number = 13 Then
            MsgBox ("Нужно число")
            nNum1 = ""
        End If
        On Error GoTo 0
    Loop While (nNum1 = "")

    Do
        nNum2 = InputBox("Введите второе число:")
        On Error Resume Next
        nError = CInt(nNum2)
        If Err.Number = 13 Then
            MsgBox ("Нужно число")
            nNum2 = ""
        ElseIf nNum2 = 0 Then
            MsgBox ("Делить на ноль нельзя!")
            nNum2 = ""
        End If
        On Error GoTo 0
    Loop While (nNum2 = "")
```

```
    nResult = nNum1 / nNum2

    Range("B1").Value = nResult

End Sub
```

Еще один вариант решения вообще не допускает возникновения ошибок:

```
Private Sub CommandButton1_Click()
    Dim nNum1 As Variant
    Dim nNum2 As Variant
    Dim nResult As Integer

    Do
        nNum1 = InputBox("Введите первое число:")
        If IsNumeric(nNum1 & "") Then Exit Do
        MsgBox "Нужно число"
    Loop

    Do
        nNum2 = InputBox("Введите второе число:")
        If IsNumeric(nNum2 & "") Then
            If Int(nNum2) <> 0 Then Exit Do
            MsgBox "Делить на ноль нельзя!"
        Else
            MsgBox "Нужно число"
        End If
    Loop

    nResult = nNum1 / nNum2

    Range("B1").Value = nResult

End Sub
```