# Советы тем, кто программирует на Visual Basic и MS Office/VBA

Андрей Колесов
Ольга Павлова

## Совет 170. Используйте функцию CopyMemory из Win32 API

При разработке приложений довольно часто встречается простая задача пересылки N последовательных байтов из одной области оперативной памяти в другую. В VB проблема возникает, когда такая область памяти определена не именем переменной, а в виде адреса (указателя —Pointer). К сожалению, встроенные средства VB не позволяют работать с адресами, а порой это крайне необходимо.

Так, некоторые функции Win API, используемые для получения информации о ресурсах системы (например, VerQueryValue), возвращают в качестве параметра адрес области памяти FiAddr&, где лежит искомая информация. Но как переписать эти данные, к примеру, в целочисленный массив для дальнейшей обработки в среде VB? Это можно сделать с помощью функций Win API.

В составе Win16 API для этого имелась функция HMemCpy:

```
Declare Sub HMemCpy Lib «kernel»_ (hpvDest As Any, hpvSource As Any, ByVal
        cbCopy As Long)
```

Она появилась только в расширенном варианте функций в Windows 3.1 (в версии 3.0 ее не было), а в пакете VB 3.0 ее описание было приведено только в файле WIN31WH.HLP.

Что же касается Win32 API, то в описании WIN32API.TXT для VB 4.0 и 5.0 о функции HMemCpy или о ее аналоге даже не упоминается. Более того, в книге Дэна Апплмана по Win32 API, на которую мы часто ссылаемся (о ней мы писали в Совете 133), ничего не говорится о подобной функции. Однако, поскольку такая функция действительно очень нужна, Дэн предлагает использовать процедуру agCopyData из его собственной библиотеки APIGI32.DLL, которая приводится на прилагаемом компакт-диске.

Все это выглядит весьма странно. Более тщательный поиск, проведенный с помощью специалиста московского отделения Microsoft Юрия Томашко, увенчался успехом — в Win32 есть функция копирования байтов CopyMemory, и ее описание появилось в VB 6.0 в файле WIN32API.TXT. Почему Microsoft скрывает от пользователей VB 4.0 и 5.0 ее существование, и почему Дэн Апплман хранит эту тайну — можно только гадать. Кстати, настоящее имя функции в библиотеке KERNEL32 —RtlMoveMemory:

```
Public Declare Sub CopyMemory Lib «kernel32» Alias «RtlMoveMemory» _
        (Destination As Any, Source As Any, ByVal Length As Long)
```

Вот типичный пример применения этой функции:

```
Di& = VerQueryValue(VerBuf(0), «\»,FiAddr&, FiLen&) ' Возвращается: '
        FiAddr& – адрес области с информацией о версии файла ' FiLen& –
        длина области в байтах CopyMemory ByVal FiAddr&, Ffi, 52 ' Ffi –
        структура данных для информации о версии ' 52 – ее длина
```

CopyMemory можно использовать для создания очень полезных функций преобразования типов данных (см. Совет 171), а также для ускорения в сотни и даже тысячи раз выполнения операций, реализуемых традиционными средствами VB. Например, для копирования целых массивов или их фрагментов:

```
Dim Arr1!(1000), Arr2!(2000), i% For i = 1 To 1000: Arr1(i) = i + 0.1:
        Next ' Вместо: ' For i = 1 To 1000: Arr2(1000 + i) = Arr1(i):
        Next ' Использовать: CopyMemory Arr2(1001), Arr1(1), 1000 * 4 '
        MsgBox («result =» & Arr2(1600))
```

## Совет 171. Копирование областей памяти в DOS

Дополнительные функции DLL-библиотек могут серьезно расширить возможности VB-программиста. При этом следует иметь в виду, что для написания таких процедур зачастую совсем не обязательно быть большим знатоком языка, на котором они будут писаться.

Например, функция копирования областей памяти пригодится и тем, кто еще работает в Basic/DOS. В силу специфики использования библиотек в этих версиях Basic (мы вновь сожалеем, что в VB/Win-проектах Microsoft не позволяет подключать к исполняемому модулю объектные библиотеки) такие внешние функции лучше всего было писать на Ассемблере. Посмотрите, какой простой код имеет функция StringCopy, написанная для варианта MASM 6.0, которая фактически является точным аналогом функции CopyMemory для режима DOS (только число байтов задается целочисленной переменной):

```
.MODEL Medium,Basic .CODE StringCopy PROC USES DS DI SI DF,
        SourceAddr:DWord, DestAddr:DWord, Len:Word ; прием входных
        параметров: MOV CX,Len          ; количество байт LES DI,DestAddr
        ; полный адрес Приемника (Куда) LDS SI,SourAddr     ; полный
        адрес Источника (Откуда) ; пересылка данных: CLD
        ; очистка флага DF REP MOVSB           ; пересылка CX-байт ;
        выход из процедуры: RET                ; возврат управления
        StringCopyByv ENDP END
```

Ее описание можно сделать двумя способами:

1. Адреса задаются с помощью двух 16-разрядных переменных — сегмент и смещение:

- описание:

```
DECLARE SUB StringCopy(BYVAL SourceSeg%, BYVAL SourceOff%,_ BYVAL
        DistSeg%, BYVAL DistOff%, BYVAL LenByte%)
```

- обращение:

```
CALL StringCopyByv(SourceSeg%, SourceOff%, _ DistSeg%, DistOff%, LenByte%)
```

2. Полные адреса задаются с помощью 32-разрядных переменных:

- описание:

```
DECLARE SUB StringCopy(BYVAL SourceAdr&, BYVAL DistAdr&, _ BYVAL LenByte%)
```

- обращение:

```
CALL StringCopyByv(SourceAddr&, DistAddr&, LenByte%)
```

## Совет 172. Как реализовать функции MKx$/CVx в VB/Win

Именно с этого совета (№ 3) три года назад мы начали свои публикации для пользователей VB в журнале КомпьютерПресс № 3'96. Напомнить о нем мы решили для иллюстрации применения функции CopyMemory.

Дело в том, что в DOS'овских версиях MS Basic (Quick, PDS, Visual) имелась группа очень полезных встроенных функций MKx$/CVx (x — тип числовых данных: I, L, S, D), которые почему-то пропали в VB/Win. (Их описание приводится во встроенной справке QBasic, которая входит в состав MS DOS 5.0 и 6.x.)

С помощью этих функций производится преобразование числовых (целых, вещественных и пр.) данных в строковый формат и наоборот. Точнее говоря, никакого преобразования значений здесь не выполняется, а просто N-е количество байт меняет название типа данных.

Основной смысл такого преобразования — это возможность хранения и передачи разнотипных данных в виде одной строковой переменной, что является отличной альтернативой структурам данных Type с их жестким описанием полей на уровне исходного текста. В свое время мы очень широко использовали в своей практике этот прием для создания гибких, динамически настраиваемых структур данных.

Применение данных функций позволяет осуществлять весьма изящные преобразования данных. Например, в Совете 117 (КомпьютерПресс, № 10'97) мы говорили о проблеме обработки беззнаковых целых чисел и приводили пример слияния двух 16-разрядных целых чисел в 32-разрядное представление и наоборот. Сравните приведенный там пример со следующим вариантом:

```
1. LongValue& => IntHigh% & IntLow% -----------------
IntHigh% = CVI(LEFT$(MKL$(LongValue&), 2)) IntLow% =
         CVI(RIGHT$(MKL$(LongValue&), 2)) ' вместо Совета 117 IntegetHigh%
         = LongValue& \ &H10000 IntegerLow% = LongValue& And &H7FFF If
         (LongValue& AND &H8000) <> 0 Then IntegerLow% = IntegerLow% Or
         &H8000 End If
2. IntHigh% & IntLow% => LongValue& -----------------
LongValue& = CVL(MKI$(IntHigh%) + MKI$(IntLow%)) 'вместо Совета 117
         LongValue& = IntHigh% * &H10000 + (IntegerLow% AND &H7FFF) If
         IntegerLow% < 0 Then LongValue& = LongValue& or &h8000&
```

С помощью функций Win API в VB/Win можно довольно просто реализовать эти полезные Basic-функции. Набор таких процедур приведен в модуле MKXCVX.BAS, а пример их применения — в модуле MKXCVXTS.BAS (см. листинг 1).

### Листинг 1

```
Attribute VB_Name = "MKxCVxTS"
Sub main()
   '
   ' Пример обращения к функциям MKx$, CVx
   ' в стиле "a-la" MS Basic for DOS (QB/PDS/VBDOS)
   IntValue% = 12345
   x$ = MKI$(IntValue%)
   MsgBox Str$(CVI(x$)), , "Integer"
   '
   LongValue& = 12345678
   x$ = MKL$(LongValue&)
   MsgBox Str$(CVL(x$)), , "Long"
   '
   SingleValue! = 123.45
   x$ = MKS$(SingleValue!)
   MsgBox Str$(CVS(x$)), , "Single"
   '
   DoubleValue# = 1.2345
   x$ = MKD$(DoubleValue#)
   MsgBox Str$(CVD(x$)), , "Double"
   '
End Sub

================================================================

Attribute VB_Name = "MKxCVx32"
#If Win32 Then
   Public Declare Sub HMemCpy Lib "kernel32" Alias _
       "RtlMoveMemory" (Destination As Any, Source As Any, _
       ByVal Length As Long)
#Else
   Declare Sub HMemCpy Lib "kernel" (hpvDest As Any, _
       hpvSource As Any, ByVal cbCopy As Long)
#End If
'
' Реализация функций CVx/MKx$ с помощью обращения к
' функциям Windows API -- HmemCpy или RtlMoveMemory
' (копирование заданного числа байт из одной области
' памяти в другую)
'
' ПРИМЕЧАНИЕ. Передача строковой переменной по значению
' (ByVal) означает, что в функцию HMemCpy передается
' адрес не описателя, а самой строки
'
Function CVD(x$) As Double
   HMemCpy Temp#, ByVal x$, 8
   CVD = Temp#
End Function

Function CVI(x$) As Integer
   HMemCpy Temp%, ByVal x$, 2
   CVI = Temp%
End Function

Function CVL(x$) As Long
   HMemCpy Temp&, ByVal x$, 4
   CVL = Temp&
End Function

Function CVS(x$) As Single
   HMemCpy Temp!, ByVal x$, 4
   CVS = Temp!
End Function

Function MKD$(x#)
   Dim Temp As String * 8
   HMemCpy ByVal Temp, x#, 8
   MKD$ = Temp
End Function

Function MKI$(x%)
   Dim Temp As String * 2
   HMemCpy ByVal Temp, x%, 2
   MKI$ = Temp
End Function

Function MKL$(x&)
   Dim Temp As String * 4
   HMemCpy ByVal Temp, x&, 4
   MKL$ = Temp
End Function

Function MKS$(x!)
   Dim Temp As String * 4
   HMemCpy ByVal Temp, x!, 4
   MKS$ = Temp
End Function
```

## Совет 173. Как сделать фон формы в виде палитры цветов

Хотите сделать фон своей формы в виде палитры цветов, вроде той, что Microsoft любит выдавать на экране при работе своих установочных утилит SETUP.EXE? Такой стиль раскрашивания называется градиентным заполнением и легко реализуется с помощью следующей процедуры:

```
Sub Dither(vForm As Form) Dim intLoop As Integer vForm.DrawStyle =
        vbInsideSolid vForm.DrawMode = vbCopyPen vForm.ScaleMode =
        vbPixels vForm.DrawWidth = 2 vForm.ScaleHeight = 256 For intLoop
        = 0 To 255 vForm.Line (0, intLoop)-(Screen.Width, intLoop - 1), _
        RGB(0, 0, 255 -intLoop), B Next intLoop End Sub
```

Теперь в событие Form_Activate соответствующей формы вставьте строку:

```
Dither ME
```

Цвет рисовки каждой полоски определяется с помощью функции RGB (Red-Green-Blue), основанной на смешении красного, зеленого и синего цветов. В приведенном выше примере закраска получается от черного до голубого цвета (как у Microsoft). Для черно-красного фона используйте такой вариант:

```
RGB(255 - intLoop, 0, 0).
```

А поклонникам красно-желтых цветов фирмы «1С» (рис. 1) следует применить

```
RGB(255, 255 - intLoop, 0).
```


Рис.1

## Совет 174. Помните о свойстве KeyPreview для формы

Как и многие другие элементы управления, форма имеет стандартные события для обработки нажатия клавиш — KeyDown, KeyPress и KeyUp (об особенностях использования этих процедур и кодов клавиатуры см. Совет 129 в КомпьютерПресс № 7'98, с.195). Их можно применять для управления кодами клавиш на уровне формы для всех находящихся на ней элементов управления.

Для этого следует сначала установить свойство KeyPreview формы как True. Тогда можно будет выполнять перехват всех событий KeyXX на уровне формы: в первую очередь будут выполняться процедуры формы, а уже потом — процедуры элементов управления.

Например, одна процедура Form_KeyPress может управлять режимом ввода во всех текстовых полях данной формы. А процедура Form_KeyUp может отслеживать нажатие «горячих» клавиш вне зависимости от того, где находится курсор (лучше использовать ее, а не KeyDown, так как традиционно считается, что команда выполняется при отжатии клавиши).

Если вам необходимо заблокировать обработку операций с клавишами в элементах управления (а они выполняются вслед за обработкой событий формы), установите значения KeyAscii=0 и KeyCode=0 в событиях KeyPress и KeyDown/Up соответственно.

Однако следует иметь в виду, что некоторые элементы управления (когда они находятся в фокусе) перехватывают определенные операции с клавишами в любом случае, и эти события не доходят до процедур формы. Например, командная кнопка всегда реагирует на нажатие Enter, а списки — на клавиши управления курсором.

## Совет 175. Использование экзотических «быстрых» клавиш в меню

Иногда возникает необходимость присвоить элементу меню «быструю» клавишу, отличающуюся от той, что предлагается редактором меню. Например, для команды Exit вы можете захотеть использовать такую сложную комбинацию клавиш — Ctrl+Shift+Alt+Q. Для этого введите следующий код в событие Form_Load для формы:

```
Private Sub Form_Load() mnuExit.Caption = mnuExit.Caption & vbTab &
        «Ctrl+Shift+Alt+Q» End Sub
```

Данный код добавляет текст «Ctrl+Shift+Alt+Q» к названию элемента меню mnuExit и выравнивает его по правому краю относительно других «быстрых» клавиш в меню. Далее вспомните о предыдущем совете и установите свойство KeyPreview для формы как True, а затем напишите код для события KeyUp:

```
Sub Form_KeyUp (KeyCode As Interger, Shift As Integer) If KeyCode = 81 And
        Shift = 7 Then ' операция по команде Exit End If End Sub
```

## Совет 176. Используйте свойства Default и Cancel для командных кнопок

Довольно часто на форме располагаются две кнопки, обычно связанные с ее закрытием: Ok — выполнение некоторых действий, заданных формой, и Cancel — отмена каких-либо действий. При работе с клавиатурой подобные операции традиционно выполняются с помощью клавиш Enter и Esc. Чтобы задействовать проименение этих клавиш, можно использовать свойства Default и Cancel для командных кнопок.

Если вы установите свойство Default для кнопки как True, то в любой момент работы с формой при нажатии Enter будет выполняться событие Click данной кнопки. Аналогично, если установить свойство Cancel как True, то при нажатии Esc будет выполняться событие Click. Во избежание путаницы VB автоматически следит за выполнением двух правил:

1. Свойство Default или Cancel может быть установлено как True только для одной кнопки на форме.

2. Одна кнопка может иметь одновременно только одно свойство, установленное как True, — либо Default, либо Cancel.

### Совет 177. Как определить имя накопителя CD-ROM

Если вам нужно определить имя накопителя на компакт-дисках, можете воспользоваться таким программным кодом:

```
Declare Function GetDriveType Lib «kernel32» Alias _ «GetDriveTypeA»
        (strDrive As String) As Long Const DRIVE_CDROM = 5
Public Function GetCDROMDrive() As String Dim lType As Long Dim i As
        Integer Dim tmpDrive As String Dim found As Boolean ' ' Просмотр
        по всем буквам A-Z: For i = 0 To 25 tmpDrive = Chr(65 + i) & «:\»
        ' обращение к функции Win32 API: lType = GetDriveType(tmpDrive)
        If (lType = DRIVE_CDROM) Then  ' найдет CD-ROM found = True: Exit
        For End If Next If Not found Then tmpDrive = «» GetCDROMDrive =
        tmpDrive End Function
```

### Совет 178. Как избежать ненужного обновления наборов записей

Приведенный здесь код пригодится для уменьшения влияния операции по обновлению наборов данных на ввод данных с клавиатуры. Для этого поместите на форму таймер (tmr_Timer) и установите свойство Interval как 1000 и свойство Enabled как False. Затем введите следующий код в событие txtFilter_Change текстового окна:

```
Private Sub txtFilter_Change() Timer1.Enabled = False Timer1.Enabled =
        True End Sub
```

В событии Timer вызовите подпрограмму, которая обновляет ваш набор записей:

```
Private Sub Timer1_Timer() Timer1.Enabled = False Call
        MyUpdateRecordsetRoutine End Sub
```

Теперь набор записей будет обновляться только в том случае, если вы не нажмете какую-либо клавишу в течение целой секунды. Каждый раз при нажатии клавиши будет происходить сброс таймера, а отсчет времени снова начинаться с нуля.

### Совет 179. Упрощайте программный код

Конструкции, подобные следующей

```
If MyNumber > 32 Then BooleanValue = True Else BooleanValue = False End If
```

встречаются в программах довольно часто. Но гораздо привлекательнее выглядит такой вариант:

```
BooleanValue = (MyNumber > 32)
```

### Совет 180 (очень длинный). Будьте внимательны при работе с повторно используемыми BAS-компонентами в VB и VBA

Этот совет появился в ходе подготовки материалов для нашей постоянной рубрики «Разработка приложений в среде MS Office 97». Суть вопроса заключается в том, что с точки зрения программирования работа в Visual Basic и в Office/VBA выглядит почти тождественной. Однако на самом деле между ними есть немало принципиальных различий, которые, в частности, касаются повторного использования ранее созданных процедур, написанных на VB.

Например, в статье «Разработка приложений с помощью Excel 97 и VBA. Часть 2» (КомпьютерПресс, № 12'98) мы использовали несколько процедур из своих более ранних примеров. И именно на это мы хотим обратить сейчас ваше внимание. Отметим сразу, что данная тема непосредственно связана с детальным изучением компонентной структуры VB- и VBA-приложений.

Например, по ходу разработки приложения вам нужно использовать процедуру Proc1, которая хранится в модуле Module1.bas. Внешне эта задача решается в VB (автономном средстве разработки) и в VBA (точнее, в среде VBA некоторого конкретного офисного приложения, например Word) примерно одинаково: нужно загрузить этот модуль командой Project|Add Module или File|Import File соответственно. Но в этих операциях имеются и принципиальные отличия.

#### Как это происходит в Visual Basic

При работе с VB каждый используемый в нем BAS- и FRM-модуль продолжает оставаться автономным компонентом приложения, каждый хранится в виде отдельных файлов на диске. Собственно VB-проект — это совокупность автономных файлов с программным кодом, которые объединяются в один загрузочный EXE-модуль только в момент его создания. Разумеется, сейчас речь идет только о файлах BAS и FRM, которые могут формироваться непосредственно в среде VB, и код которых помещается в загрузочный модуль. Другие компоненты приложения — OCX, DLL и прочие являются внешними и формируются отдельно.

Суть вопроса заключается в том, что создавая или корректируя BAS-модули (и FRM-файлы), вы автоматически изменяете файлы, хранящиеся на диске (обычно это делается в момент завершения работы пакета). Таким образом, если некоторые модули применяются сразу в нескольких проектах (то есть фактически являются повторно используемыми компонентами), сделанные изменения автоматически вносятся во все остальные приложения (разумеется, только в момент перекомпиляции исполняемого модуля).

Такая логика работы с исходными модулями кода (традиционная для всех систем программирования) имеет свои плюсы и минусы. Но общим выводом является необходимость уделять особое внимание именно повторно используемым компонентам, которые зачастую являются небольшими, но очень полезными вспомогательными процедурами.

Хорошим решением при работе в среде систем MS Basic для DOS было использование вспомогательных BAS-процедур в виде объектных (сейчас их часто называют статическими) LIB-библиотек, которые потом включались в состав исполняемого модуля. Остается

только сожалеть, что Microsoft почему-то упорно не хочет реализовать такой удобный и простой вариант в своем Visual Basic для Windows. Однако, начиная с версии VB4, подобные процедуры можно оформлять в виде ActiveX-серверов (DLL или EXE), к которым могут обращаться любые программы, поддерживающие технологии ActiveX (в том числе и приложения MS Office 97).

Однако речь идет о двух разных технологиях: одно из важных различий между объектными и динамическими библиотеками заключается в том, что в первом случае в состав исполняемого модуля записываются только те процедуры, на которые имеются ссылки. В варианте же DLL в исполняемый модуль вообще ничего не записывается, но при обращении к любой функции из такой библиотеки в память грузится весь файл целиком.

В то же время нужно иметь в виду то, что при использовании LIB-библиотеки производится загрузка не только кода процедуры, на которую имеется ссылка, но и всего объектного модуля, включая остальные записанные в нем процедуры. То же самое происходит и при загрузке повторно используемых компонентов в виде исходных модулей.

Мы говорим об этом потому, что хотим обратить ваше внимание на распределение процедур по модулям — это особое искусство, и о нем можно говорить отдельно. Тут есть два крайних варианта: либо поместить все вспомогательные процедуры в один модуль, либо каждую процедуру записать в отдельный модуль. Первый вариант представляется изначально неверным (слишком большая избыточность кода для конкретного приложения); второй вариант (который в свое время был классическим стилем программирования) тоже не оптимален, так как заставляет программиста работать с огромным числом файлов. Оптимальный вариант лежит где-то посередине, не говоря уже о том, что объединение взаимосвязанных процедур (например, на уровне общих данных) в один модуль бывает просто необходимо.

Кроме использования готовых процедур в виде загрузки модуля целиком, можно просто копировать код самой процедуры и переносить его из одного модуля в другой. Но этот способ представляется одним из самых неудачных (обычно его применяют начинающие программисты), так как это приводит к путанице в версиях процедур и сложностям в их поддержке (например, если обнаруживается ошибка, которую надо исправить во всех задействованных проектах). Столь же плох и вариант с копированием всего модуля (или работа с его копией под другим именем.)

## Как обстоит дело в Office 97

Тут все происходит совсем не так, как в VB.

BAS-модуль, загруженный в Office/VBA (это относится и к Word, и к Excel) командой File|Import File, автоматически теряет логическую связь с соответствующим исходным BAS-файлом, хранимым на диске, и становится сугубо внутренним компонентом данного приложения. Соответственно, все изменения, сделанные впоследствии в BAS-файле, никак не влияют на состояние уже загруженного модуля. И наоборот, коррекция кода, выполненная внутри VBA, никак не влияет на состояние исходного модуля. Если вы хотите использовать созданный (или измененный) код такого модуля, его нужно специально записать на диск командной File|Export File. Все это относится и к FRM-модулям.

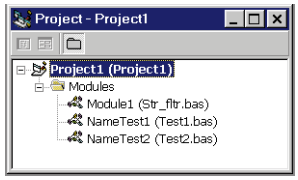

Рис. 2

Эти различия при работе в VB и VBA видны уже из названий команд ввода/вывода модулей (Load/Save и Import/Export), а также в представлении модулей в окне проектов. В случае VB (рис. 2) BAS-модуль представлен и именем файла (в скобках), и его свойством Name (Attribute VB_Name в первой строке файла). При этом если загружается BAS-модуль старой структуры (без Attribute VB_Name), то формируется стандартное имя ModuleX. При работе в VBA модуль идентифицируется только свойством Name без какого-то упоминания об исходном файле (рис. 3).

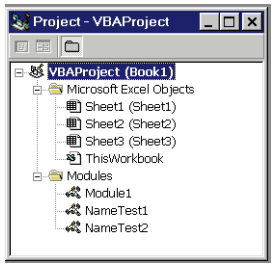

Рис. 3

Такой «внутренний» характер компонентов VBA-приложения в значительной степени изменяет технологию работы с повторно используемыми программными кодами. У программиста здесь фактически полностью развязаны руки в операциях перемещения процедур между модулями, удаления ненужных процедур и пр. При этом если работа с конкретным проектом упрощается, то технология поддержки повторно используемого инструментария существенно усложняется.

## Выводы и рекомендации

1. Квалификация программиста во многом определяется его умением использовать готовые компоненты, а также формировать и пополнять набор таких средств из числа собственных разработок. У опытного программиста до 90% программного кода конкретного приложения состоит из готовых процедур, написанных им самим в предыдущие годы.
2. При разработке VB/VBA-приложений четко разделяйте (хотя бы мысленно) повторно используемые модули и модули, которые вы собственно и пишете для работы данной программы.
3. Храните повторно используемые файлы в отдельных каталогах. Каждое приложение и его собственные (уникальные) исходные модули храните также в отдельных каталогах. Избегайте дублирования

имен файлов, даже хранимых в разных каталогах, а тем более избегайте дублирования имен процедур. На вашем диске должна быть только одна рабочая копия программного кода.

4. Если вам захотелось использовать программный код, уже реализованный для другого приложения, очевидно, имеет смысл оформить его в виде процедуры, которую можно включить в состав своих повторно используемых средств.
5. Избегайте создания процедур общего назначения в FRM-модулях (особенно в VB), кроме тривиальных случаев однозначной привязки программного кода к данной форме. Но лучше создайте BAS-модуль с тем же названием для хранения подобных процедур.
6. Тестирование и оформление (например, написание комментариев) повторно используемых процедур нужно проводить особенно тщательно. Помните, что внесение в них изменений и отладка в рамках конкретного приложения еще не гарантируют их работоспособности в других программах (ведь ошибка может проявиться при специфическом наборе входных параметров).
7. Избегайте модернизации готовых процедур, а без особой нужды тем более не меняйте их входные спецификации. При работе с библиотеками LIB и DLL это может привести к аварийным ситуациям.
8. Внимательно относитесь к распределению процедур между отдельными BAS-модулями. Старайтесь не злоупотреблять перемещением процедур между различными файлами.
9. Для надежного сохранения сделанных в проекте изменений рекомендуем при работе в среде Visual Basic установить режим автоматического сохранения всех откорректированных компонентов при запуске программы на выполнение. Для этого нужно командой Tools|Options вызвать диалоговое окно Options, открыть там вкладку Environment и в переключателе When a program starts установить позицию Save Changes. В таком режиме работы запуск программы будет выполняться несколько медленнее (скорее всего, вы этого даже не заметите), но вам будет гарантирована сохранность результатов вашего труда в случае неожиданного аварийного завершения VB при отладке приложения (что хотя и маловероятно, но все же случается). К сожалению, в среде VBA, включенной в офисные пакеты Office 97, такое автоматическое сохранение проекта не предусмотрено — за этим разработчик должен следить сам.
10. Если при работе в среде VBA вы вносите изменения в загруженный готовый модуль (например, удаляете ненужную процедуру), сразу же поменяйте его название, чтобы не путать его с исходным файлом. Аналогично, если вы меняете код процедуры, то скорректируйте ее имя и сделайте соответствующую пометку в комментарии. Храните все повторно используемые процедуры в отдельных модулях. Созданные в рамках данного приложения процедуры, которые могут пригодиться в дальнейшем, записывайте в виде отдельного файла и бережно храните.