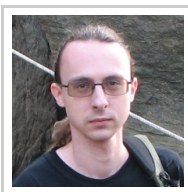
БЛОГ GUNSMOKER-A

...WHEN ALTERING ONE'S MIND BECOMES AS EASY AS PROGRAMMING A COMPUTER, WHAT DOES IT MEAN TO BE HUMAN?..

Режим чтения |
Мобильная версия

ОБО МНЕ



АЛЕКСАНДР АЛЕКСЕЕВ
ТВЕРЬ, ТВЕРСКАЯ ОБЛАСТЬ, RUSSIA

Tech geek, have social skills of a thermonuclear device.

ПРОСМОТРЕТЬ ВЕСЬ ПРОФИЛЬ
Написать письмо (e-mail)

СЛУЧАЙНЫЕ ПОСТЫ

Vista-talks
Delphi, HelpInsight, Documentation Insight и... QIP!
Спасём Google Wave!
Задачка №17

ПОДПИСКА

4 АПРЕЛЯ 2011 Г.

Что плохого в глобальных переменных?

Переменные в программе бывают локальными (это когда они объявлены в "чём-то": процедура, метод, объект и т.п.) и глобальными (это когда они объявлены в самой программе, модуле - на самом верхнем уровне без вложения во "что-то"). Параметры процедур, функций и методов также относятся к локальным переменным.

Что лучше использовать?

Стандартный совет при использовании глобальных переменных: держитесь от них подальше, используя их только тогда, когда без них не обойтись.

Почему?

Казалось бы, "ведь это действительно иногда необходимо или же просто удобней, а кроме того - быстрее в работе". Давайте посмотрим!

Передача параметров

Вот два варианта:

```
1  // Локальные переменные:
2  procedure Calc(const Matrix: TMatrix; out Result: TMat
3  begin
4    // работаем с Matrix и Result, к примеру, делаем как
5  end;
```

и:

```
1  // Глобальные переменные:
2  var
3    Matrix: TMatrix;
4    Result: TMatrix;
5
6  procedure Calc;
7  begin
8    // работаем с Matrix и Result
9  end;
```

Что лучше? Тут даже думать не надо. Попробуйте написать штук десять разных подпрограмм вроде этой `Calc` - и не запутаться при этом в параметрах! Ведь вам придётся заводить переменные, которые передаются только в `Calc`, и не перепутать их с теми, которые передаются в `Calc2`. А когда вы создаёте новую функцию `Funcenstein` - вам лучше бы создать новые переменные для неё, иначе `Funcenstein` не сможет вызвать `Calc` или `Calc2` - ведь они используют переменные с одинаковыми именами!

Код вроде второго способен написать только человек, которому "сказали сделать процедуру", а передавать параметры он *не умеет* - вот он и написал "как сумел", по старинке: глобальными переменными. Ведь это работает.

Вычисления внутри подпрограммы

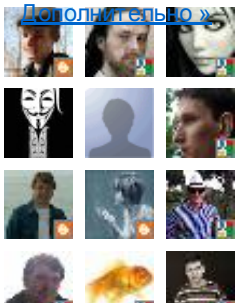
Тут тоже не так уж сложно (если подумать). Согласитесь, что код:

```
1  procedure TForm1.Button1Click(Sender: TObject);
2  var
3    X: Integer;
4    Y: Integer;
5    S: String;
6  begin
7    // Работаем с X, Y, S
8  end;
9
10 procedure TForm1.Button2Click(Sender: TObject);
11 var
12   X: Integer;
13   S: String;
14   H: String;
15 begin
16   // Работаем с X, S, H
17 end;
```

Намного лучше чем:

```
1  var
2    X: Integer;
3    Y: Integer;
4    S: String;
5    H: String;
6
7  procedure TForm1.Button1Click(Sender: TObject);
8  begin
9    // Работаем с X, Y, S
10 end;
11
12 procedure TForm1.Button2Click(Sender: TObject);
13 begin
14   // Работаем с X, S, H
15 end;
```

Второй код способен написать лишь школьник/студент, только-только начавший программировать - просто потому, что он вообще не различает эти два случая, и вставил объявление



переменных "в первое попавшееся место".

Почему первый код лучше? Даже в таком небольшом примере уже видно, что локальные переменные лучше изолированы, чем глобальные. Что это значит? Это значит, что изменения в переменной `X` в методе `Button1Click` не влияют на переменную `X` в методе `Button2Click`. Хорошо это или плохо?

Неопытный программист может сказать, что общедоступность глобальных переменных - это хорошо: "ведь это простой способ передать данные". Но если чуть подумать, то это оказывается не так уж здорово - и вот почему:

1. Масштабирование
2. Побочные эффекты
3. Проблемы инициализации

1. Масштабирование
Это просто. Когда вы используете глобальные переменные для чего бы то ни было, вы тем самым неявно предполагаете, что это "что-то" может быть **только в одном экземпляре**.

К примеру, использовав глобальные переменные для передачи данных в подпрограмму, вы подразумеваете, что эту подпрограмму можно вызвать только один раз.

Вам не удастся вызвать подпрограмму одновременно из двух потоков (ведь второй вызов испортит данные первого - потому что они хранятся в одном и том же месте: глобальных переменных). Но даже в однопоточной программе это проблема: подпрограмма не может вызвать сама себя. Конечно, если вы пишете рекурсивный код, то вы наверняка это предусмотрите и будете сохранять/восстанавливать переменные перед повторным вызовом самого себя (уф, сколько работы - вместо того, чтобы просто отказаться от глобальных переменных). Но рекурсивный вызов может происходить опосредованно и, таким образом, совершенно незаметно для вас (т.е. функция вызывает вторую функцию, которая вызывает первую).

2. Побочные эффекты
Вторая проблема - это контроль за изменениями в глобальных переменных. Дело в том, что глобальные переменные видимы отовсюду, глобально. Это удобно: ведь нет никаких ограничителей. С другой стороны, становится совершенно невозможно отследить, кто меняет данные. Неконтролируемые изменения - это первое, что обычно приходит в голову на вопрос о том, чем же плохи глобальные переменные.

ТЭГИ

7 (4)

Предположим, у вас есть функция, результат которой зависит от глобальной переменной. Вы вызываете её, вызываете - но через 10 минут функция начинает возвращать неверные результаты. Что случилось? Ведь на вход вы передаёте ей всё тот же набор параметров? Гм, кто-то поменял значение глобальной переменной... Кто это мог быть? Да кто угодно - ведь глобальная переменная доступна всем. Да чего там далеко ходить: вот простой и наглядный пример.

Лучший рецепт при проектировании подпрограмм: сделать так, чтобы результат вашей функции **зависел бы только от аргументов**. Это идеал, к которому нужно стремиться.

Это в одну сторону. Есть проблемы и если посмотреть на это наоборот. Если у вас есть подпрограмма, которая меняет состояние глобальной переменной, то это может стать для вас неожиданностью, когда вы вызываете такую подпрограмму. Вы вызываете подпрограмму, чтобы получить какие-то данные, но эта подпрограмма меняет глобальную переменную - что никак не следует из её имени или прототипа заголовка.

Тут тоже есть простое правило: если функция устанавливает значение глобальной переменной, то **это должно быть её единственной задачей**, а её название должно явно отражать этот факт (к примеру, там может быть слово `Set` или `Setup`). Если вам нужно, скажем, вычислить что-то и сохранить в глобальную переменную - нужно сделать это двумя отдельными действиями.

Ещё один совет при этом - максимально изолировать глобальную переменную: поместить её в секцию implementation модуля и объявить как можно ниже по тексту - эти действия призваны максимально сузить размер кода, который работает с переменной.

3. Проблемы инициализации
Здесь я скажу совсем кратко: очень легко проморгать момент, что для вызова чего-то вам нужно предварительно инициализировать какую-то глобальную переменную. Откуда следуют всевозможные проблемы перекрытия жизненных циклов.

Итак, у глобальных переменных есть некоторые проблемы. Вы можете уменьшить их отрицательный эффект, но настоящее решение заключается в избегании глобальных переменных.

Тестирование

Вы можете подумать, что это может не иметь к вам никакого

отношения - ведь вы пишете очень простую программу, где "даже нет подпрограмм". Почему бы не сделать всё глобальным?

Ну, я всегда говорю, что программы очень быстро развиваются, и если сегодня у вас одна ситуация, то это не значит, что завтра она будет такой же. Но я не буду этого говорить (а скажу это здесь). Вместо этого, я рассмотрю случай, про который вы не подумали - и он не включает в себя эволюцию/изменение программы.

Это - модульное тестирование. Положим, вы хотите убедиться и иметь гарантию в будущем, что ваш, ну скажем, код преобразования матрицы работает правильно. Как вы это делаете? Вы пишете тест, где подаёте вашему коду заранее просчитанный результат и сравниваете результат работы кода с уже готовым ответом. Вызовите ваш код несколько раз (по разу - на граничный случай, и один раз - на случайный) - и у вас будет хорошая уверенность, что вы написали его правильно.

В чём же здесь затык? Посмотрите, положим у вас есть код:

```
1  unit MatrixCalculations;
2
3  interface
4
5  uses
6    MyTypes;
7
8  function DoSomething(const ASource: TMatrix): TMatrix;
9
10 implementation
11
12 ...
```

Как бы вы тестировали этот код? "Ну, я создам новое приложение - это будет мой тест, подключу к нему этот модуль, а потом просто передам массив `ASource` и проверю результат. Это же просто, да?".

Да. Только вот это не будет работать. Почему? Да потому что `DoSomething` требует для работы установки коэффициентов в глобальных переменных - быть может в каком-то другом модуле. Хорошо ещё, если это явно видно в тексте `DoSomething` в этом же модуле. А если она вызывает другую функцию с таким требованием? Ой. Теперь элементарная задача по проверке одной функции превращается в страшного монстра по распутыванию зависимостей вызовов.

Если говорить научно, то глобальные переменные увеличивают число зависимостей между компонентами. Модульный тест - это просто один из примеров на практике, где этот момент хорошо виден. Искусство написания программ заключается в управлении

сложностью - т.е. "делаем вещи максимально простыми". Рост зависимостей этой задаче не способствует.

Мы уже видели этот принцип: нужно стараться делать функции такими, чтобы их результат зависел бы только от их аргументов. Это здорово увеличивает предсказуемость системы и уменьшает число взаимосвязей в ней.

Синглтоны

Очень часто доводы за глобальные переменные встречаются у начинающих разрабочиков игр. Ведь глобальные переменные - это "быстро и грязно". А именно это им в начале и нужно. К примеру, часто можно услышать слова вроде таких: "я храню в глобальных переменных системные установки, игровой мир и профиль игрока".

Почему это хранится в глобальных переменных? Потому что все эти вещи существуют в единственном экземпляре.

Тут настаёт время познакомится с понятием (шаблоном программирования) синглтон (singleton) - его также называют "шаблон Одиночка".

Суть подхода в том, что он гарантирует, что у класса есть только один экземпляр, и предоставляет к нему глобальную точку доступа. Иными словами, синглтон - это класс, у которого может существовать лишь один объект этого класса, и доступен он из единственной точки кода. Хотя синглтон - это не аналог глобальных переменных (синглтон не обязан быть глобальным), но всё же: в чём достоинство синглтонов по сравнению с глобальными переменными?

- Нет проблем с именами (конфликты имён). Два разных синглтона не пересекаются и не конфликтуют.
- Нет проблем с инициализацией и перекрытием времени жизни. Синглтон - он всегда один и доступ к своей инициализации контролируется им же.
- Нет проблем с многопоточным взаимодействием (при дополнительных усилиях).
- Нужно поддерживать только интерфейс. При глобальной переменной нужно отслеживать весь код по всей программе, который с ней работает. Синглтон инкапсулирует часть работы в себя, предоставляя наружу только ограниченный интерфейс.

К реализации синглтона есть два основных подхода:

- Реализация на классовых методах. Это самый примитивный

случай - тут уникальность гарантируется компилятором. Понятно, что у классовых методов есть очевидные ограничения.
- Наследованием класса от шаблонного. Это уже сложнее. Наиболее удачная реализация, что я видел - это реализация от Ins-a.

Итак, в нашем примере с начинающим разработчиком игр: настройки программы - это должен быть синглтон, а не разрознённый ворох глобальных переменных. Т.е. было:

```
var
  FileName: String;
begin
  ...
  // SaveFolder - это настройка. Скажем, папка для сей
  Save(SaveFolder + FileName);
end;
```

Стало:

```
var
  FileName: String;
begin
  ...
  // Settigns - это синглтон, хранящий настройки прогр
  Save(Settings.SaveFolder + FileName);
end;
```

Здесь `Settings` - это функция, реализованная так (в предположении, что вы используете шаблон от Ins-a):

```
unit AppSettings;

interface

uses
  SingletonTemplate; // модуль, содержащий TSingleton

type
  TSettings = class(TSingleton)
  protected
    constructor Create; override;
  public
    destructor Destroy; override;
  private
    FSaveFolder: String;
    // ...
  public
    property SaveFolder: String read FSaveFolder;
    // ...и другие настройки программы

    // Можно добавить и методы загрузки/сохранения нас
    // procedure Save;
    // procedure Load;
    // А можно и не добавлять - тогда вы сделаете это

    // ...и другие методы TSettings
  end;

function Settings: TSettings;

```

```
implementation

function Settings: TSettings;
begin
  Result := TSettings.GetInstance;
end;

constructor TSettings.Create;
begin
  inherited Create;
  // Сюда можно поместить загрузку настроек программы
end;

destructor TSettings.Destroy;
begin
  // Сюда можно поместить сохранение настроек программ
  inherited Destroy;
end;

end.
```

Сделаю ещё замечание - для сильно "нелюбителей" объектного подхода: пусть вы забили на синглтоны и сделали профиль игрока просто глобальными переменными. А потом... потом в вашей игре появляется сплит-скрин (split-screen). Или мультиплейер. Ой. Теперь у вас может быть два или даже больше профилей. Да, только первый профиль содержит полный набор данных, второй и последующих ограничены: там только имя, настройка клавиатуры (для сплит-скрина) и вещи вроде сетевых идентификаторов (для мультиплеера) - тем не менее, это профиль. А весь ваш код, работающий с настройками игрока, теперь нужно переписать. И простой Search&Replace по коду, скорее всего, не поможет. Вам придётся пройтись по всей написанной программе и руками отследить все обращения к профилю.

А если бы вы писали на объектах? Всё просто: профиль игрока был синглтоном - стал обычным объектом. Вы можете оставить синглтон главного игрока. А все остальные заносятся в массив профилей. Весь ваш код был на объектах и работал, скажем, со синглтоном Profile. Тогда вы просто делаете Profile свойством объекта. К примеру, было:

```
type
  TPlayer = class(...)
  ...
    procedure Move;
  ...
  end;

...

procedure TPlayer.Move;
begin
  // Здесь: Profile - это глобальный синглтон
  case KeyPressed of
    Profile.KeyLeft:
      MoveLeft;
    Profile.KeyRight:
```

```
      MoveRight;
    Profile.KeyFire:
      Fire;
  end;
end;
```

Стало:

```
type
  TPlayer = class(...)
  private
    FProfile: TProfile;
  public
  ...
    procedure Move;
  ...
    constructor Create(const AProfile: TProfile);
    property Profile: TProfile read FProfile;
  end;

...

constructor TPlayer.Create(const AProfile: TProfile);
begin
  ...
  FProfile := AProfile;
end;

procedure TPlayer.Move;
begin
  // Здесь: Profile - это свойство TPlayer
  case KeyPressed of
    Profile.KeyLeft:
      MoveLeft;
    Profile.KeyRight:
      MoveRight;
    Profile.KeyFire:
      Fire;
  end;
end;
```

Волшебным образом уже написанный код (`Move` и другие методы объекта "игрок") совершенно не изменился! Это снова к вопросу гибкости кода.

Глобальные константы

Ещё один случай, когда глобальные переменные обычно оправданы - это, так называемые, переменные-константы. К примеру, всем очевидно, что число Пи должно быть глобальной константой. Вот и переменная, которая инициализируется при запуске программы и остаётся неизменной на протяжении всей жизни программы, вполне может быть сделана глобальной (раз уж основная претензия к глобальным переменным - неконтролируемые изменения). А чтобы не было соблазна её поменять, можно сделать к ней непрямой доступ. Например, было:

```
var
  X: String;
```

```
3
4   implementation
5
6   ...
7
8   initialization
9     X := ...;
10  end.
```

стало:

```
1   function X: String;
2
3   implementation
4
5   ...
6
7   var
8     fX: String;
9
10  function X: String;
11  begin
12    if fX = '' then
13      fX := ...;
14    Result := fX;
15  end;
16
17  end.
```

В случае, если это нечто большее чем простое значение, возможно, будет предпочтительнее использовать синглтон.

Историческая справка

Давным-давно в программах не было подпрограмм, а все переменные были, очевидным образом, глобальными. Позднее в языках программирования начали появляться средства структурирования кода, и среди них - подпрограммы. Но в них пока нельзя было объявлять переменных. Поэтому все переменные всё ещё были глобальными. И лишь потом, наконец, появились локальные переменные, а глобальные переменные стали подвергаться преследованию.

Замаскированные глобальные переменные

Конечно же, когда глобальные переменные объявили злом - появилось что-то, что называется по-другому, но, на самом деле, является замаскированной глобальной переменной. К примеру, уже рассмотренный выше синглтон можно считать глобальной переменной с доступом только на чтение (к примеру, через функцию-акцессор), чей код инициализации скрыт. Ещё пример? Ну, скажем, переменные класса (**НЕ** объекта) - их также называют классовыми переменные. Это тоже глобальные переменными, даже хотя они таковыми не выглядят на первый взгляд.

Суть всех этих телодвижений в том, что в программе часто бывают данные, глобальные по своей природе - но это вовсе не значит, что к ним нужно предоставлять бесконтрольный доступ (читай: использовать глобальные переменные). Вот и применяются разные другие способы. Все они различны, но служат одной цели: изолировать и ограничить - и минимизировать, тем самым, отрицательные эффекты глобальных переменных.

Если трактовать заголовок этой секции немного в другом направлении, то можно получить и такой смысл: локальные переменные, притворяющиеся глобальными. Это когда весь ваш (основной) код программы написан в одной большой-большой процедуре - так называемая "волшебная кнопка". В этом случае, хотя формально все ваши переменные - локальны, но работают они как глобальные, со всеми вытекающими из этого минусами. Вариантом этого является случай, когда весь код программы заключён в единственном объекте - т.н. объект-Бог.

В обоих случаях решение заключается в реструктуризации кода: выделения подпрограмм, разбивки на классы и т.п.

Как Delphi учит нас плохому

Что я действительно ненавижу в Delphi (это полусерьёзно) - так это этот код:

```
1 var
2   Form1: TForm1;
```

Из-за него каждый начинающий считает, что делать так - это нормально:

```
1 Form1.Edit1.Text := 'gg';
```

Но что не так с переменной? Ведь главная форма всегда одна? Верно. И для этого у нас есть `Application.MainForm`. И если для главной формы у вас был слабенький аргумент (это же глобальный объект!), то для всех прочих форм у вас нет даже такого аргумента. Более того, этих форм может быть и много. Поэтому, первое, что необходимо сделать в программе - удалить все глобальные переменные форм, оставив, быть может, только главную. Нужна ссылка на форму? Объяви локально. Несколько форм? Используй `Screen.Forms` или веди учёт форм в списке (пример: многооконный редактор). Примечание: кстати, если в приложении используется MDI, то такой список уже есть - это `TForm.MDIChildren`.

Иногда код вида `Form1.Edit1.Text := 'gg';` стоит даже в методах

самой формы! Уж это-то точно не поддаётся ни оправданию, ни объяснению.

Консольные программы и им подобные: быть или не быть?

Ещё один пример, где часто наблюдаются глобальные переменные - консольные приложения. Этот тип приложений часто представляет собой подход "пишем код прямо в begin/end и всё делаем глобально" - со всеми вытекающими: ни потестировать код, ни запустить в двух экземплярах и так далее.

Действовать надо так же, как мы обсуждали выше - реструктуризацией кода: введением подпрограмм, созданием классов.

Итого

Изолирование - это одна из ключевых концепций в программировании. Это - один из инструментов уменьшения и контроля сложности программы. Вы всегда должны стараться делать код максимально независящим от другого, а имеющиеся зависимости должны быть выражены как можно более очевидно. Это - один из необходимых компонентов в "сделать код максимально простым". В ООП это называется инкапсуляцией и обычно ассоциируется с private/protected/public и скрытием данных класса. Использование глобальных переменных - это равносильно использованию объектов с одной только секцией public.

Суммируя сказанное: если у вас есть два решения, одно - с глобальной переменной, другое - без, то предпочитайте вариант без глобальной переменной. Иными словами, если можно обойтись без глобальной переменной - обходитесь.

Пытаясь обосновать использование глобальных переменных, часто говорят, что они мол, удобны. Это иллюзорный и эгоистический аргумент, поскольку сопровождение программы обычно продолжается дольше, чем первоначальная разработка. Иными словами: глобальные переменные - это "быстро и грязно". И если вы ввели глобальную переменную по соображениям "быстрей написать", то в дальнейшем этот код надо переписать (улучшить).

В общем, не знаю, насколько удачной получилась эта статья. Мне кажется, что глобальные переменные - это один из тех моментов, пока человек сам на грабли не наступит, любые слова

- мимо. Чем-то это похоже на попытки объяснить плюсы ООП человеку, не писавшему программ больше небольших.

Читать далее.

P.S. Эта статья говорит про то, как правильно делать. Понятно, что не всегда есть возможность делать правильно. Не всегда это бывает и нужно. Если надо по быстрому набросать прототип, написать код для проверки гипотезы, либо же код совсем простой и пустяковый - используйте глобальные переменные, вас за это не съедят. В общем, смотреть надо по ситуации. Заранее говорить, что глобальные переменные - зло, в абстрактном вакууме не имеет большого смысла. А когда так говорят - имеют ввиду вполне конкретные ситуации, которых просто большинство: средние и большие программы с прицелом на дальнейшее сопровождение. Подробнее про эту мысль: программирование - это искусство.

P.P.S. Замечание для тех, кто не умеет вычленять контекст: это статья говорит про мир Windows, Delphi и типичных программ, которые пишутся на Delphi. Я полагаю вас достаточно сообразительными, чтобы "не проецировать на микроконтроллеры речь пиджака о преимуществах XML".

-= АЛЕКСАНДР АЛЕКСЕЕВ =- 0:03 ТЭГИ DELPHI, НАЧИНАЮЩИМ, НЕ ДЕЛАЙ ТАК, СТАТЬЯ

РОДСТВЕННЫЕ ПОСТЫ

Click to load related posts list

37 КОММЕНТАРИЙ(ЕВ):

Usoft комментирует...

я уж думал вы расскажете, про кучу и про то как инициализируются локальные и глобальные переменные, и к чему это ведет

а тут.. неужели и правда кому-то интересно что трава зеленая а вода водянистая? или вы для школьников посты пишете?

4 АПРЕЛЯ 2011 Г., 10:38

Kylt комментирует...

Я так понимаю, очередной пост, что бы кидаться ссылками :)

4 АПРЕЛЯ 2011 Г., 13:54

GunSmoker комментирует...

Ага. "По просьбам трудящихся".

4 АПРЕЛЯ 2011 Г., 13:56

Анонимный комментирует...

Спасибо, понял то что ты хотел до нести...

4 АПРЕЛЯ 2011 Г., 14:52

Анонимный комментирует...

В чем же разница глобальных переменных от глобальных объектов?
Глобальный объект - частный случай глобальной переменной.
И очень часто есть в программе несколько глобальных объектов - может настройки или справочники общие для всех.

4 АПРЕЛЯ 2011 Г., 16:08

GunSmoker комментирует...

Если глобальный объект хранится в переменной - разницы нет, потому что это и есть глобальная переменная.

Но глобальный объект не обязан быть глобальной переменной. Его можно (и часто - нужно) сделать синглтоном.

4 АПРЕЛЯ 2011 Г., 17:24

СССР комментирует...

>>неужели и правда кому-то интересно что трава >>зеленая а вода водянистая?
Представьте себе большая часть народа об этом не знает!
Правда и не интересуется :(

5 АПРЕЛЯ 2011 Г., 0:56

Всеволод Леонов комментирует...

Александр (aka GunSmoker) как всегда продемонстрировал

глубину и системность в изложении материала.

Сложно переоценить данный пост. Пока на запрос в поисковике "Delphi глобальные переменные" будут выдаваться рекомендации, где и как это "зло" нужно декларировать без обсуждения недостатков/достоинств/необходимости - мы будем сталкиваться с отраслевой проблемой создания несопровождаемого кода.

Возможно, у программиста с профильным образованием на подобные вопросы уже есть свои ответы, но хочу поделиться своим опытом. Для "научного" программирования (а Delphi здесь - идеальный инструмент) весьма характерно, когда рабочие массивы объявляются глобально, а большинство уже существующих программ (выросших из Фортран и Си-кодирования) вначале имеют длинное "полотенце" глобальных структур. И потом возникает задача "прикрутить" цивильный интерфейс на вполне пригодной глобально-алгоритмической реализации. И вот тут нужно а) показать грабельки б) дать рекомендации, с чем Александр справился как никто другой до этого (см. результаты поисковиков).
Что можно сказать - любимый писатель! :)

5 АПРЕЛЯ 2011 Г., 10:34

GunSmoker комментирует...

Пример реального кода, страдающего от глобальных переменных и вот этого.

5 АПРЕЛЯ 2011 Г., 12:25

Анонимный комментирует...

По поводу глобальных объектов и ссылок на него.

Даже, если мы глобальный объект сделали руками или синглтоном (в новых версиях :-)), это не меняет идею того, что мы имеем вызов глобального объекта и прошиваем работу с ГЛОБАЛЬНЫМ объектом красной ниткой по всей программе.
Так говорить ПЛОХО об определении ниже некорректно, если вам нужен глобальный и единственный объект. Только пользоваться им нужно корректно, как и любым другим объектом.
var Form1: TForm1;

Так что же плохо в глобальных переменных? :-)

5 АПРЕЛЯ 2011 Г., 16:06

Николай Зверев комментирует...

Ещё одну ссылку в копилку, спасибо)

>> я уж думал вы расскажете, про кучу...

Присоединяюсь к хотелке: хочется в доступной форме почитать ("покидать ссылкой") о том, чем отличается размещение данных в стеке от размещения в куче, а также о "старых" паскалевских объектах (что-то типа object vs class)

5 АПРЕЛЯ 2011 Г., 23:45

MrShoor комментирует...

Имхо вообще var в секции interface нужно запретить. Придумать только способ объявления переменных-функций для динамического подключения длл (например словом varproc).

Жаль что CG врядли когда-нибудь так сделает. Слишком много народу "взвоет", да и сами они не "безгрешные".

7 АПРЕЛЯ 2011 Г., 2:18

Анонимный комментирует...

Вот еще вопрос.
Есть глобальная переменная - полный путь к каталогу программы.
Что делать - обернуть ее в класс? или сделать глобальной переменой - например gProgramPathFull?

Я использую второй путь. Что делать?
Кто виноват не рассматриваем. :-)

8 АПРЕЛЯ 2011 Г., 14:27

GunSmoker комментирует...

Если это вопрос ко мне, то я бы сделал её функцией (которая может кэшировать значение в глобальной переменной, находящейся в implementation) - её незачем делать глобальной переменной, с возможностью изменения: ведь это, по сути, константа.

С другой стороны, большого зла в этом тоже нет - ведь наврядли кто-то будет её трогать (кроме чтения). Повторюсь, глобальные переменные не плохи сами по себе. Плохо, когда ими злоупотребляют.

Если это вопрос вообще, то лучше задать его на форуме.

8 АПРЕЛЯ 2011 Г., 14:49

Aleksey Timohin комментирует...

А меня больше всего зацепила часть про тестирование. Забавно, но написание юнит-тестов, действительно помогает увидеть баги, там где раньше всё казалось идеальным. А особенно в проектировании.

14 АПРЕЛЯ 2011 Г., 11:20

Анонимный комментирует...

А у меня уже несколько лет в проектах только одна глобальная переменная:

```
unit vars;

interface

type
TSystemVars = record
// здесь то, что надо
end;

var
SystemVars: TSystemVars;

implementation

end.
```

16 АПРЕЛЯ 2011 Г., 2:01

Nofate комментирует...

Использовать синглетоны в дельфе - это замечательно, но злоупотреблять ими тоже не стоит.
Во-первых, нужно стараться не допускать связей между синглетонами разных типов (классическая проблема с порядком уничтожения синглетонов).
Во-вторых синглетоны могут не меньше глобальных переменных усложнить модульное тестирование, т.к. сохраняют свое состояние между отдельными тестами.

21 АПРЕЛЯ 2011 Г., 0:17

GunSmoker комментирует...

>>> Использовать синглетоны в дельфе - это замечательно, но злоупотреблять ими тоже не стоит.

Мне кажется, что какой-бы подход мы не предложили - всегда найдутся люди, которые его используют неправильно.

Что я имею в виду: вот взять, хотя бы, ООП. Вот, придумали его, сказали, что это круто... Но находятся же люди, которые пишут на ООП в стиле процедурного программирования. Т.е. ООП у них - для галочки. Формально они-то его используют, но в реальности у них получается процедурный код со всеми вытекающими...

Так же и с вышесказанным: всегда найдутся люди, которые будут использовать указанные средства, ну скажем, "не согласно их духу", будут злоупотреблять ими.

28 АПРЕЛЯ 2011 Г., 6:59

Анонимный комментирует...

Глобальные переменные часто являются чуть ли не проблемой №1 при переходе к многопоточности. Вот написал ты прогу, нормально написал, переменными глобальными сильно не злоупотреблял, но где-то использовал. Они даже могут быть более-менее грамотно сделаны: закрыты в implementation модуля и т.п.

А потом... ты решил, что было бы неплохо прикрутить к программе многопточность.

Берёшь и прикручиваешь. И вот тогда у тебя начинают вылезать странные проблемы: зависания, access violation на ровном месте, утечки памяти, вылеты, да и прочие страшные проблемы. А дело оказывается в том, что глобальные переменные одновременно меняются из разных потоков. Нет синхронизации - откуда и лезет куча проблем.

А ведь отследить все обращения к глобальным переменным, чтобы навесить на них синхронизацию, может быть не такой уж простой задачей.

А вот если бы использовались локальные переменные, поля/свойства объекта - не было бы никаких проблем. Т.к. у каждого бы потока была своя локальная копия. Лучшая синхронизация данных - это та, которую не нужно делать!

Понятно, что это не гарантия (на один объект по указателям может ссылаться два потока), но довольно типичный пример. ИМХО, конечно.

16 ИЮНЯ 2011 Г., 2:54

Анонимный комментирует...

А что не верного в том чтобы писать:

Form1.Edit1.Text := 'Строка';

Разве нельзя обращаться через форму к компонентам и их свойствам?

Тогда может лучше создавать метод в классе формы и в реализации метода можно сразу писать:

Edit1.Text := 'Строка';

Или в классе формы нужно создавать свойства для обращения к компонентам на форме?

Пожалуйста разъясните. Спасибо

3 АВГУСТА 2011 Г., 20:33

GunSmoker комментирует...

Тут два момента.

Самый очевидный (и именно про него шла речь в первую очередь):

```
procedure TForm1.Button1Click(Sender: TObject);
begin
Form1.Edit1.Text := 'Строка';
end;
```

Надеюсь, этот случай очевиден?

Должно быть:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
Edit1.Text := 'Строка';
end;
```

Потому что в противном случае мы манипулируем сами собой через стороннего посредника (глобальную переменную). Во втором (правильном) коде - мы манипулируем собой напрямую.

Создавать или нет методы доступа - это отдельный

вопрос, к глобальным переменным отношения не имеющий.

Второе - чуть сложнее:

```
procedure TForm2.Button1Click(Sender: TObject);
begin
Form1.Edit1.Text := 'Строка';
end;
```

Итак, что с этим не так?

В идеале код должен строиться по принципу чёрного ящика.

Грубо говоря, если вам надо зажечь свет, вы говорите выключателю: "включить свет". Вы не говорите: "замкнуть линию 1", потому что хрен его знает, как там подводка сделана. Может там две линии разомкнуты. А может там вообще по ИК-сигналу включается.

Так же и здесь. Форма - это самодостаточный объект. Примерно как выключатель.

Сказать Form1.Edit1.Text := 'Строка' - это аналог "замкни линию 1".

Сказать Form1.ChangeText('Строка') - это аналог "включи свет".

Иными словами, внешний код не должен манипулировать объектами формы. Содержание формы - это забота исключительно самой формы. Форма должна предоставлять наружу методы по манипулированию собой.

Опять же, если вам нужно будет поменять интерфейс (скажем, меню заменить на Ribbon) - то с правильным подходом вам нужно будет изменить только форму, вам не нужно будет перелопачивать весь код, который с ней работает - потому что этот код будет вызывать методы доступа формы, прототип которых не изменился - изменилось лишь их содержание, но про это знает только сама форма.

В некоторых случаях (когда код делает что-то, не подразумевающее однозначно показ формы) имеет смысл вообще заменить форму на чистый интерфейс.

3 АВГУСТА 2011 Г., 20:58

GunSmoker комментирует...

Надо помнить, что единственная причина, почему поля компонентов и обработчики событий доступны внешнему коду - это помещение их в секцию published, что необходимо для работы потоковой системы загрузки формы (см., к примеру, в частности - методы и поля).

С моей личной точки зрения, это не совсем удачный дизайн. Было бы лучше, если бы published секция имела бы видимость секции private или protected.

В общем, об этом нужно помнить. Что доступность методов и полей делается для работы сериализации, а не для того, чтобы вы из внешнего кода к форме обращались.

3 АВГУСТА 2011 Г., 21:02

GunSmoker комментирует...

А у меня уже несколько лет в проектах только одна глобальная переменная:

```
type TSystemVars = record
// здесь то, что надо
end;

var SystemVars: TSystemVars;
```

Кстати, хотя вот это на первый взгляд ничем не отличается от просто глобальных переменных - оно всё же уже существенно лучше. Потому что код будет написан как *SystemVars.ЧтоТоТам*.

Это значит, что при желании мы легко заменим глобальный SystemVars на локально передаваемый параметр **не меняя кода**!

21 СЕНТЯБРЯ 2011 Г., 19:55

Анонимный комментирует...

Довольно неплохо, молодей

12 АВГУСТА 2012 Г., 11:40

dondublon комментирует...

Хороший обзор.
Даже удивительно, как много говнокодеров не понимает

таких простых вещей.

Следующая ступень дао - понимание того, что переменные вообще зло, не только глобальные :)

23 ОКТЯБРЯ 2012 Г., 13:26

Mortarez комментирует...

Я так понимаю, к числу исключений можно отнести и переменные больших размеров, чтобы переполнения стека не было. Хоть и не хочется объявлять глобальную переменную, которая не будет использоваться нигде, кроме как в одной-единственной функции.

7 ИЮЛЯ 2013 Г., 21:06

Александр Алексеев комментирует...

Достаточно выделять память в куче, а не на стеке (читай: использовать указатели/ссылочные типы данных).

8 ИЮЛЯ 2013 Г., 18:37

Илья Николаевич комментирует...

годная статья.

14 ОКТЯБРЯ 2013 Г., 11:03

Анонимный комментирует...

Александр, дико извиняюсь, но проясните, пожалуйста, этот момент начинающему:
"Потому что в противном случае мы манипулируем сами собой через стороннего посредника (глобальную переменную). Во втором (правильном) коде - мы манипулируем собой напрямую."

Ведь внутри TForm1 обращение вида: Form1.Edit1.Text и Edit1.Text - это ведь обращение к одному и тому же свойству объекта, тогда чем плоха приставка Form1? Такое обращение дольше обрабатывается (в ассемблере практически ничего не понимаю, но в окне "Entire CPU" видны явные отличия при вызове первого и второго варианта)? Ваши статьи и так разжёваны больше некуда, но тут просьба объяснить для совсем начинающих. Заранее спасибо, если найдёте минуту на разъяснения.

19 ДЕКАБРЯ 2013 Г., 8:46

Анонимный комментирует...

Кстати, второй пример: "Иными словами, внешний код не

должен манипулировать объектами формы. Содержание формы - это забота исключительно самой формы. Форма должна предоставлять наружу методы по манипулированию собой."- как раз очень доходчивый, это, я так понимаю, и есть пример инкапсуляции в чистом виде. Так?

19 ДЕКАБРЯ 2013 Г., 8:58

Александр Алексеев комментирует...

>>> Ведь внутри TForm1 обращение вида: Form1.Edit1.Text и Edit1.Text - это ведь обращение к одному и тому же свойству объекта, тогда чем плоха приставка Form1?

Подумайте вот о чём: что будет, если мы создадим две формы одного класса? У нас не может быть две переменные с одним и тем же именем (Form1). Значит, одна из форм будет хранится в другой переменной, а Form1 будет указывать не на неё (или вовсе не будет существовать). Тогда обращение Form1.Edit1 может: а). не скомпилироваться, б). скомпилироваться, но обратиться не к той форме, в). скомпилироваться, но вылететь с Access Violation во время выполнения.

Обращаться к самому себе через посредника - это всё равно что отправлять письмо по почте своему соседу по лестничной клетке. Конечно, это допустимо, но несколько странно. И тебе нужно быть уверенным, что ты не напутал в адресе, а то письмо не дойдёт. Насколько проще просто постучать в соседнюю дверь.

19 ДЕКАБРЯ 2013 Г., 11:14

Александр Алексеев комментирует...

>>> как раз очень доходчивый, это, я так понимаю, и есть пример инкапсуляции в чистом виде

Да.

19 ДЕКАБРЯ 2013 Г., 11:15

Александр Алексеев комментирует...

В идеале, форму лучше всего вообще скрыть за интерфейсом, но для простых форм это будет уж слишком много писанины. Так что, наверное, это имеет смысл только для сложных форм, а простые обойдутся грамотно составленной public-секцией с методами.

19 ДЕКАБРЯ 2013 Г., 11:19

Анонимный комментирует...

"Конечно, это допустимо, но несколько странно" - другими словами вариант Form1.Edit1.Text использовать можно, но не нужно, т.к. правильнее использовать приставку типа Form1. только для обращения к внешним для вызывающего юнита формам, а к собственным объектам нужно обращаться напрямую (т.е. верным будет обращение через Edit1.Text). Я правильно понял все аналогии?

19 ДЕКАБРЯ 2013 Г., 11:23

Анонимный комментирует...

Возможно, вопрос не совсем в тему, но есть ли разница где размещать функции, которые затем будут вызываться из других юнитов?

Лучше определять глобальные функции так -

```
type
TForm1 = class(TForm)
...
public
function GlobalOne(): Boolean;
...
end;
```

и вызывать их затем так -

```
Answer := Form1.GlobalOne();
```

или правильнее определять так -

```
type
TForm1 = class(TForm)
...
end;

function GlobalTwo(): Boolean;
```

и вызывать их затем как -

```
Answer := GlobalTwo();
```

Или в этих способах нет разницы, в отличии от глобальных переменных?

20 ДЕКАБРЯ 2013 Г., 9:02

Александр Алексеев комментирует...

>>> Я правильно понял

Да.

>>> использовать можно, но не нужно

Тут вся статья, про "можно, но не нужно". Надо понимать, что любое решение работать будет - в том смысле, что не приведёт к неверной работе, вылету или т.п. И его даже **иногда** имеет смысл применять - например, при прототипировании (быстром создании предварительного варианта кода для тестирования/проверки).

>>> есть ли разница где размещать функции, которые затем будут вызываться из других юнитов?

Конечно есть, принцип тут ровно такой же: если, при прочих равных, есть возможность не вводить промежуточное звено (имя переменной) - его нужно не вводить. Иными словами, код по управлению формой (в общем случае - любым объектом) должен быть оформлен как метод этого объекта, а не как отдельно стоящая процедура.

Это не единственное применимое решение. В зависимости от контекста, в некоторых случаях разумнее выделять такие методы в обработчики событий, отдельные объекты, либо вводиться не наследованием, а инкапсуляцией. Я бы советовал почитать книжку.

20 ДЕКАБРЯ 2013 Г., 19:58

Анонимный комментирует...

Ну, я думал, речь пойдёт про ГлОбАлЬнЫе КОНСТАНТЫ, которые не меняются - а вы начали воду лить. Так уж совсем не интересно.

12 ЯНВАРЯ 2014 Г., 20:17

ОТПРАВИТЬ КОММЕНТАРИЙ

Можно использовать некоторые HTML-теги, например:

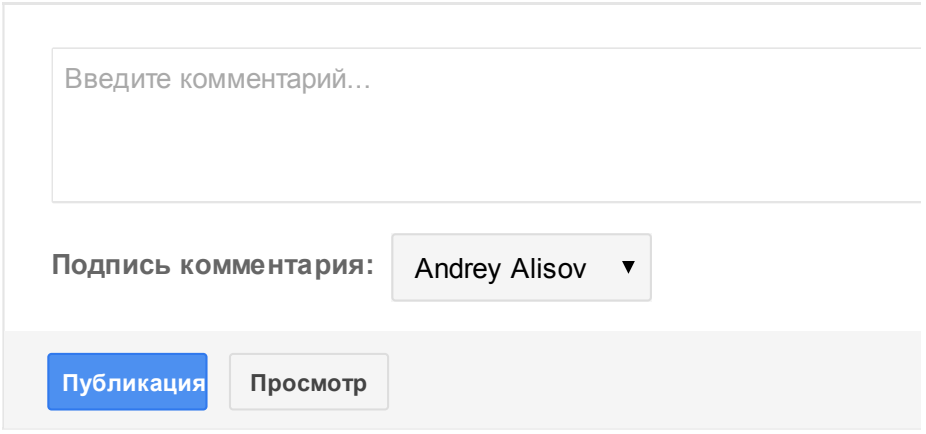
Жирный
Курсив
Ссылка

Вам необязательно регистрироваться для комментирования - для этого просто выберите из списка "Анонимный" (для анонимного комментария) или "Имя/URL" (для указания вашего имени и (опционально) ссылки на сайт). Все прочие варианты

потребуют от вас входа в вашу учётку (поддерживается OpenID).

Пожалуйста, по возможности **используйте "Имя/URL" вместо "Анонимный"**. URL можно просто не указывать.

Ваше сообщение может быть помечено как спам спам-фильтром - не волнуйтесь, оно появится после проверки администратором.



ССЫЛКИ

Создать ссылку

Следующее Главная страница Предыдущее

NEVER SEND A HUMAN TO DO A MACHINE'S JOB