**Тема №14. Отладка программ - DEBUGGER (отладчик)**

Цель: изучить технологию отладки программ.

План:

1. Отладка программ - DEBUGGER (отладчик). Возможности.
2. Критические ошибки.
3. Понятие исключительной стуации.
4. Принудительное создание исключительной ситуации.
5. Обработка исключений.

**1 Отладка программ - DEBUGGER (отладчик). Возможности**

**Отладка** — это процесс поиска и устранения ошибок. Ошибки в программе разделяют на две группы: синтаксические (ошибки в тексте) и алгоритмические. Синтаксические ошибки — наиболее легко устраняемые. Алгоритмические ошибки обнаружить труднее. Этап отладки можно считать законченным, если программа правильно работает на одном-двух наборах входных данных.

**Отла́дка** — этап разработки компьютерной программы, на котором обнаруживают, локализуют и устраняют ошибки. Чтобы понять, где возникла ошибка, приходится:

* узнавать текущие значения переменных;
* и выяснять, по какому пути выполнялась программа.

Существуют две взаимодополняющие технологии отладки.

* Использование отладчиков — программ, которые включают в себя пользовательский интерфейс для пошагового выполнения программы: оператор за оператором, функция за функцией, с остановками на некоторых строках исходного кода или при достижении определённого условия.
* Вывод текущего состояния программы с помощью расположенных в критических точках программы операторов вывода — на экран, принтер или в файл. Вывод отладочных сведений в файл называется журналированием.

**Отла́дчик** (debugger) является модулем среды разработки или отдельным приложением, предназначенным для поиска ошибок в программе. Отладчик позволяет выполнять пошаговую трассировку, отслеживать, устанавливать или изменять значения переменных в процессе выполнения программы, устанавливать и удалять контрольные точки или условия остановки и т. д.

**2 Критические ошибки**

EAbort-«Молчаливое» исключение, предназначенное для намеренного прерывания вычислений и быстрого выхода! глубоко вложенных процедур и функций. Генерируй процедурой Abort.

Е Access Violation- Ошибочный доступ к памяти; генерируется при попытке разыменования нулевого указателя nil, попыткаЯ писи в кодовую страницу, попытке доступа к адреса вне памяти, распределенной приложению.

EAssertionFailed- Ложное выражение, проверяемое процедурой AsserOT

EControlC- Нажатие пользователем клавиш Ctrl-C при выполнения консольного приложения. При обработке этого исключения можно выдать запрос пользователю, действительно ли он хочет прервать работу, и предпринять действия в зависимости от его ответа.

EConvertError- Ошибка преобразования строк или объектов (в частности, в функциях StrToInt, StrToFloat, StrToDatel.lJ

EData base Error- Ошибка работы с базами данных.

EDBClient- Ошибка в наборе чанных клиента. Свойство ErrorCode содержит код ошибки, возвращаемый BDE.

EReconcileEr-гог- Ошибка обновления данных компонента TClientData-set; свойство Context содержит информацию в виде сообщения об ошибке.

EDBEngineError- Ошибка в BDE. Свойство Errors содержит информацию об ошибке — объект типа TDBErrors. Свойство Error-Count хранит число ошибок.

ENoResultSet- Генерируется компонентом TQuery при попытке открыть запрос без оператора SELECT.

EUpdateError- Ошибка при обновлении в TProvider.

EDateTimeError- Ошибка ввода даты или времени в компоненте TDateTi-mePicker.

EInOutError- Ошибка ввода-вывода из файла; поле errorcode содержит информацию о конкретном виде ошибки

EIntError- Базовый класс исключений целочисленных математических операций (см. директивы компилятора {$1+} и {$1-}

EDivByZero- Попытка целочисленного деления на нуль.

ERangeError- Целочисленное значение или индекс вне допустимого диапазона.

EIntOverflow - Переполнение при операции с целыми числами.

EInvalidCast - Ошибка преобразования типа объекта операцией as.

EInvalidOperation - Ошибочная операция с компонентом; генерируется при попытке выполнить операцию, которая требует обработчика окна, над компонентом, не имеющем родителя (свойство Parent = nil). Это исключение также генерируется при выполнении операций перетаскивания формы (например, при попытке выполнить операцию Forml.BeginDrag).

EInvalidPointer - Ошибочная операция с указателем.

EListError - Ошибка работы с объектом типа списка TStringList и TList: попытке сослаться на элемент с индексом вне допустимых пределов, попытке добавления дубликата строки в объект TStringList, в котором значение свойства Duplicates равно dupError, попытке вставить элемент в сортированный список, так как это может нарушить правильную последовательность элементов.

EMathError Базовый класс исключений операций с плавающей запятой; всегда генерируются только потомки этого исключения; обработка исключения EMathError может использоваться для перехвата всех исключений операций с плавающей запятой.

ElnvalidArgu-^^rnent Недопустимое значение параметра при обращении к математической функции.

**3 Понятие исключительной ситуации**

**Исключительная ситуация (исключение)** – это некоторая ситуация в программе, которая требует специальной обработки. К таким ситуациям относятся ошибки времени выполнения программы, например, деление на ноль или попытка чтения информации из несуществующего файла.

При возникновении исключительной ситуации Delphi создает экземпляр некоторого класса, в зависимости от характера произошедшей ошибки. Все такие классы являются наследниками класса Exception (англ. Exception - исключение), что позволяет им быть «узнаваемыми» для Delphi. Механизм исключений состоит в досрочном прерывании подпрограммы или метода объекта, в которых произошла ошибка, и поиск обработчика исключения в этой подпрограмме (реализация обработчиков описана ниже).

Если обработчик не найден, то управление передается подпрограмме, вызвавшей ту подпрограмму, в которой произошла исключительная ситуация. В вызвавшей подпрограмме также производится поиск обработчика, и в случае его отсутствия управление передается следующей подпрограмме по стеку вызова подпрограмм.

Искомый обработчик должен обрабатывать именно те виды исключений, объект которого создан во время возникновения ошибки. Если ни в одной из подпрограмм стека вызова не будет найден обработчик, то исключение передается в стандартный обработчик исключений, автоматически создаваемый в программе. В этом случае на экран будет выдан диалог с сообщением об ошибке (рисунок 58), а программа продолжит свое выполнение в ожидании действий пользователя.

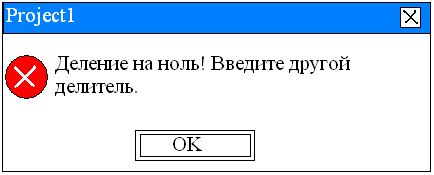


Рисунок – 58. Диалог с сообщением об ошибке.

Если же в одной из подпрограмм найден обработчик соответствующих исключений, то управление передается ему, а дальнейший путь работы программы определяется обычным образом.

**Модель исключительных ситуаций в Delphi**

Модель исключительных ситуаций в Object Pascal является невозобновляемой(non-resumable). При возникновении исключительной ситуации Вы уже не сможете вернуться в точку, где она возникла, для продолжения выполнения программы (это позволяет сделать возобновляемая(resumable) модель). Невозобновляемые исключительные ситуации разрушают стек, поскольку они сканируют его в поисках обработчика; в возобновляемой модели необходимо сохранять стек, состояние регистров процессора в точке возникновения ошибки и выполнять поиск обработчика и его выполнение в отдельном стеке. Возобновляемую систему обработки исключительных ситуаций гораздо труднее создать и применять, нежели невозобновляемую.

**4 Принудительное создание исключительной ситуации**

**«Тихие» исключения**

В Delphi предусмотрен специальный класс исключений **eAbort** (от англ. Abort Exception – исключительная ситуация прерванного выполнения), информация о которых не выдается пользователю стандартным обработчиком. Это так называемое «тихое» исключение (англ. Silent exception – тихое исключение). Заметим сходство названия этого исключения с процедурой досрочного выхода из подпрограммы **Abort**, которая, как раз и выбрасывает исключение eAbort.

Для выхода из подпрограммы с передачей управления в вызвавшую подпрограмму без возникновения исключения используется процедура **Exit** (англ. Exit - выход).

**Эмуляция ошибок**

Любая подпрограмма может также сама сгенерировать исключительную ситуацию («выбросить исключение», «возбудить исключение») с помощью оператора **Raise** (англ. Raise – будить, провоцировать):

**Raise** <Экземпляр класса исключений>;

Например, можно воспользоваться стандартным классом исключений **EdivByZero** (от англ. Division By Zero Exception – Исключительная ситуация деления на ноль), возникающим автоматически при попытке программы произвести деление на ноль.

Создание объекта исключения может располагаться прямо в вызове оператора Raise. Конструктор этого класса должен быть вызван с параметром-строкой, которая будет выдана пользователю:

**Raise** EDivByZero.Create (‘Деление на ноль! Введите другой делитель.’);

**5 Обработка исключений**

Когда некоторый фрагмент программного кода необходимо защитить от досрочного завершения по ошибке, его следует заключить в **блок обработки исключения** («защищенный юлок»). Таких блоков в Delphi предусмотрено два, а отличаются они характером действий, выполняемых в случае ошибки.

**Блок обработки исключений Try .. Except**

Блок обработки исключений Try .. Except (от англ. Try – Пробовать и Except - кроме) пытается выполнить заданный фрагмент программы, а в случае появления исключения – передает управление специальному обработчику, расположенному в секции Except :

**Try**

<Фрагмент программы>

**Except**

**On <**Класс исключений**> do** <Обработчик исключения>;

**. . . . . . . . . . .**

**On <**Класс исключений**> do** <Обработчик исключения>;

**Else <**Обработчик по умолчанию**>;**

**End;**

**Синтаксис обработки исключительных ситуаций**

Теперь, когда мы рассмотрели, что такое исключительные ситуации, давайте дадим ясную картину, как они применяются. Новое ключевое слово, добавленное в язык Object Pascal - try. Оно используется для обозначения первой части защищенного участка кода. Существует два типа защищенных участков:

* try..except
* try..finally

Первый тип используется для обработки исключительных ситуаций. Его синтаксис:

try

Statement 1;

Statement 2;

...

except

on Exception1 do Statement;

on Exception2 do Statement;

...

else

Statements; {default exception-handler}

end;

Для уверенности в том, что ресурсы, занятые вашим приложением, освободятся в любом случае, Вы можете использовать конструкцию второго типа. Код, расположенный в части finally, выполняется в любом случае, даже если возникает исключительная ситуация. Соответствующий синтаксис:

try

Statement1;

Statement2;

...

finally

Statements; { These statements always execute }

end;

**Вызов исключительной ситуации**

В процедуре C из примера мы уже могли видеть, как должна поступать программа при обнаружении состояния ошибки - она вызывает исключительную ситуацию:

raise ESampleError.Create('Error!');

После ключевого слова raise следует код, аналогичный тому, что используется для создания нового экземпляра класса. Действительно, в момент вызова исключительной ситуации создается экземпляр указанного класса; данный экземпляр существует до момента окончания обработки исключительной ситуации и затем автоматически уничтожается. Вся информация, которую нужно сообщить в обработчик ошибки передается в объект через его конструктор в момент создания.

Почти все существующие классы исключительных ситуаций являются наследниками базового класса Exception и не содержат новых свойств или методов. Класс Exception имеет несколько конструкторов, какой из них конкретно использовать - зависит от задачи. Описание класса Exception можно найти в on-line Help.

## Классы исключительных ситуаций

Исключительные ситуации в языке Delphi описываются классами. Каждый класс соответствует определенному типу исключительных ситуаций. Когда в программе возникает исключительная ситуация, создается объект соответствующего класса, который переносит информацию об этой ситуации из места возникновения в место обработки.

Классы исключительных ситуаций образуют иерархию, корнем которой является класс **Exception**. Класс **Exception** описывает самый общий тип исключительных ситуаций, а его наследники — конкретные виды таких ситуаций (таблица 4.1). Например, класс **EOutOfMemory** порожден от **Exception** и описывает ситуацию, когда свободная оперативная память исчерпана.

В следующей таблице приведены стандартные классы исключительных ситуаций, объявленные в модуле SysUtils. Они покрывают практически весь спектр возможных ошибок. Если их все-таки окажется недостаточно, вы можете объявить новые классы исключительных ситуаций, порожденные от класса **Exception** или его наследников.

Таблица – 5. Классы исключительных ситуаций

|  |  |
| --- | --- |
| **Класс исключительных ситуаций** | **Описание** |
| EAbort | «Безмолвная» исключительная ситуация, используемая для выхода из нескольких уровней вложенных блоков или подпрограмм. При этом на экран не выдается никаких сообщений об ошибке. Для генерации исключительной ситуации класса EAbort нужно вызвать стандартную процедуру Abort. |
| EInOutError | Ошибка доступа к файлу или устройству ввода-вывода. Код ошибки содержится в поле ErrorCode. |
| EExternal | Исключительная ситуация, возникшая вне программы, например, в операционной системе. |
| EExternalException | Исключительная ситуация, возникшая за пределами программы, например в DLL-библиотеке, разработанной на языке C++. |
| EHeapException | Общий класс исключительных ситуаций, возникающих при работе с динамической памятью. Является базовым для классов EOutOfMemory и EInvalidPointer.Внимание! Создание исключительных ситуаций этого класса (и всех его потомков) полностью берет на себя среда Delphi, поэтому никогда не создавайте такие исключительные ситуации с помощью оператора **raise**. |
| EOutOfMemory | Свободная оперативная память исчерпана (см. EHeadException). |
| EInvalidPointer | Попытка освободить недействительный указатель (см. EHeadException). Обычно это означает, что указатель уже освобожден. |
| EIntError | Общий класс исключительных ситуаций целочисленной арифметики, от которого порождены классы EDivByZero, ERangeError и EIntOverflow. |
| EDivByZero | Попытка деления целого числа на нуль. |
| ERangeError | Выход за границы диапазона целого числа или результата целочисленного выражения. |
| EIntOverflow | Переполнение в результате целочисленной операции. |
| EMathError | Общий класс исключительных ситуаций вещественной математики, от которого порождены классы EInvalidOp, EZeroDivide, EOverflow и EUnderflow. |
| EInvalidOp | Неверный код операции вещественной математики. |
| EZeroDivide | Попытка деления вещественного числа на нуль. |
| EOverflow | Потеря старших разрядов вещественного числа в результате переполнения разрядной сетки. |
| EUnderflow | Потеря младших разрядов вещественного числа в результате переполнения разрядной сетки. |
| EInvalidCast | Неудачная попытка приведения объекта к другому классу с помощью оператора **as**. |
| EConvertError | Ошибка преобразования данных с помощью функций IntToStr, StrToInt, StrToFloat, StrToDateTime. |
| EVariantError | Невозможность преобразования варьируемой переменной из одного формата в другой. |
| EAccessViolation | Приложение осуществило доступ к неверному адресу в памяти. Обычно это означает, что программа обратилась за данными по неинициализированному указателю. |
| EPrivilege | Попытка выполнить привилегированную инструкцию процессора, на которую программа не имеет права. |
| EStackOverflow | Стек приложения не может быть больше увеличен. |
| EControlC | Во время работы консольного приложения пользователь нажал комбинацию клавиш Ctrl+C. |
| EAssertionFailed | Возникает при вызове процедуры Assert, когда первый параметр равен значению False. |
| EPackageError | Проблема во время загрузки и инициализации библиотеки компонентов. |
| EOSError | Исключительная ситуация, возникшая в операционной системе. |

Наследование классов позволяет создавать семейства родственных исключительных ситуаций. Примером такого семейства являются классы исключительных ситуаций вещественной математики, которые объявлены в модуле SysUtils следующим образом.

|  |
| --- |
| type  EMathError = class(Exception);  EInvalidOp = class(EMathError);  EZeroDivide = class(EMathError);  EOverflow = class(EMathError);  EUnderflow = class(EMathError); |

Класс исключительных ситуаций EMathError является базовым для классов EInvalidOp, EZeroDivide, EOverflow и EUnderflow, поэтому, обрабатывая исключительные ситуации класса EMathError, вы будете обрабатывать все ошибки вещественной математики, включая EInvalidOp, EZeroDivide, EOverflow и EUnderflow.

Нетрудно заметить, что имена классов исключений начинаются с буквы **E** (от слова Exception). Этого правила полезно придерживаться при объявлении собственных классов исключений, например:

|  |
| --- |
| type  EMyException = class(Exception)  MyErrorCode: Integer;  end; |

Как описываются классы исключительных ситуаций понятно, рассмотрим теперь, как такие ситуации обрабатываются.

## 

## Обработка исключительных ситуаций

### Создание исключительной ситуации

Идея обработки исключительных ситуаций состоит в следующем. Когда подпрограмма сталкивается с невозможностью выполнения последующих действий, она создает объект с описанием ошибки и прерывает нормальный ход своей работы с помощью оператора **raise**.Так возникает исключительная ситуация.

|  |
| --- |
| raise EOutOfMemory.Create('Маловато памяти'); |

Данный оператор создает объект класса **EOutOfMemory** (класс ошибок исчерпания памяти) и прерывает нормальное выполнение программы. Вызывающие подпрограммы могут эту исключительную ситуацию перехватить и обработать. Для этого в них организуется так называемый *защищенный блок*:

|  |
| --- |
| try  // защищаемые от ошибок операторы  except  // операторы обработки исключительной ситуации  end; |

Между словами **try** и **except** помещаются защищаемые от ошибок операторы. Если при выполнении любого из этих операторов возникает исключительная ситуация, то управление передается операторам между словами **except** и **end**, образующим блок обработки исключительных ситуаций. При нормальном (безошибочном) выполнении программы блок **except**...**end** пропускается.

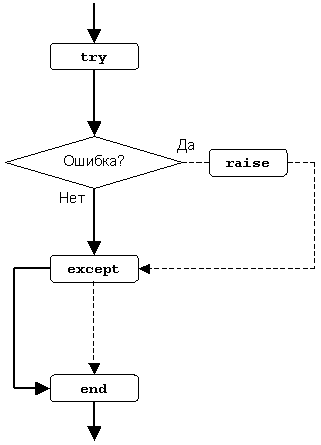


Рисунок – 59. Логика работы оператора try…except…end

При написании программы вы можете использовать вложенные защищенные блоки, чтобы организовать локальную и глобальную обработку исключительных ситуаций. Концептуально это выглядит следующим образом:

|  |
| --- |
| try  // защищаемые операторы  try  // защищаемые операторы  except  // локальная обработка исключительных ситуаций  end;  // защищаемые операторы  except  // глобальная обработка исключительных ситуаций  end; |

Исключительные ситуации внешнего защищенного блока, возникающие за пределами вложенного блока, обрабатываются внешней секцией **except**...**end**. Исключительные ситуации вложенного защищенного блока обрабатываются вложенной секцией **except**...**end**.

### Распознавание класса исключительной ситуации

Распознавание класса исключительной ситуации выполняется с помощью конструкций

|  |
| --- |
| on <класс исключительной ситуации> do <оператор>; |

которые записываются в секции обработки исключительной ситуации, например:

|  |
| --- |
| try  // вычисления с вещественными числами  except  on EZeroDivide do ... ; // обработка ошибки деления на нуль  on EMathError do ... ; // обработка других ошибок вещественной математики  end; |

Поиск соответствующего обработчика выполняется последовательно до тех пор, пока класс исключительной ситуации не окажется совместимым с классом, указанным в операторе **on**. Как только обработчик найден, выпоняется оператор, стоящий за словом **do** и управление передается за секцию **except**...**end**. Если исключительная ситуация не относится ни к одному из указанных классов, то управление передается во внешний блок **try**...**except**...**end** и обработчик ищется в нем.

Обратите внимание, что порядок операторов **on** имеет значение, поскольку распознавание исключительных ситуаций должно происходить от частных классов к общим классам, иначе говоря, от потомков к предкам. С чем это связано? Сейчас поймете. Представьте, к чему приведет изменение порядка операторов **on** в примере выше, если принять во внимание, что класс **EMathError** является базовым для **EZeroDivide**. Ответ простой: обработчик **EMathError** будет поглощать все ошибки вещественной математики, в том числе **EZeroDivide**, в результате обработчик **EZeroDivide** никогда не выполнится.

На самом высоком уровне программы бывает необходимо перехватывать все исключительные ситуации, чтобы в случае какой-нибудь неучтенной ошибки корректно завершить приложение. Для этого применяется так называемый *обработчик по умолчанию* (default exception handler). Он записывается в секции **except** после всех операторов **on** и начинается ключевым словом **else**:

|  |
| --- |
| try  { вычисления с вещественными числами }  except  on EZeroDivide do { обработка ошибки деления на нуль };  on EMathError do { обработка других ошибок вещественной математики };  else { обработка всех остальных ошибок (обработчик по умолчанию) };  end; |

Учтите, что отсутствие части **else** соответствует записи **else** **raise**, которое нет смысла использовать явно. Мы со своей стороны вообще не советуем вам пользоваться обработкой исключительных ситуаций по умолчанию, поскольку все ваши приложения будут строиться, как правило, на основе библиотеки VCL, в которой обработка по умолчанию уже предусмотрена.

### Пример обработки исключительной ситуации

В качестве примера обработки исключительной ситуации рассмотрим две функции: StringToCardinal и StringToCardinalDef.

Функция StringToCardinal выполняет преобразование строки в число с типом Cardinal. Если преобразование невозможно, функция создает исключительную ситуацию класса EConvertError.

|  |
| --- |
| function StringToCardinal(const S: string): Cardinal;  var  I: Integer;  B: Cardinal;  begin  Result := 0;  B := 1;  for I := Length(S) downto 1 do  begin  if not (S[I] in ['0'..'9']) then  raise EConvertError.Create(S + ' is not a valid cardinal value');  Result := Result + B \* (Ord(S[I]) - Ord('0'));  B := B \* 10;  end;  end; |

Функция StringToCardinalDef также выполняет преобразование строки в число с типом Cardinal, но в отличие от функции StringToCardinal она не создает исключительную ситуацию. Вместо этого она позволяет задать значение, которое возвращается в случае неудачной попытки преобразования:

|  |
| --- |
| function StringToCardinalDef(const S: string; Default: Cardinal = 0): Cardinal;  begin  try  Result := StringToCardinal(S);  except  on EConvertError do  Result := Default;  end;  end; |

Для преобразования исходной строки в число используется определенная выше функция StringToCardinal. Если при преобразовании возникает исключительная ситуация, то она «поглощается» функцией StringToCardinalDef, которая в этом случае возвращает значение параметра Default. Если происходит какая-нибудь другая ошибка (не EConvertError), то управление передается внешнему блоку обработки исключительных ситуаций, из которого была вызвана функция StringToCardinalDef.

Пример очень прост, но хорошо демонстрирует преимущества исключительных ситуаций перед традиционной обработкой ошибок. Представьте более сложные вычисления, состоящие из множества операторов, в каждом из которых может произойти ошибка. Насколько сложной окажется обработка ошибок многочисленными операторами **if** и насколько простой оператором **try**.

### Возобновление исключительной ситуации

В тех случаях, когда защищенный блок не может обработать исключительную ситуацию полностью, он выполняет только свою часть работы и возобновляет исключительную ситуацию с тем, чтобы ее обработку продолжил внешний защищенный блок:

|  |
| --- |
| try  // вычисления с вещественными числами  except  on EZeroDivide do  begin  // частичная обработка ошибки  raise; // возобновление исключительной ситуации  end;  end; |

Если ни один из внешних защищенных блоков не обработал исключительную ситуацию, то управление передается стандартному обработчику исключительной ситуации, завершающему приложение.

### 

### Доступ к объекту, описывающему исключительную ситуацию

При обработке исключительной ситуации может потребоваться доступ к объекту, описывающему эту ситуацию и содержащему код ошибки, текстовое описание ошибки и т.д. В этом случае используется расширенная запись оператора **on**:

|  |
| --- |
| on <идентификатор объекта> : <класс исключительной ситуации> do <оператор>; |

Например, объект исключительной ситуации нужен для того, чтобы выдать пользователю сообщение об ошибке:

|  |
| --- |
| try  // защищаемые операторы  except  on E: EOutOfMemory do  ShowMessage(E.Message);  end; |

Переменная **E** — это объект исключительной ситуации, **ShowMessage** — процедура модуля DIALOGS, отображающая на экране небольшое окно с текстом и кнопкой OK. Свойство **Message** типа **string** определено в классе **Exception**, оно содержит текстовое описание ошибки. Исходное значение для текста сообщения указывается при конструировании объекта исключительной ситуации.

Обратите внимание, что после обработки исключительной ситуации освобождение соответствующего объекта выполняется автоматически, вам этого делать не надо.

**Контрольные вопросы:**

* 1. Что называют отладкой в языках программирования?
  2. Назовите существующие взаимосвязанные технологии отладки ошибок.
  3. Что называют Отла́дчиком (debugger)?
  4. Что в программировании называют журналированием?
  5. Перечислите часто возникающие критические ошибки.
  6. Что называют «тихим исключением»?
  7. Что подразумевают под эмулятором ошибок?