Глава 17. Исключения

Суть дела не в полноте знания, а в полноте разумения.

Демокрит, VI век до н.э.

Исключением (exception) в программировании называют возникновение особой ситуации при счёте программы (Run-time error), когда её дальнейшее нормальное выполнение невозможно. Сначала разделим все исключения на синхронные и асинхронные. Синхронное исключение вызывается выполняемым в данный момент оператором программы, это, например, деление на ноль, выход результата за допустимый диапазон, попытка открыть несуществующий файл и т.д. Асинхронное исключение возникает в аппаратуре компьютера и его устройствах ввода/вывода независимо от выполняемых операторов программы. Это, например, нажатие кнопки на клавиатуре и движение мышки, истечение времени на таймере, фатальный сбой в работе какого-либо устройства ЭВМ и т.д.

Обычно в случае возникновения исключения операционная система (управляющая всей работой компьютера) прекращает выполнение программы и выдаёт аварийную диагностику.[1] Это называется стандартной реакцией на исключение. Мы уже изучили, как можно *блокировать* (запретить) такую реакцию, задав в программе нужную директиву. Например, директива `{$R-}` в языке Free Pascal блокирует реакцию на выход значения за допустимый диапазон, а директива `{$I-}` блокирует реакцию на ошибки ввода/вывода.

Во многих языках программирования, однако, можно задать свою собственную реакцию для обработки данной исключительной ситуации. Для этого вводится понятие защищённой области программы, в которой контролируется возникновение исключений, и программный обработчик таких исключений (exception handler). Посмотрим, как это делается в языке Free Pascal (мы рассмотрим только одну из возможностей). Само исключение реализовано в этом языке как класс, поэтому для использования таких возможностей следует включить работу программы в объектно-ориентированном режиме директивой `{$`**`mode`** `objfpc}`. Кроме того, необходимо подключить модуль языка Free Pascal с именем `SysUtils` (**`uses`** `SysUtils;`), ответственный за работу с исключениями.

Защищённая область программы и область обработки заключены в операторные скобки **`try except`** и **`except end`** и, таким образом, они должны примыкать друг у другу:

```
try  { начало защищённой области }
     { операторы защищённой области }
except  { область обработки исключений }
     { операторы обработки исключений }
end
```

Вся эта конструкция должна заканчивать раздел операторов, причём область обработки не может содержать операторы перехода **`goto`**. Таким образом, невозможно передать управление из области обработки в защищённую область, в частности, повторно выполнить оператор, вызвавший исключение.[2] Отметим что конструкция **`try except end`** может быть *вложенной* одна в другую, но мы такую возможность рассматривать не будем.

Каждое исключение имеет своё имя, все такие имена описаны в модуле `SysUtils` и начинаются с буквы `E` (Exception), например, `EDivByZero` (ошибка целочисленного деления). В области обработки исключения обычно располагаются так называемые обработчики, каждый из которых связан с конкретной ошибкой и выполняется при её возникновении. Обработчик имеет вид:

<обработчик> ::= **`on`** <исключение> **`do`** <оператор> [**`else`** <оператор>]

---

[1] В случае, когда произошло что-то ужасное (например, сбой основной памяти или процессора) и выдача «нормальной» аварийной диагностики невозможна, аппаратура ЭВМ пытается вывести примитивное аварийное сообщение. В первых версиях ОС Windos это называлось «синий экран смерти».

[2] В языках *высокого* уровня при обработке исключений обычно нельзя передавать управление в точку возникновения исключения. В частности, нельзя повторить оператор, вызвавший исключение, хотя отдельные языки (например, Lisp) это допускают. В языке Free Pascal, если защита применяется внутри блока операторов процедуры или функции, то можно из области обработки вызвать эту процедуру или функцию рекурсивно.

С возможностью повторить ошибочную команду на языке Ассемблера Вы познакомитесь в курсе по архитектурам ЭВМ.

<исключение> ::= [<переменная>:]<имя исключения>

Когда задана переменная, ей присваивается значение объекта, описывающего данное исключение. Эту переменную заранее описывать не надо, она считается локальной внутри данного обработчика исключения.

Основным назначением этого механизма является защита подпрограмм, чтобы при возникновении исключения они могли завершить свою работу не аварийно (со стандартной диагностикой операционной системы), а с возвратом в главную программу. При этом в точку возврата можно передать какого-нибудь код, сигнализирующий о «ненормальном» завершении этой подпрограммы.

В качестве примера рассмотрим следующую задачу. Надо написать защищённую функцию, которая вводит из стандартного потока `input` одно вещественное число, извлекает из него квадратный корень и возвращает целую часть (`trunc`) этого корня. При возникновении исключения, функция будет выводить диагностику, и вырабатывать отрицательный код возврата (`Exitcode`). Предусмотрим реакцию на следующие стандартные исключения:

1). `EInOutError`. Введена плохая лексема вещественного числа `{$I+}` (`Exitcode=-1`).
2). `ERangeError`. Результат функции вне допустимого диапазона `{$R+}` (`Exitcode=-2`).
3). `EMathError`. Корень из отрицательного числа (`Exitcode=-3`).[1]
4). `EControlC`. При вводе нажато ^C (прерывает счёт программы, `Exitcode=-4`).
5). Другие ошибки (`Exitcode=-5`).

Видно, что исключения будут обрабатываться, если в программе включены режимы работы с контролем `{$I+}` ошибок ввода/вывода и контролем `{$R+}` выхода значений за допустимый диапазон:

```
{$mode objfpc}
uses SysUtils;
function TruncSqrt: integer;
  var x: real;
begin {$I+} {$R+}
try
  Write('Введите x='); read(x);
  x:=sqrt(x); TruncSqrt:=trunc(x);
{ здесь хороший возврат }
except
  on E: EInOutError do begin TruncSqrt:=-1;
    Writeln('Причина АВОСТа=',E.message);
    Write('EInOutError: Ошибка ввода/вывода')
  end;
  on ERangeError do begin TruncSqrt:=-2;
    Write('ERangeError: Выход за диапазон')
  end;
  on EMathError do begin TruncSqrt:=-3;
    Write('EMathError: Ошибка с вещ. числами')
  end;
  on EControlC do begin TruncSqrt:=-4;
    Write('EControlC: Нажато ^C')
  end;
  else begin TruncSqrt:=-5;
    Write('Неизвестное исключение')
  end
{ здесь аварийный возврат }
end { try except }
end; { TruncSqrt }
```

[1] Сюда же относятся и другие ошибки при операциях с вещественными числами.

В точке ❶ создаётся объект-исключение с именем E, поэтому мы можем получить свойство объекта E.message, содержащее стандартной описание исключения (в нашем примере это будет 'Invalid Input'). У объекта E есть много других полезных свойств и методов.

В нашем языке Free Pascal реализована достаточно примитивная обработка исключений без возврата. Так как у нас запрещена передача управления назад по программе, то не получиться повторить ошибочный оператор с исправленными данными. Правда, можно вызывать подпрограммы (в том числе защищённые), и рекурсивно. Вот, например, как можно было бы повторить ошибочный оператор, если бы был разрешён **goto**:

```
{$mode objfpc}
uses SysUtils;
function TruncSqrt: integer;
  label L;
  var x: real;
begin {$I+} {$R+}
try
     Write('Введите x='); read(x);
L: ❶ x:=sqrt(x); TruncSqrt:=trunc(x);
except

  on EMathError do begin x:=abs(x);
     Write('x<0: Работаем с abs(x)');
     goto L
  end;
....
end { try except }
end; { TruncSqrt }
```

Во многих языках программирования (Lisp, и другие) реализована обработка исключений с возвратом (конечно, без «плохого» **goto** 🙂).

Таким образом, вызывающая программа может гибко реагировать на все возникающие исключения. Отметим, что на языке Ассемблера, который Вы будете изучать в следующем семестре, у программиста значительно больше возможностей по обработке исключений. Например, при извлечении корня из отрицательного числа программист может изменить значение аргумента (скажем, на его абсолютную величину) и повторить оператор ❶ `x:=sqrt(x)`.

Все исключения описаны в модуле SysUtils как объекты класса исключений. Среди классов исключений установлено наследование, например, класс исключений EMathError содержит подчинённые классы:

EZeroDivide – деление на ноль;
EOverflow – переполнение;
EUnderflow – потеря значимости;
EInvalidOp – неверная операция и т.д.

Программист может описать своё собственное исключение, например, получена плохо обусловленная матрица (ill-conditioned matrix) и назвать его, скажем EIllMatrix. Во время счёта программист может сам вызвать любое исключение, выполнив директиву **raise**, порождающую объект-исключение нужного класса, например:

raise EInOutError.Create; **raise** EIllMatrix.Create; и т.д.

Впервые обработка исключений была реализована в языке PL/I ещё в 1976 году. При этом программист мог описывать на этом языке свои исключительные ситуации и вызывать (любые) исключительные ситуации с помощью оператора

SIGNAL <условие> (<имя исключения>)