Документ подписан простой электронной подписью
Информация о владельце:
ФИО: Локтионова Оксана Геннадьевна
Должность: проректор по учебной работе
Дата подписания: 19.01.2023 14:05:31
Уникальный программный ключ:
0b817ca911e6668abb13a5d426d39e5f1c11eabbf73e943df4a4851fda56d089

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

Юго-Западный государственный университет

(ЮЗГУ)

Кафедра вычислительной техники

УТВЕРЖДАЮ
Проректор по учебной работе
_____ О.Г. Локтионова
«17» 01 2022 г.

ОБРАБОТКА ИСКЛЮЧИТЕЛЬНЫХ СИТУАЦИЙ

Методические указания по выполнению самостоятельной работы

для студентов направления подготовки 09.03.01

Курск 2022

УДК 621.3

Составитель: Э.И. Ватутин

Рецензент
Кандидат технических наук, доцент *Т.Н. Конаныхина*

Обработка исключительных ситуаций: методические указания по выполнению самостоятельной работы по дисциплинам «Программирование», «Объектно-ориентированное программирование» / Юго-Зап. гос. ун-т; сост.: Э.И. Ватутин; Курск, 2022. 9 с.: ил. 0.

Методические рекомендации содержат сведения по обработке исключительных ситуаций на современных языках программирования высокого уровня.

Предназначены для студентов направления подготовки 09.03.01 «Информатика и вычислительная техника».

Текст печатается в авторской редакции

Подписано в печать 17.01.22. Формат 60x84 1/16.
Усл. печ. л. Уч. – изд.л. Тираж 30 экз. Заказ 313. Бесплатно.
Юго-Западный государственный университет
305040, Курск, ул. 50 лет Октября, 94.

Содержание

Понятие и обработка исключительных ситуаций

Исключительные ситуации (исключения) – непредвиденные ситуации в программе, приводящие к ее аварийному завершению (например, деление на ноль, ошибка преобразования (например, в функции `IntToStr()`), нарушение защиты (access violation)). При добавлении в программу модуля `SysUtils` многие ошибки времени выполнения, немедленно завершающие программу, можно обрабатывать как исключения. Обработка исключений в современных языках программирования происходит через механизм ОС Windows, именуемый Structured Exception Handling (SEH).

Исключения являются элегантным механизмом отслеживания некорректных ситуаций в программе, который возможно применять, например, вместо использования множества проверок с использованием оператора `if`. В качестве примера рассмотрим проверку корректности числа с плавающей запятой.

```
Result := True;
try
  A := FloatToStr(Str);
except
  Result := False;
end;
```

Однако введение обработки исключений ведет к увеличению размера кода и уменьшению скорости выполнения программы в случае возникновения исключения (в некоторых случаях до нескольких порядков), поэтому по возможности необходимо избегать использования механизма исключений, в особенности в многократно повторяемых фрагментах программы.

Пример применения исключений при работе с файлами:

```
try
  AssignFile(F, FileName);
  Reset(F);  // Генерирует исключение EInOutError в случае
             // отсутствия файла
except
  on Exception do ...
end;
```

и пример фрагмента программы без них:

```
if FileExists(FileName) then begin
  AssignFile(F, FileName);
  Reset(F);
end;
```

Для поддержки обработки исключений в модуле SysUtils есть класс Exception, от которого в качестве наследников порождаются все остальные типы исключений, что в том числе допускает создание своих типы исключений. Для возбуждения исключений применяется следующая конструкция:

```
raise Объект_исключения at Адрес;
```

или в более короткой форме без указания адреса

```
raise Объект_исключения;
```

Это бывает полезно, например, в случае, если в процессе обработки была обнаружена некоторая ошибочная ситуация (например, получены некорректные данные). Пример программы:

```
function IsWithin(Value, Min, Max: Integer): Boolean;
```

```
begin
  if Min > Max then
    raise Exception.Create('Min должно быть меньше Max');

  Result := (Min <= Value) and (Value <= Max);
end;
```

В приведенном примере вместо непосредственного возбуждения исключения можно воспользоваться ASSERT'ами.

Созданный объект исключения всегда удаляется автоматически, вызов его деструктора вручную не требуется.

Использование исключений в секциях initialization и finalization моделей потенциально опасно и не рекомендуется, т.к. исключение может возникнуть до инициализации/после финализации модуля SysUtils, что не гарантирует его корректную обработку и устойчивую работу программы.

Для отслеживания исключительной ситуации применяется следующая конструкция:

```
try
  Защищаемый_блок
except
  Блок_обработки_исключений
end;
```

Защищаемый блок – набор строк кода, в составе которых потенциально может возникнуть исключение. Блок обработки исключений – список возможных исключений и их обработчики. Рассмотрим обработку исключений на примере математического расчета:

```
try
  …
```

```
except
  on E: EZeroDivide do …;
  on E: EOverflow do …;
else
  Обработка всех остальных исключений
end;
```

Если нет необходимости отслеживать конкретный тип исключения, можно писать код обработки исключения непосредственно между ключевыми словами `except` и `end`.

В списке исключений управления передается только первому соответствующему типу (в иерархии классов исключений), остальные игнорируются. По этой причине нельзя писать первым исключение `Exception` – оно является базовым классом для всех остальных – отбор дальнейших типов исключений не произойдет.

Если в текущем списке исключений нет соответствующей альтернативы, то производится раскрутка стека до тех пор, пока мы не окажемся в блоке `try ... except`. Если такого блока не будет найдено, то программа аварийно завершится.

Иногда возникает необходимость в повторном возбуждении исключения. Для этого используется конструкция `raise;` без указания объекта исключения (может применяться только внутри блока `try ... except`). Ниже приведен пример подобной ситуации – составление списка файлов в заданном каталоге.

```
function GetFileList(const Path: string): TStringList;
var
  I: Integer;
  SearchRec: TSearchRec;
begin
  Result := TStringList.Create;
  try
```

```
    I := FindFirst(Path, 0, SearchRec);
    while I = 0 do begin
      Result.Add(SearchRec.Name);
      I := FindNext(SearchRec);
    end;
  except
    Result.Free;    // Освобождение динамической памяти из-под
                    // созданного объекта во избежание
                    // образования утечки памяти
    raise;  // Отправка исключения «наружу» для дальнейшей
            // обработки
  end;
end;
```

Другой способ обработки исключений – `try ... finally` – обычно используется при выделении/освобождении каких-либо ресурсов. Пример работы с файлом и его гарантированного закрытия в случае возникновения исключения:

```
Reset(F);
try
  Действия с файлом
finally
  CloseFile(F);
end;
```

Блок обработки исключений является оператором – могут быть вложены друг в друга (`try ... except` может быть вложен в `try ... finally` и наоборот).

Контрольные вопросы

1. Для чего применяются исключения?
2. В чем отличие блоков try … except и try … finally?
3. Для чего используются различные классы исключений?
4. Какая конструкция языка Delphi применяется для возбуждения исключения?
5. Какая конструкция языка Delphi применяется для отлова и обработки исключительной ситуации?
6. Как влияет использование исключений на скорость выполнения программы?
7. Что происходит со стеком во время обработки исключений?

Библиографический список

1. Емельянов С.Г., Ватутин Э.И., Панищев В.С., Титов В.С. Процедурно-модульное программирование на Delphi: учебное пособие. М.: Аргамак-Медиа, 2014. 352 с.
2. Зотов И.В., Ватутин Э.И., Борзов Д.Б. Процедурно-ориентированное программирование на C++: учебное пособие. Курск: КурскГТУ, 2008. 211 с.