Строки в C#

Строки

Строка является объектом типа String, значением которого является текст. Текст хранится в виде последовательной доступной только для чтения набора объектов Char. В конце строки на языке C# отсутствует символ, заканчивающийся на NULL; поэтому строка C# может содержать любое число внедренных символов NULL ("\0"). Свойство Length строки представляет число объектов Char, содержащихся в этой строке, а не число символов Юникода.

В C# ключевое слово string является псевдонимом свойства String. Поэтому String и string эквивалентны

Строки

Объявление и инициализацию строк можно выполнять различными способами:

```
string message1;
string str = "Пример строки";
char[] letters = { 'A', 'B', 'C' };
string alphabet = new string(letters);
```

Строки

Над строками определены следующие операции:

- присваивание (=);
- конкатенация (объединение) или сцепление строк (+);
- две операции проверки эквивалентности: равно (= =) и не равно (!=);
- взятие индекса ([]).

Строки

Переприсваивание

Строки можно целиком переприсваивать:

```
string s1 = "Hello";
string s2 = s1;
```

Строки

Объединение строк

Можно объединять строки с помощью оператора +:

```
string s1 = "orange";
string s2 = "red";
s1 += s2;Console.WriteLine(s1);  // напечатается "orangered"
```

Строковые объекты являются неизменяемыми: после создания их нельзя изменить. Все методы String и операторы C#, которые, как можно было бы представить, изменяют строку, в действительности возвращают результаты в новый строковый объект.

.

Строки

Постоянство строк

Строковые объекты являются неизменяемыми: после создания их нельзя изменить. Все методы String и операторы C#, которые, как можно было бы представить, изменяют строку, в действительности возвращают результаты в новый строковый объект. В примере, когда содержимое строк s1 и s2 объединяется в одну строку, две исходные строки не изменяются. Оператор += создает новую строку с объединенным содержимым. Этот новый объект присваивается переменной s1, а исходный объект, который был присвоен строке s1, освобождается для сборки мусора, поскольку ни одна переменная не содержит ссылку на него.

Строки

Сравнения

Самый простой способ сравнения двух строк — использовать операторы == и !=, осуществляющие сравнение с учетом регистра:

```
string color1 = "red";
string color2 = "green";
string color3 = "red";
if (color1 == color3)   Console.WriteLine("Cтроки равны");
if (color1 != color2)   Console.WriteLine("Cтроки не равны");
```

Не допускается использование >,<,>=,<= для сравнения строк. Для строковых объектов существует метод CompareTo(), возвращающий целочисленное значение, зависящее от того, что одна строка может быть меньше (<), равна (==) или больше другой (>). При сравнении строк используется значение Юникода, при этом значение строчных букв меньше, чем значение заглавных.

Строки

Доступ к отдельным знакам

Квадратные скобки [] служат для доступа к отдельным знакам в объекте string, но при этом возможен доступ только для чтения:

```
string str = "test";
char x = str[2];  // x = 's';
```

Строки

В C# существуют два вида строковых констант:

- обычные константы, которые представляют строку символов, заключённую в кавычки;
- @-константы, заданные обычной константой с предшествующим знаком @.

Строки

Обычные константы

В обычных константах некоторые символы интерпретируются особым образом. Связано это, прежде всего, с тем, что необходимо уметь задавать в строке непечатаемые символы, такие, как, например, символ табуляции. Возникает необходимость задавать символы их кодом – в виде escape-последовательностей. Для всех этих целей используется комбинация символов, начинающаяся символом "\" - обратная косая черта. Это так называемые Escape-последовательности

Строки

Escape-последовательность	Имя символа
\'	Одинарная кавычка
\"	Двойная кавычка
\\	Обратная косая черта
\0	Нуль-символ
\n	Новая строка
\r	Возврат каретки
\t	Горизонтальная табуляция
\v	Вертикальная табуляция

При этом возникают неудобства: например, при задании констант, определяющих путь к файлу, приходится каждый раз удваивать символ обратной косой черты.

```
string path2 = "C:\\Users\\Mikant\\Documents";
```

Строки

@-константы

В @-константах все символы трактуются в полном соответствии с их изображением. Поэтому путь к файлу лучше задавать @-константой.
Единственная проблема в таких случаях: как задать символ кавычки, чтобы он не воспринимался как конец самой константы.
Решением является удвоение символа.

```
string path1 = @"C:\Users\Mikant\Documents";
```

Строки

Метод	Назначение
Compare()	Статический метод, который позволяет сравнить две строки
Concat()	Комбинирует отдельные экземпляры строк в одну строку (конкатенация)
Contains()	Метод, который позволяет определить, содержится ли в строке определенная подстрока
CopyTo()	Копирует определенное число символов, начиная с определенной позиции в новый экземпляр массива
IndexOf()	Находит первое вхождение заданной подстроки или символа в строке
Insert()	Метод, который позволяет вставить строку внутрь другой определенной строки
LastIndexOf()	То же, что IndexOf, но находит последнее вхождение

Строки

Метод	Назначение
Remove() Replace()	Методы, которые позволяют получить копию строки с соответствующими изменениями (удалением или заменой символов)
Substring()	Извлекает подстроку, начиная с определенной позиции строки
ToUpper () ToLower()	Методы, которые позволяют создавать копию текущей строки в формате, соответственно, верхнего или нижнего регистра
Trim()	Метод, который позволяет удалять все вхождения определенного набора символов с начала и конца текущей строки
Length()	Определяет длину строки

Строки

```
string s6 = «РГппУ";
Console.WriteLine(s6.ToUpper()); // Напечатается РГППУ

string s3 = "Visual C# Express";
string s5 = s3.Replace("C#", "Basic");
Console.WriteLine(s5);// напечатается "Visual Basic Express”

string s3 = "Visual C# Express";
string s4 = s3.Substring(7, 2);
Console.WriteLine(s4);// напечатается "C#"
```

Строки

Преобразование строк в другие типы

С помощью объекта Convert:

```
N = Convert.ToInt32(s1);
M = Convert.ToDouble(s2);
F = Convert.ToBoolean(s3);
B = Convert.ToByte(s4);
C = Convert.ToChar(k);
s5= Convert.ToString(x);
```

Структуры

Структуры

Структуры являются фундаментальными типами данных в C# и большинстве других современных языках программирования. Структуры в **C#** практически ничем не отличаются от структур в любом другом языке.

Структура - это набор зависимых друг от друга переменных. Зависимость здесь исключительно логическая и определяется условиями задачи.

Структура относится к типу значения, а не к ссылочному типу данных.

Структуры

Описание структуры:

```
struct имя_структуры
{
   public тип поле1;
   public тип поле2;
. . .
}
```

Структуры

Пример:

```
struct student
 {
   public string fio;
   public string FormOfEducation;
   public int course;
   public string faculty;
}
```

Структуры

```
static void Main(string[] args)
 {
    student stud;
   stud.fio = «Иванов Егор Петрович";
   stud.FormOfEducation = "очного";
   stud.course = 3;
   stud.faculty = "электроэнергетического";
   Console.WriteLine("СПРАВКА подтверждает, что "+stud.fio +
   " является студентом "+stud.FormOfEducation+
   " отделения ВоГТУ " +stud.course+" курса "+stud.faculty+
   " факультета");
    Console.Read();
 }
```

Структуры

student[]

Сортировка по возрастанию:

Массив.OrderBy(x=>x.поле)

stud.OrderBy(x=>x.fio);

Сортировка по убыванию:

Массив.OrderByDescending (x=>x.поле)

stud.OrderBy Descending (x=>x.cours)

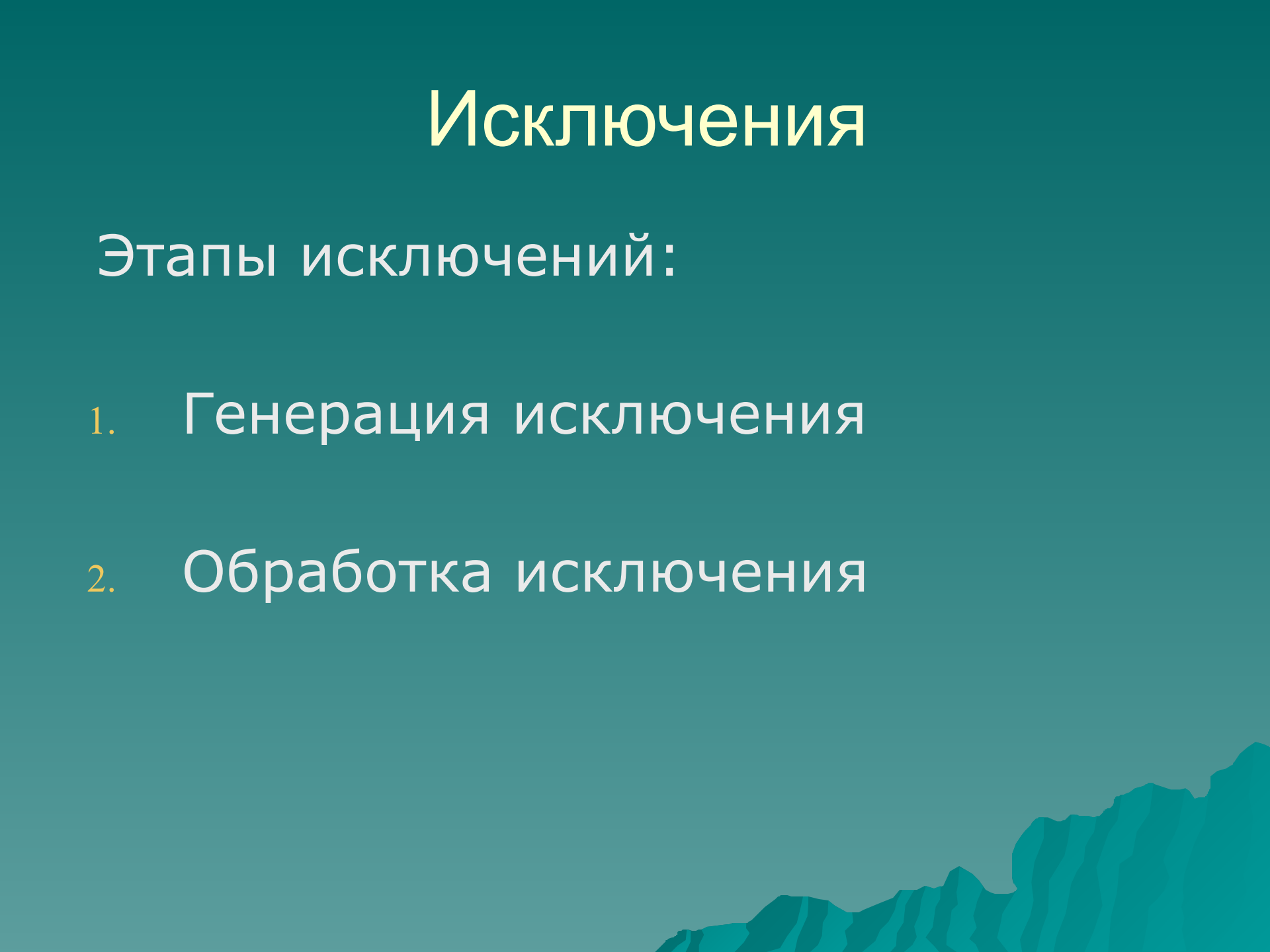
Исключения

В языке C# ошибки в программе во время выполнения передаются через программу посредством механизма, называемого *исключениями*. *Исключения* создаются кодом, который встречает ошибку и перехватываются кодом, который может исправить ее. Исключения могут создаваться средой CLR платформы .NET Framework или кодом в программе.

Исключения

Этапы исключений:

1. Генерация исключения
2. Обработка исключения



Исключения

Схема обработки исключений в C#:

```
try {... }
  catch (T1 e1) {...}
  ...
catch(Tk ek) {...}
finally {...}
```

*Всюду в тексте модуля, где синтаксически допускается использование блока, этот блок можно сделать охраняемым, добавив ключевое слово **try**. Вслед за **try**-блоком могут следовать **catch**-блоки, называемые блоками- обработчиками исключительных ситуаций, их может быть несколько, они могут и отсутствовать. Завершает эту последовательность **finally**-блок - блок завершения (финализации), который также может отсутствовать.*

Исключения

Свойства исключений:

- Исключения имеют типы, в конечном счете являющиеся производными от System.Exception.
- Следует использовать блок try для заключения в него инструкций, которые могут выдать исключения.
- При возникновении исключения в блоке try поток управления немедленно переходит к первому соответствующему обработчику исключений, присутствующему в стеке вызовов. В языке C# ключевое слово catch используется для определения обработчика исключений.
- Если обработчик для определенного исключения не существует, выполнение программы завершается с сообщением об ошибке.
- Не перехватывайте исключение, если его нельзя обработать, и оставьте приложение в известном состоянии. При перехвате System.Exception вновь инициируйте это исключение с использованием ключевого слова throw в конце блока catch.
- Исключения могут явно генерироваться программной с помощью ключевого слова throw.
- Код в блоке finally выполняется, даже при возникновении исключения. Блок finally используется для освобождения ресурсов, например для закрытия потоков или файлов, открытых в блоке try.

Исключения

- Блок try используется программистами C# для разбиения на разделы кода, который может затрагиваться исключением.
- Блок catch может указывать тип перехватываемого исключения. Спецификация типа называется фильтр исключений. Несколько блоков catch с различными фильтрами исключений могут быть соединены друг с другом. Блоки catch проверяются сверху вниз в коде, однако для каждого вызванного исключения выполняется только один блок catch.
- Блок finally позволяет удалить действия, выполненные в блоке try. При наличии блока finally он выполняется последним, после блока try и всех выполняемых блоков catch. Блок finally выполняется всегда, вне зависимости от возникновения исключения или обнаружения блока catch, соответствующего типу исключения.

Исключения

В стандарте языка C# конструкция try/catch/finally называются try оператором. Определены три формы оператора try:

- Блок контроля, за которым следуют catch-обработчики (один или несколько)
- Блок контроля, за которым следует блок завершения (finally-блок)
- Блок контроля, за которым следуют catch-обработчики (один или несколько), размещен блок завершения (finally-блок)

Исключения

Имеется три формы catch-инструкции:

```
catch (тип_исключения имя) {операторы}
catch (тип_исключения) {операторы}
catch {операторы}
```

Для выполнения перехвата исключений вне зависимости от их типа (перехват всех исключений) используется оператор catch без параметров.

Исключения

```
private void button1_Click(object sender, EventArgs e)
      {  try
         {  stud[i].fio = textBox1.Text;
            stud[i].dat = dateTimePicker1.Value;
            if (radioButton1.Checked) stud[i].pol = 'ж';
            else stud[i].pol = 'м';
            stud[i].ves =
            Convert.ToInt32(numericUpDown1.Value);
            stud[i].ekat = checkBox1.Checked;
            stud[i].rost = Convert.ToDouble(textBox2.Text);
            i++;
         }
         catch
         { MessageBox.Show("Ошибка");}

      }
```

Исключения

Основные системные исключения	
Исключение	**Значение**
ArrayTypeMismatchException	Тип сохраненного значения несовместим с типом массива
DivideByZeroException	Предпринята попытка деления на ноль
IndexOutOfRangeException	Индекс массива выходит за пределы диапазона
InvalidCastException	Некорректное преобразование в процессе выполнения
ArgumentException	Недопустимое значение аргумента
Overflow/Exception	Переполнение при выполнении арифметической операции
FormatException	Ошибка в формате данных

Исключения

```
private void button1_Click(object sender, EventArgs e)
      {
         try
         {   stud[i].fio = textBox1.Text;
            stud[i].dat = dateTimePicker1.Value;
            if (radioButton1.Checked) stud[i].pol = 'ж';
            else stud[i].pol = 'м';
            stud[i].ves = Convert.ToInt32(numericUpDown1.Value);
            stud[i].ekat = checkBox1.Checked;
            stud[i].rost = Convert.ToDouble(textBox2.Text);
            i++;  }
         catch (System.FormatException)
         { MessageBox.Show("Ввод не числа"); }
         catch (System.IndexOutOfRangeException)
         {  MessageBox.Show("Выход за пределы массива"); }
         finally
         { textBox1.Text="";
          textBox2.Text="";
         textBox1.Focus();}
      }
```

Контролируемый ввод

```
private void textBox2_KeyPress(object sender,
  KeyPressEventArgs e)
      {  bool zpt=false;
         if (char.IsDigit(e.KeyChar) == true) return;
         if (e.KeyChar == (char)Keys.Back) return;
         if (textBox2.Text.IndexOf(',') != -1)
             zpt = true;
         if (zpt == true) { e.Handled = true; return;
  }
         if (e.KeyChar == ',') return;
         e.Handled = true;
      }
```

Контролируемый ввод