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;
. . .
}
```

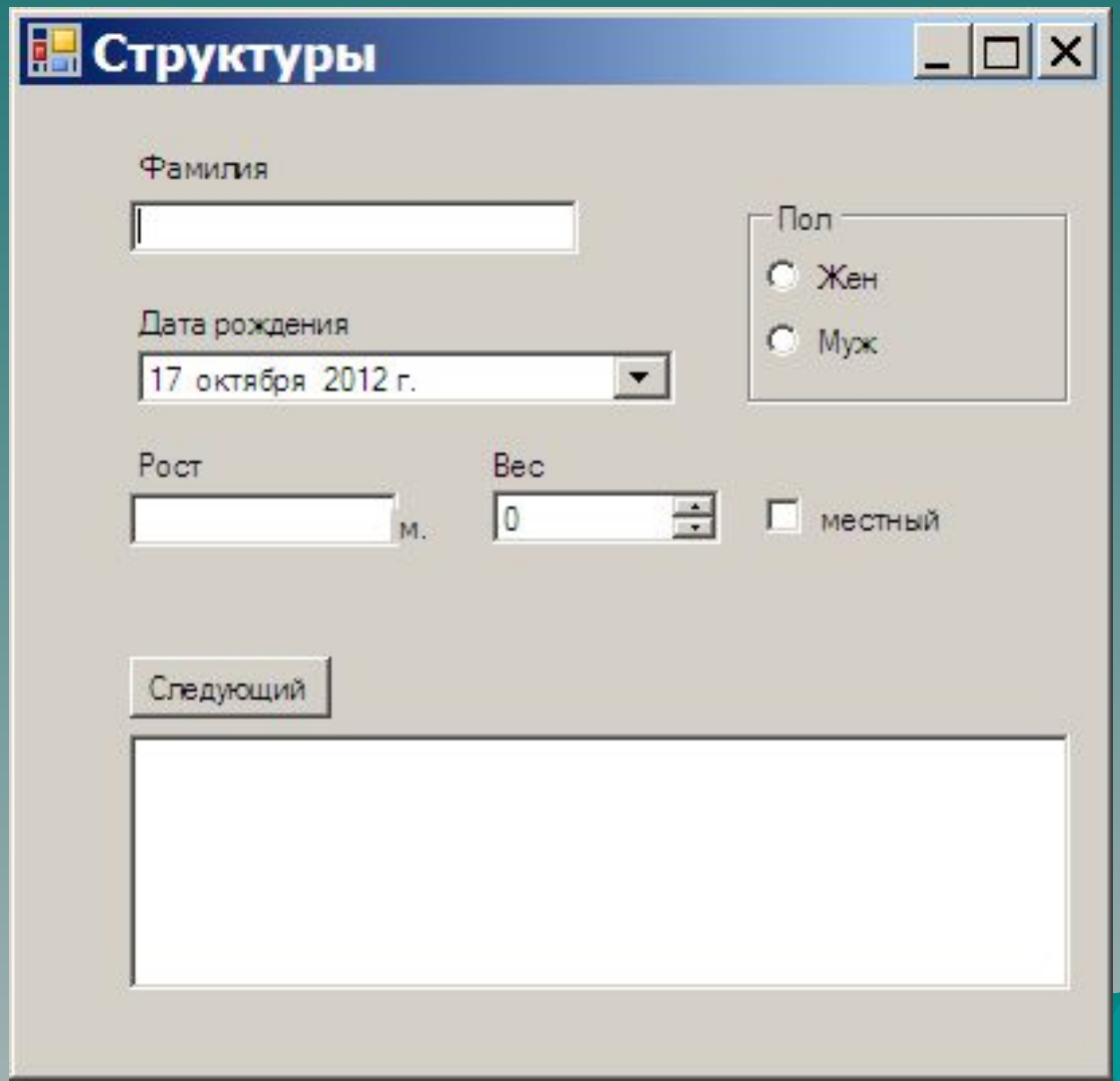
Структуры

Пример:

```
struct person
        {
            public string fio;
            public DateTime dat;
            public char pol;
            public double rost;
            public int ves;
            public bool ekat;
        };
```

Структуры

Ввод массива структур



Структуры

```
public partial class Form1 : Form
    {
        struct person
        {
            public string fio;
            public DateTime dat;
            public char pol;
            public double rost;
            public int ves;
            public bool ekat;
        };
        int i;
        person[] stud = new person[10];
        public Form1()
        {  InitializeComponent();
            i = 0;
        }
```

Структуры

```
private void button1_Click(object sender, EventArgs e)
    {
        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++;
    }
```

Структуры

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.rost)

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 форсы 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