Синтаксис C#
(ветвления, исключения)

Артём Трофимушкин

Ветвления

Ветвление – это команда алгоритма, в которой делается выбор, выполнять или не выполнять какую-нибудь группу команд в зависимости от условий.

На алгоритмической схеме ветвление изображают в виде ромба, имеющего один вход и два выхода. Внутри ромба пишется утверждение. В зависимости от истинности утверждения выполняется та или иная ветка кода:

Условный оператор if...else

```
// Example of if...else #1
Console.WriteLine("Press any key for analysis:");
char c = Console.ReadKey(true).KeyChar;

if (char.IsLetterOrDigit(c))                    // conditional statement
{                                               // block if it is true
    Console.WriteLine("You entered letter or digit!");
}
else
{                                               // block if it is false
    Console.WriteLine("You pressed a strange key...");
}

Console.WriteLine("Press any key to exit...");
Console.ReadKey();
```

Условный оператор if...else

```
// Example of if...else #2
Console.WriteLine("Enter a number less than 100:");
string strNum = Console.ReadLine();
int num = int.Parse(strNum);
string message;

if (num < 100)
{
    message = "Correct!";
}
else
{
    message = "Error!";
}

Console.WriteLine(message + " Press any key to exit...");
Console.ReadKey();
```

Тернарный условный оператор ?:

Терна́рная условная операция (от латинского *ternarius* — “тройной”):

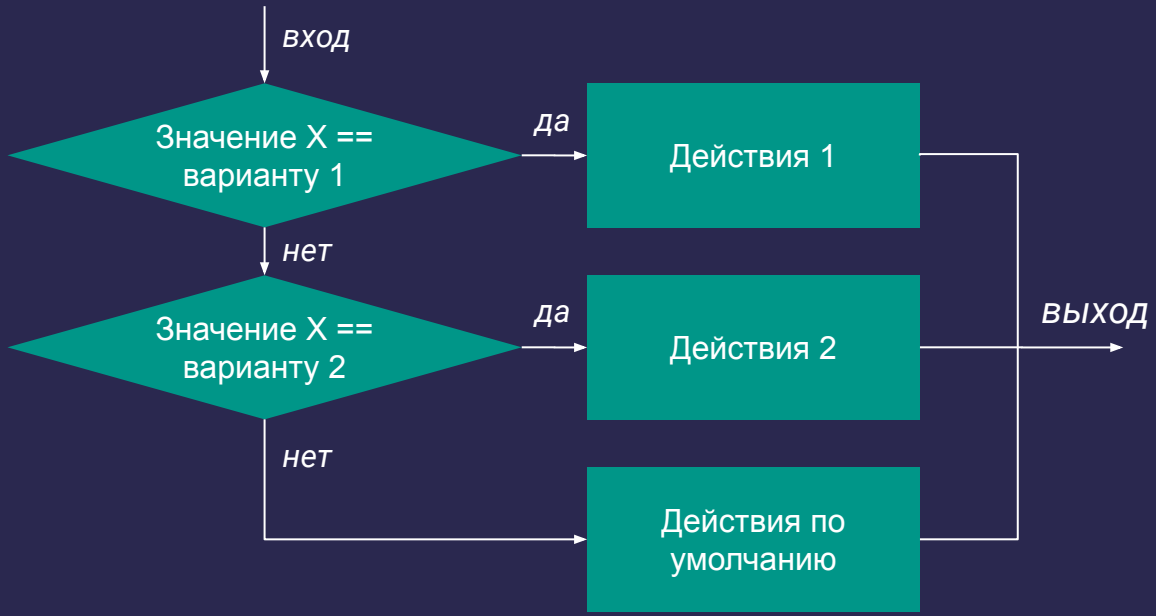
```
// Example of operator ?:
Console.WriteLine("Enter a number less than 100:");
string strNum = Console.ReadLine();
int num = int.Parse(strNum);
string message;

message = num < 100 // condition statement
    ? "Correct!"        // value when condition is true
    : "Error!";     // value when condition is false

Console.WriteLine(message + " Press any key to exit...");
Console.ReadKey();
```

Конструкция switch

Конструкция switch отличается от if...else: она позволяет сравнить выражение с набором возможных значений:



Конструкция switch

```
public enum Color { Red, Green, Blue }
public static void Main()
{
    Color c = (Color)(new Random()).Next(0, 3);
    switch (c)
    {
        case Color.Red:
            Console.WriteLine("The color is red");
            break;
        case Color.Blue:
            Console.WriteLine("The color is blue");
            break;
        default:
            Console.WriteLine("The color is unknown.");
            break;
    }
}
```

Исключения

Исключения позволяют обозначить, что во время выполнения программы произошла ошибка.

Объекты исключений, описывающие ошибку, создаются и затем вызываются с помощью ключевого слова throw.

Программисты должны вызывать исключения в том случае, если прогнозируется неверное поведение программы.

Объекты исключений наследуются от базового класса System.Exception.

Исключения не рекомендуется использовать для изменения потока программы в рамках обычного выполнения. Их следует использовать только для сообщения о состояниях ошибки и их обработки.

Генерация собственного исключения

```
Console.WriteLine("Enter a number less than 100:");
string strNum = Console.ReadLine();
int num = int.Parse(strNum);

if (num >= 100)
{
    // throwing new exception according to our logic
    throw new Exception("The value should be less than 100!");
}

Console.WriteLine($"You entered correct value {num}");
Console.WriteLine("Press any key to exit...");
Console.ReadKey();
```

Обработка исключений

Функции обработки исключений помогают справиться с непредвиденными или исключительными проблемами, которые возникают при выполнении программы.

Обработка исключений использует ключевые слова try, catch и finally для действий, которые могут оказаться неудачными.

Если исключение перехватывается, его необходимо либо обработать, либо генерировать повторно используя ключевое слово throw.

Обработка исключений

```
// for example, we would like to create a simple calculator
Console.WriteLine("Enter integer value A: ");
int a = int.Parse(Console.ReadLine());

Console.WriteLine("Enter integer value B: ");
int b = int.Parse(Console.ReadLine());

int result = a / b;

Console.WriteLine($"{a} divide by {b} equals to {result:##.###}");

Console.WriteLine("Press any key to exit...");
Console.ReadKey();
```

Обработка исключений

```
// for example, we would like to create a simple calculator
Console.WriteLine("Enter integer value A: ");
int a = int.Parse(Console.ReadLine());

Console.WriteLine("Enter integer value B: ");
int b = int.Parse(Console.ReadLine());

int result = a / b;

Console.WriteLine($"{a} divide by {b} equals to {result:##.###}");

Console.WriteLine("Press any key to exit...");
Console.ReadKey();
```

Обработка исключений

```
try
{
    Console.WriteLine("Enter integer value A: ");
    int a = int.Parse(Console.ReadLine());
    Console.WriteLine("Enter integer value B: ");
    int b = int.Parse(Console.ReadLine());
    int result = a / b;

    Console.WriteLine($"{a} divide by {b} equals to {result}");
}
catch (Exception e) // We can specify exception variable to use it later
{
    Console.WriteLine("Can't continue calculation:");
    Console.WriteLine($"{e.GetType()}: {e.Message}"); // here!
}
Console.WriteLine("Press any key to exit...");
Console.ReadKey();
```

Обработка исключений

```
try
{
    Console.WriteLine("Enter integer value A: ");
    int a = int.Parse(Console.ReadLine());
    Console.WriteLine("Enter integer value B: ");
    int b = int.Parse(Console.ReadLine());
    int result = a / b;

    Console.WriteLine($"{a} divide by {b} equals to {result}");
}
catch // catching ALL exceptions is a bad practice as it may hide problems!
{
    Console.WriteLine("Can't continue calculation! Something goes wrong!");
}

Console.WriteLine("Press any key to exit...");
Console.ReadKey();
```

Обработка исключений

```
try
{
    Console.WriteLine("Enter integer value A: ");
    int a = int.Parse(Console.ReadLine());
    Console.WriteLine("Enter integer value B: ");
    int b = int.Parse(Console.ReadLine());
    int result = a / b;
    Console.WriteLine($"{a} divide by {b} equals to {result}");
}
catch (FormatException)         // Cannot parse integer value!
{
    Console.WriteLine("You entered wrong data!");
}
catch (DivideByZeroException e) // Cannot divide by zero!
{
    Console.WriteLine("Cannot divide by zero!");
}
```

Домашнее задание

Написать консольное приложение, которое спросит у пользователя тип фигуры (1 - круг, 2 - равносторонний треугольник, 3 - прямоугольник), затем спросит параметры фигуры:

1. для круга - диаметр
2. для треугольника - длину стороны
3. для прямоугольника - ширину и высоту

В качестве результата программа должна вывести площадь поверхности и длину периметра соответствующей фигуры.

Тип фигур должен быть объявлен в виде перечисления.

Необходимо обработать все предсказуемые исключения.

Домашнее задание

Пример работы программы (при корректном вводе):

```
> Введите тип фигуры (1 круг, 2 равносторонний треугольник, 3 прямоугольник):
> 3 /это ввод пользователя, соответствующий выбору прямоугольника/
> Введите длину прямоугольника:
> 12.1 /ввод пользователем ширины/
> Введите высоту прямоугольника:
> 9.4 /ввод пользователя высоты/
> Площадь поверхности: 113.74
> Длина периметра: 43
```

Пример работы программы (при неверном вводе):

```
> Введите тип фигуры (1 круг, 2 равносторонний треугольник, 3 прямоугольник):
> 3 /это ввод пользователя, соответствующий выбору прямоугольника/
> Введите длину прямоугольника:
> Abcd /ввод пользователем нечислового значения/
> Ошибка! Введено нечисловое значение!
```

Спасибо за внимание.