# Министерство образования Республики Беларусь

УПРАВЛЕНИЕ ОБРАЗОВАНИЯ МОГИЛЕВСКОГО ОБЛАСТНОГО ИСПОЛНИТЕЛЬНОГО КОМИТЕТА

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«Могилевский государственный политехнический колледж»

# Домашняя контрольная работа №1

По дисциплине: «**Конструирование программ и языки программирования**»

Группа ПО-455

## Выполнил О. И. Чижик

Шифр 14

# 2021

**25 Опишите синтаксис операций ввода/вывода информации на консоль**

Консольный вывод

Для вывода информации на консоль мы уже использовали встроенный метод Console.WriteLine. То есть, если мы хотим вывести некоторую информацию на консоль, то нам надо передать ее в метод Console.WriteLine:

using System;

namespace HelloApp

{

class Program

{

static void Main(string[] args)

{

string hello = "Привет мир";

Console.WriteLine(hello);

Console.WriteLine("Добро пожаловать в C#!");

Console.WriteLine("Пока мир...");

Console.WriteLine(24.5);

Console.ReadKey();

}

}

}

Консольный вывод:

Привет мир!

Добро пожаловать в C#!

Пока мир...

24,5

Нередко возникает необходимость вывести на консоль в одной строке значения сразу нескольких переменных. В этом случае мы можем использовать прием, который называется интерполяцией:

using System;

namespace HelloApp

{

class Program

{

static void Main(string[] args)

{

string name = "Tom";

int age = 34;

double height = 1.7;

Console.WriteLine($"Имя: {name} Возраст: {age} Рост: {height}м");

Console.ReadKey();

}

}

}

Для встраивания отдельных значений в выводимую на консоль строку используются фигурные скобки, в которые заключается встраиваемое значение. Это можем значение переменной ({name}) или более сложное выражение (например, операция сложения {4 + 7}). А перед всей строкой ставится знак доллара $.

При выводе на консоль вместо помещенных в фигурные скобки выражений будут выводиться их значения:

Имя: Tom Возраст: 34 Рост: 1,7м

Есть другой способ вывода на консоль сразу нескольких значений:

using System;

namespace HelloApp

{

class Program

{

static void Main(string[] args)

{

string name = "Tom";

int age = 34;

double height = 1.7;

Console.WriteLine("Имя: {0} Возраст: {2} Рост: {1}м", name, height, age);

Console.ReadKey();

}

}

}

Этот способ подразумевает, что первый параметр в методе Console.WriteLine представляет выводимую строку ("Имя: {0} Возраст: {2} Рост: {1}м"). Все последующие параметры представляют значения, которые могут быть встроенны в эту строку (name, height, age). При этом важен порядок подобных параметров. Например, в данном случае вначале идет name, потом height и потом age. Поэтому у name будет представлять параметр с номером 0 (нумерация начинается с нуля), height имеет номер 1, а age - номер 2. Поэтому в строке "Имя: {0} Возраст: {2} Рост: {1}м" на место плейсхолдеров {0}, {2}, {1} будут вставляться значения соответствующих параметров.

Кроме Console.WriteLine() можно также использовать метод Console.Write(), он работает точно так же за тем исключением, что не осуществляет переход на следующую строку.

Консольный ввод

Кроме вывода информации на консоль мы можем получать информацию с консоли. Для этого предназначен метод Console.ReadLine(). Он позволяет получить введенную строку.

using System;

namespace HelloApp

{

class Program

{

static void Main(string[] args)

{

Console.Write("Введите свое имя: ");

string name = Console.ReadLine();

Console.WriteLine($"Привет {name}");

Console.ReadKey();

}

}

}

В данном случае все, что вводит пользователь, с помощью метода Console.ReadLine передается в переменную name.

Пример работы программы:

Введите свое имя: Том

Привет Том

Таким образом мы можем вводить информацию через консоль. Однако минусом этого метода является то, что Console.ReadLine считывает информацию именно в виде строки. Поэтому мы можем по умолчанию присвоить ее только переменной типа string. Как нам быть, если, допустим, мы хотим ввести возраст в переменную типа int или другую информацию в переменные типа double или decimal? По умолчанию платформа .NET предоставляет ряд методов, которые позволяют преобразовать различные значения к типам int, double и т.д. Некоторые из этих методов:

Convert.ToInt32() (преобразует к типу int)

Convert.ToDouble() (преобразует к типу double)

Convert.ToDecimal() (преобразует к типу decimal)

Пример ввода значений:

using System;

namespace HelloApp

{

class Program

{

static void Main(string[] args)

{

Console.Write("Введите имя: ");

string name = Console.ReadLine();

Console.Write("Введите возраст: ");

int age = Convert.ToInt32(Console.ReadLine());

Console.Write("Введите рост: ");

double height = Convert.ToDouble(Console.ReadLine());

Console.Write("Введите размер зарплаты: ");

decimal salary = Convert.ToDecimal(Console.ReadLine());

Console.WriteLine($"Имя: {name} Возраст: {age} Рост: {height}м Зарплата: {salary}$");

Console.ReadKey();

}

}

}

При вводе важно учитывать текущую операционную систему. В одних культурах разделителем между целой и дробной частью является точка (США, Великобритания...), в других - запятая (Россия, Германия...). Например, если текущая ОС - русскоязычная, значит, надо вводить дробные числа с разделителем запятой. Если локализация англоязычная, значит, разделителем целой и дробной части при вводе будет точка.

Пример работы программы:

Введите имя: Том

Введите возраст: 25

Введите рост: 1,75

Введите размер зарплаты: 300,67

Имя: Том Возраст: 25 Рост: 1,75м Зарплата: 300,67$

**51 Опишите средства для обработки исключительных ситуаций в языке C#**

Иногда при выполнении программы возникают ошибки, которые трудно предусмотреть или предвидеть, а иногда и вовсе невозможно. Например, при передачи файла по сети может неожиданно оборваться сетевое подключение. такие ситуации называются исключениями. Язык C# предоставляет разработчикам возможности для обработки таких ситуаций. Для этого в C# предназначена конструкция try...catch...finally.

1

2

3

4

5

6

7

8

9

10

11

12

try

{

}

catch

{

}

finally

{

}

При использовании блока try...catch..finally вначале выполняются все инструкции в блоке try. Если в этом блоке не возникло исключений, то после его выполнения начинает выполняться блок finally. И затем конструкция try..catch..finally завершает свою работу.

Если же в блоке try вдруг возникает исключение, то обычный порядок выполнения останавливается, и среда CLR начинает искать блок catch, который может обработать данное исключение. Если нужный блок catch найден, то он выполняется, и после его завершения выполняется блок finally.

Если нужный блок catch не найден, то при возникновении исключения программа аварийно завершает свое выполнение.

Рассмотрим следующий пример:

1

2

3

4

5

6

7

8

9

10

11

class Program

{

static void Main(string[] args)

{

int x = 5;

int y = x / 0;

Console.WriteLine($"Результат: {y}");

Console.WriteLine("Конец программы");

Console.Read();

}

}

В данном случае происходит деление числа на 0, что приведет к генерации исключения. И при запуске приложения в режиме отладки мы увидим в Visual Studio окошко, которое информирует об исключении:

Исключения в C#

В этом окошке мы видим, что возникло исключение, которое представляет тип System.DivideByZeroException, то есть попытка деления на ноль. С помощью пункта View Details можно посмотреть более детальную информацию об исключении.

И в этом случае единственное, что нам остается, это завершить выполнение программы.

Чтобы избежать подобного аварийного завершения программы, следует использовать для обработки исключений конструкцию try...catch...finally. Так, перепишем пример следующим образом:

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

class Program

{

static void Main(string[] args)

{

try

{

int x = 5;

int y = x / 0;

Console.WriteLine($"Результат: {y}");

}

catch

{

Console.WriteLine("Возникло исключение!");

}

finally

{

Console.WriteLine("Блок finally");

}

Console.WriteLine("Конец программы");

Console.Read();

}

}

В данном случае у нас опять же возникнет исключение в блоке try, так как мы пытаемся разделить на ноль. И дойдя до строки

1

int y = x / 0;

выполнение программы остановится. CLR найдет блок catch и передаст управление этому блоку.

После блока catch будет выполняться блок finally.

Возникло исключение!

Блок finally

Конец программы

Таким образом, программа по-прежнему не будет выполнять деление на ноль и соответственно не будет выводить результат этого деления, но теперь она не будет аварийно завершаться, а исключение будет обрабатываться в блоке catch.

Следует отметить, что в этой конструкции обязателен блок try. При наличии блока catch мы можем опустить блок finally:

1

2

3

4

5

6

7

8

9

10

try

{

int x = 5;

int y = x / 0;

Console.WriteLine($"Результат: {y}");

}

catch

{

Console.WriteLine("Возникло исключение!");

}

И, наоборот, при наличии блока finally мы можем опустить блок catch и не обрабатывать исключение:

1

2

3

4

5

6

7

8

9

10

try

{

int x = 5;

int y = x / 0;

Console.WriteLine($"Результат: {y}");

}

finally

{

Console.WriteLine("Блок finally");

}

Однако, хотя с точки зрения синтаксиса C# такая конструкция вполне корректна, тем не менее, поскольку CLR не сможет найти нужный блок catch, то исключение не будет обработано, и программа аварийно завершится.

Обработка исключений и условные конструкции

Ряд исключительных ситуаций может быть предвиден разработчиком. Например, пусть программа предусматривает ввод числа и вывод его квадрата:

1

2

3

4

5

6

7

8

9

static void Main(string[] args)

{

Console.WriteLine("Введите число");

int x = Int32.Parse(Console.ReadLine());

x \*= x;

Console.WriteLine("Квадрат числа: " + x);

Console.Read();

}

Если пользователь введет не число, а строку, какие-то другие символы, то программа выпадет в ошибку. С одной стороны, здесь как раз та ситуация, когда можно применить блок try..catch, чтобы обработать возможную ошибку. Однако гораздо оптимальнее было бы проверить допустимость преобразования:

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

static void Main(string[] args)

{

Console.WriteLine("Введите число");

int x;

string input = Console.ReadLine();

if (Int32.TryParse(input, out x))

{

x \*= x;

Console.WriteLine("Квадрат числа: " + x);

}

else

{

Console.WriteLine("Некорректный ввод");

}

Console.Read();

}

Метод Int32.TryParse() возвращает true, если преобразование можно осуществить, и false - если нельзя. При допустимости преобразования переменная x будет содержать введенное число. Так, не используя try...catch можно обработать возможную исключительную ситуацию.

С точки зрения производительности использование блоков try..catch более накладно, чем применение условных конструкций. Поэтому по возможности вместо try..catch лучше использовать условные конструкции на проверку исключительных ситуаций.

**67 Вычислите приближенно значение бесконечной суммы с точностью до е:**



**78 Решите задачу. Дан двумерный массив. Заполните его по строкам с клавиатуры и определите:**

* **количество столбцов, не содержащих ни одного нулевого элемента;**
* **минимальное из чисел, в заданном столбце массива.**

**Список использованных источников**

1. Албахари, Д. C# 5.0. Справочник. Полное описание языка / Д.Албахари. – М.: Вильямс, 2014.
2. Лабор, В.В. Си Шарп: Создание приложений для Windows / В.В.Лабор. - Мн.: Харвест, 2003.
3. Павловская, Т.А С#. Программирование на языке высокого уровня / Т.А.Павловская. - СПб: Питер, 2014.
4. Фролов, А.В. Визуальное проектирование приложений С# / А.В.Фролов. - М: КУДИЦ - ОБРАЗ, 2003.
5. Фленов, М. Библия С# / М.Фленов. - СПб.: Питер, 2011.