# 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.

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:

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:

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:

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

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:

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:

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.

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

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:

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:**



**Блок-схема алгоритма решения задачи.**

**Текст программы.**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp67

{

class Program

{

static void Main(string[] args)

{

double s=0;// значение функции

double sp = 0;// предыдущее значение функции

int x = 2;// начальное значение

int step = 1;// шаг

double c = 0;

Console.WriteLine("Введите нужную точность:");

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

c = x \* (x + 1);

s = s + 1 / c; //вычисляем значение функции

Console.WriteLine("ШАГ={0} S={1} SP={2} SP-S={3}", x - 1, s, sp, sp - s);// выводим результат

x += step;// приращаем аргумент шага

sp = s;

do

{

c = x \* (x + 1);

sp = s;

s = s+ 1/c; //вычисляем значение функции

Console.WriteLine("ШАГ={0} S={1} delta={2}", x-1, s,Math.Abs(sp-s) );// выводим результат

x += step;// приращаем аргумент шага

}

while (Math.Abs(sp-s) > e);// проверяем условие окончания

Console.WriteLine("Работа завершена, нажмите ВВОД");

Console.Read();

}

}

}

**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.**

**Блок-схема алгоритма решения задачи.**

**Текст программы.**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp78

{

class Program

{

static void Main(string[] args)

{

const int n = 2; //число строк

const int m = 3; //число столбцов

int bd;

int[,] A = new int[n, m];//объявление массива

Console.WriteLine("Размер масива: [{0}x{1}]", n, m);

for (int i = 0; i < n; i++)

{

for (int j = 0; j < m; j++)

{

Console.WriteLine("Заполните элемент массива A[{0},{1}]", i + 1, j + 1);

A[i, j] = Int32.Parse(Console.ReadLine());

}

}

// вывод массива

for (int i = 0; i < n; i++)

{// перебираем столбцы

for (int j = 0; j < m; j++)

{// перебираем элементы столбца

Console.Write(A[i, j]);

Console.Write(" ");

}

Console.WriteLine("");

}

// поиск количества столбцов без 0

int s=0;// количество

for (int j = 0; j < m; j++)

{// перебираем столбцы

bd = 1;

for (int i = 0; i < n ; i++)

{// перебираем элементы столбца

if (A[i, j] == 0)

{// если есть 0

bd=0;

}

}

s += bd;// добавляем ненулевой столбец

}

Console.WriteLine("Столбцов без нулей: {0}",s);

Console.WriteLine("Укажите столбец: ");

s = Int32.Parse(Console.ReadLine())-1;// столбец для поиска минимума

int mn = A[0, s];

for (int i = 0; i < n; i++)

{

if (A[i, s] < mn) { mn = A[i, s]; }// обновляем минимум если надо

}

Console.WriteLine("Минимальный элемент: {0}", mn);

Console.WriteLine("Нажмите ВВОД");

Console.ReadLine();

}

}

}

**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.