Практическая работа № 1.  
Основы языка C#

**Цель практической работы**

Закрепление теоретических знаний по разработке модульных тестов к программам.

**Постановка задачи**

**Форматированный вывод**

**Рассмотрим на примерах**.

У нас есть некоторая переменная int value = 5;

и мы хотим получить строку вида: **«Васе value лет» («Васе 5 лет»),**

Для этого есть ряд способов:

string str = «Васе » + value + « лет»; **Конкатенация строк**

**Недостатки:** при каждом вызове оператора конкатенации «+», в памяти выделяется место под новую строку, новая строка записывается в новую ячейку памяти, а старая строка уничтожается. Много «+» => очень медленная работа программы.

int value = 5;

string str = string.Format("Васе {0} лет", value);

Примечание:

Если строку нужно вывести в консоль, то string.Format можно не писать:

Console.WriteLine("Васе {0} лет", value);

Современный способ форматированного вывода:

string str = $"Васе {value} лет"; // Интерполяция строк (знак доллара)

**Пример 2:**

Мы хотим получить строку вида: **«Васе age лет, у него money рублей»**

int age = 25;

int money = 20000;

string str = string.Format(«Васе {0} лет, у него {1} руб-лей», age, money);

string str = $"Васе {age} лет, у него {money} рублей";

**Пример 3:**

Мы хотим получить строку вида: **«Васе age лет, у него money рублей»,** но тип **money** вещественный и нам нужно только 2 знака.

int age = 25;

double money = 2.15367521E4; //21536.7521

string str = string.Format(«Васе {0} лет, у него {1 : f2} рублей», age, money);

string str = $"Васе {age} лет, у него {money : f2} рублей";

d – целые числа; f – дробные; c – денежный формат;

e – экспоненциальная форма записи;

g – выбирает наиболее короткий из f и e.

ИЛИ

string str = $"Васе {age} лет, у него {money : 0.00} рублей";

Получим строку вида: **«Васе 25 лет, у него 21536.75 рублей»**

**Перечислимые константы (перечисления)** – это тип значения, определенный набором именованных констант применяемого целочисленного типа. Чтобы определить тип перечисления, используйте ключевое слово **enum** и укажите имена:

enum Transport { Car, Airplane, Bicycle, Boat, Spaceship }

Transport myTransport = Transport.Car;

enum DayTime { Morning, Afternoon, Evening, Night}

DayTime dayTime = DayTime.Evening;

**Строки.**

Со строками можно работать как с массивом символов, т.е:

string str = "Это строка";

Можно обратиться к символу строки как к элементу массива:

Console.WriteLine(str[0]); //Ответ: ‘Э’

Console.WriteLine(str[3]); //Ответ: ‘ ’

Можно узнать длину строки с помощью свойства Length:

Console.WriteLine(str.Length); //Ответ: 10

Однако, строки – неизменяемый тип данных, заменить отдельный символ на другой нельзя!

Существуют методы «замены» символа на другой, но они создают новую строку:

str = str.Replace(‘о’, ‘а’); //Ответ: “Эта страка”

**Условные конструкции**

Условные конструкции - один из базовых компонентов многих языков программирования, которые направляют работу программы по одному из путей в зависимости от определенных условий.

В языке C# используются следующие условные конструкции: if..else и switch..case

**Конструкция if/else**

Конструкция if/else проверяет истинность некоторого условия и в зависимости от результатов проверки выполняет определенный код:

int num1 = 8;

int num2 = 6;

if(num1 > num2)

{

Console.WriteLine($"Число {num1} больше числа {num2}");

}

После ключевого слова if ставится условие. И если это условие выполняется, то срабатывает код, который помещен далее в блоке if после фигурных скобок. В качестве условий выступают ранее рассмотренные операции сравнения.

В данном случае у нас первое число больше второго, поэтому выражение num1 > num2 истинно и возвращает true, следовательно, управление переходит к строке Console.WriteLine("Число {num1} больше числа {num2}");

Но что, если мы захотим, чтобы при несоблюдении условия также выполнялись какие-либо действия? В этом случае мы можем добавить блок else:

int num1 = 8;

int num2 = 6;

if(num1 > num2)

{

Console.WriteLine($"Число {num1} больше числа {num2}");

}

else

{

Console.WriteLine($"Число {num1} меньше числа {num2}");

}

Но при сравнении чисел мы можем насчитать три состояния: первое число больше второго, первое число меньше второго и числа равны. Используя конструкцию else if, мы можем обрабатывать дополнительные условия:

int num1 = 8;

int num2 = 6;

if(num1 > num2)

{

Console.WriteLine($"Число {num1} больше числа {num2}");

}

else if (num1 < num2)

{

Console.WriteLine($"Число {num1} меньше числа {num2}");

}

else

{

Console.WriteLine("Число num1 равно числу num2");

}

Также мы можем соединить сразу несколько условий, используя логические операторы:

int num1 = 8;

int num2 = 6;

if(num1 > num2 && num1==8)

{

Console.WriteLine($"Число {num1} больше числа {num2}");

}

В данном случае блок if будет выполняться, если num1 > num2 равно true и num1==8 равно true.

**Конструкция switch**

Конструкция **switch/case** аналогична конструкции if/else, так как позволяет обработать сразу несколько условий:

Console.WriteLine("Нажмите Y или N");

string selection = Console.ReadLine();

switch (selection)

{

case "Y":

Console.WriteLine("Вы нажали букву Y");

break;

case "N":

Console.WriteLine("Вы нажали букву N");

break;

default:

Console.WriteLine("Вы нажали неизвестную букву");

break;

}

После ключевого слова **switch** в скобках идет сравниваемое выражение. Значение этого выражения последовательно сравнивается со значениями, помещенными после оператора **сase**. И если совпадение будет найдено, то будет выполняться определенный блок **сase**.

В конце каждого блока сase должен ставиться один из операторов перехода: **break**, **return** или **throw**. Как правило, используется оператор **break**. При его применении другие блоки case выполняться не будут.

int number = 1;

switch (number)

{

case 1:

Console.WriteLine("case 1");

case 3:

Console.WriteLine("case 3");

break;

case 5:

Console.WriteLine("case 5");

break;

default:

Console.WriteLine("default");

break;

}

Если мы хотим также обработать ситуацию, когда совпадения не будет найдено, то можно добавить блок **default**, как в примере выше.

Применение оператора return позволит выйти не только из блока case, но и из вызывающего метода. То есть, если в методе Main после конструкции switch..case, в которой используется оператор return, идут какие-либо операторы и выражения, то они выполняться не будут, а метод Main завершит работу.

Оператор throw применяется для выброса ошибок и будет рассмотрен в одной из следующим тем.

**Тернарная операция**

Тернарную операция имеет следующий синтаксис: [первый операнд - условие] ? [второй операнд] : [третий операнд]. Здесь сразу три операнда. В зависимости от условия тернарная операция возвращает второй или третий операнд: если условие равно true, то возвращается второй операнд; если условие равно false, то третий. Например:

int x=3;

int y=2;

Console.WriteLine("Нажмите + или -");

string selection = Console.ReadLine();

int z = selection=="+"? (x+y) : (x-y);

Console.WriteLine(z);

Здесь результатом тернарной операции является переменная z. Если мы выше вводим "+", то z будет равно второму операнду - (x+y). Иначе z будет равно третьему операнду.

**Циклы**

Циклы являются управляющими конструкциями, позволяя в зависимости от определенных условий выполнять некоторое действие множество раз. В C# имеются следующие виды циклов:

* for
* foreach
* while
* do...while

**Цикл for**

Цикл for имеет следующее формальное определение:

for ([инициализация счетчика]; [условие]; [изменение счетчика])

{

// действия

}

Рассмотрим стандартный цикл for:

for (int i = 0; i < 9; i++)

{

Console.WriteLine($"Квадрат числа {i} равен {i\*i}");

}

Первая часть объявления цикла - int i = 0 - создает и инициализирует счетчик i. Счетчик необязательно должен представлять тип **int**. Это может быть и другой числовой тип, например, float. И перед выполнением цикла его значение будет равно 0. В данном случае это то же самое, что и объявление переменной.

Вторая часть - условие, при котором будет выполняться цикл. Пока условное выражение возвращает true, будет выполняться цикл. В данном случае цикл будет выполняться, пока счетчик i не достигнет 9.

И третья часть - приращение счетчика на единицу. Опять же нам необязательно увеличивать на единицу. Можно уменьшать: i--.

В итоге блок цикла сработает 9 раз, пока значение i не станет равным 9. И каждый раз это значение будет увеличиваться на 1.

Нам необязательно указывать все условия при объявлении цикла. Например, мы можем написать так:

int i = 0;

for (; ;)

{

Console.WriteLine($"Квадрат числа {++i} равен {i \* i}");

}

Формально определение цикла осталось тем же, только теперь блоки в определении у нас пустые: for (; i <;). У нас нет инициализированной переменной-счетчика, нет условия, поэтому цикл будет работать вечно - бесконечный цикл.

Мы также можем опустить ряд блоков:

int i = 0;

for (; i<9;)

{

Console.WriteLine($"Квадрат числа {++i} равен {i \* i}");

}

Этот пример по сути эквивалентен первому примеру: у нас также есть счетчик, только создан он вне цикла. У нас есть условие выполнения цикла. И есть приращение счетчика уже в самом блоке for.

**Цикл do**

В цикле do сначала выполняется код цикла, а потом происходит проверка условия в инструкции while. И пока это условие истинно, цикл повторяется. Например:

int i = 6;

do

{

Console.WriteLine(i);

i--;

}

while (i > 0);

Здесь код цикла сработает 6 раз, пока i не станет равным нулю. Но важно отметить, что цикл do гарантирует хотя бы единократное выполнение действий, даже если условие в инструкции while не будет истинно. То есть мы можем написать:

int i = -1;

do

{

Console.WriteLine(i);

i--;

}

while (i > 0);

Хотя у нас переменная i меньше 0, цикл все равно один раз выполнится.

**Цикл while**

В отличие от цикла do цикл **while** сразу проверяет истинность некоторого условия, и если условие истинно, то код цикла выполняется:

int i = 6;

while (i > 0)

{

Console.WriteLine(i);

i--;

}

**Операторы continue и break**

Иногда возникает ситуация, когда требуется выйти из цикла, не дожидаясь его завершения. В этом случае мы можем воспользоваться оператором **break**.

Например:

for (int i = 0; i < 9; i++)

{

if (i == 5)

break;

Console.WriteLine(i);

}

Хотя в условии цикла сказано, что цикл будет выполняться, пока счетчик i не достигнет значения 9, в реальности цикл сработает 5 раз. Так как при достижении счетчиком i значения 5, сработает оператор break, и цикл завершится.

Теперь поставим себе другую задачу. А что если мы хотим, чтобы при проверке цикл не завершался, а просто пропускал текущую итерацию. Для этого мы можем воспользоваться оператором **continue**:

for (int i = 0; i < 9; i++)

{

if (i == 5)

continue;

Console.WriteLine(i);

}

В этом случае цикл, когда дойдет до числа 5, которое не удовлетворяет условию проверки, просто пропустит это число и перейдет к следующей итерации:

**Сложность алгоритма** – оценка количества базовых действий, необходимых для выполнения алгоритма.

Если требуется **n, 2n, … kn**, действий, то говорят, что сложность алгоритма **O(n)**, где k – некоторая константа, не зависящая от числа действий.

**Пример 1.**

var n = int.Parse(Console.ReadLine());

var sum = 0.0;

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

{

sum += n;

}

Console.WriteLine(sum);

Сложность алгоритма **O(n)**

**Пример 2.**

var n = int.Parse(Console.ReadLine());

var sum = 0.0;

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

{

for (var j = 0; j < n; j++)

{

sum += n;

}

}

Сложность алгоритма **O(n2)**

**Задание на практическую работу**

1. Разработать консольные приложения для решения задач согласно варианту.
2. Предусмотреть понятный интерфейс ввода и вывода данных на консоль.

**Задание 1.**

По заданному номеру месяца в году требуется определить время года.

**Входные данные**

Входная строка содержит натуральное число N (N≤100) – номер месяца.

**Выходные данные**

Выведите для летних месяцев значение «Summer», для зимних – «Winter», для весенних – «Spring», для осенних – «Autumn». Если число не соответствует возможному значению месяца, то в этом случае следует вывести «Error».

**Примеры**

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT** | **OUTPUT** |
| 1 | 3 | Spring |
| 2 | 15 | Error |

**Задание 2.**

Однажды известный профессор обнаружил описания k конечных автоматов. По его мнению, нетривиальность конечного автомата, имеющего n состояний и m переходов, можно описать целым числом d = 19m + (n + 239)\*(n + 366) / 2 . Чем больше d, тем больший интерес для науки представляет изучение его свойств.

Помогите профессору вычислить нетривиальность имеющихся у него автоматов.

**Входные данные**

Первая строка содержит целое число k (1 ≤ k ≤ 10000) – количество конечных автоматов. Следующие k строк содержат по два целых числа ni (0 ≤ ni ≤ 1000) и mi (0 ≤ mi ≤ 26ni2) – число состояний и переходов i-го автомата.

**Выходные данные**

Выходная строка должна состоять из k строк. На i-й строке выведите одно число – нетривиальность i-го автомата.

**Примеры**

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT** | **OUTPUT** |
| 1 | 4 2 0 13 20 5 23 18 6 | 44344 48134 45699 49458 |
| 2 | 2 15 20 1000 26000 | 48767 1340237 |

**Задание 3.**

Требуется написать программу, определяющую, является ли четырехзначное натуральное число N палиндромом, т.е. числом, которое одинаково читается слева направо и справа налево.

**Входные данные**

Входная строка содержит натуральное число N (1000 ≤ N ≤ 9999).

**Выходные данные**

Следует вывести слово «YES», если число N является палиндромом, или «NO» – если нет.

**Примеры**

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT** | **OUTPUT** |
| 1 | 6116 | YES |
| 2 | 1231 | NO |

**Пример разработки приложения**

**Задание 1.**

По заданному номеру месяца в году требуется определить время года.

**Входные данные**

Входная строка содержит натуральное число N (N≤100) – номер месяца.

**Выходные данные**

Выведите для летних месяцев значение «Summer», для зимних – «Winter», для весенних – «Spring», для осенних – «Autumn». Если число не соответствует возможному значению месяца, то в этом случае следует вывести «Error».

**Примеры**

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT** | **OUTPUT** |
| 1 | 3 | Spring |
| 2 | 15 | Error |

**Шаг 1: Понимание задачи**

Нам нужно написать программу, которая по номеру месяца (целое число от 1 до 12) определяет время года. Если введённое число не входит в этот диапазон, программа должна сообщить об ошибке.

**Шаг 2: Определение времён года**

Обычно времена года распределяются по месяцам следующим образом:

* **Зима (Winter)**: декабрь (12), январь (1), февраль (2)
* **Весна (Spring)**: март (3), апрель (4), май (5)
* **Лето (Summer)**: июнь (6), июль (7), август (8)
* **Осень (Autumn)**: сентябрь (9), октябрь (10), ноябрь (11)

**Шаг 3: Анализ входных данных**

На вход подаётся число N (1 ≤ N ≤ 100). Однако месяцы имеют номера только от 1 до 12, поэтому если N < 1 или N > 12, нужно вывести "Error".

**Шаг 4: Реализация логики**

Используем условные операторы (if, else if, else) для проверки диапазонов номеров месяцев и определения времени года.

**Шаг 5: Написание кода на C#**

// Шаг 1: Чтение входных данных

Console.Write("Введите номер месяца: ");

int month = int.Parse(Console.ReadLine());

// Шаг 2: Проверка на корректность номера месяца

if (month < 1 || month > 12)

{

Console.WriteLine("Error");

return;

}

// Шаг 3: Определение времени года

if (month == 12 || month == 1 || month == 2)

{

// Доделать самостоятельно

}

else if (month >= 3 && month <= 5)

{

// Доделать самостоятельно

}

else if (month >= 6 && month <= 8)

{

// Доделать самостоятельно

}

else if (month >= 9 && month <= 11)

{

// Доделать самостоятельно

}

**Шаг 6: Тестирование**

Проверим программу на примерах из условия:

1. **Вход**: 3 → **Ожидаемый вывод**: "Spring" (март — весна).
2. **Вход**: 15 → **Ожидаемый вывод**: "Error" (нет такого месяца).

# Содержание пояснительной записки

1. Постановка задачи. Приводится теоретический материал, использованный при написании приложения.

2. Формулировка задания и вариант. Приводится задание на лабораторную работу и вариант этого задания.

3. Описание выполняемых действий. Необходимо привести описание последовательности разработки программы, реализации используемых методов, алгоритмов, блок-схем.

4. Анализ результатов. Привести анализ входных и выходных данных. Показать результаты выполнения программного кода. Предоставить скриншоты обработки тестовых примеров. Сделать выводы.

5. Листинг программы. Привести листинг разработанного программного кода, содержание файлов входных и выходных данных.

# Используемое программное обеспечение

1. Среда программирования MS Visual Studio Community 2022 (Свободно распространяемое программное обеспечение (в учебных целях));
2. Microsoft Office Standard 2007 (Open License: 42267924);
3. Open Office (Свободно распространяемое программное обеспечение).
4. Браузер (Свободно распространяемое программное обеспечение).

# Список литературы

* + - 1. Мейер Б. Объектно-ориентированное программирование и программная инженерия [Электронный ресурс]/ Мейер Б. – Электрон. текстовые данные. – М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. – 285 c.
      2. Биллиг, В. A. Основы объектного программирования на С# (C# 3.0, Visual Studio 2008) [Электронный ресурс] : учебное пособие / В. A. Биллиг. — Электрон. текстовые данные. — Москва, Саратов : Интернет-Университет Информационных Технологий (ИНТУИТ), Вузовское образование, 2017. — 583 c. — 978-5-4487-0145-0. — Режим доступа: http://www.iprbookshop.ru/72339.html
      3. Павловская, Т. А. Программирование на языке высокого уровня C# [Электронный ресурс] / Т. А. Павловская. — 2-е изд. — Электрон. текстовые данные. — М. : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 245 c. — 2227-8397. — Режим доступа: http://www.iprbookshop.ru/73713.html
      4. Агапов, В. П. Основы программирования на языке С# [Электронный ресурс] : учебное пособие / В. П. Агапов. — Электрон. текстовые данные. — М. : Московский государственный строительный университет, ЭБС АСВ, 2012. — 128 c. — 978-5-7264-0576-6. — Режим доступа: http://www.iprbookshop.ru/16366.html
      5. Медведев, М. А. Программирование на СИ# [Электронный ресурс] : учебное пособие / М. А. Медведев, А. Н. Медведев ; под ред. А. В. Присяжный. — Электрон. текстовые данные. — Екатеринбург : Уральский федеральный университет, ЭБС АСВ, 2015. — 64 c. — 978-5-7996-1561-1. — Режим доступа: http://www.iprbookshop.ru/69667.html
      6. Казанский А.А. Объектно-ориентированное программирование на языке Microsoft Visual С# в среде разработки Microsoft Visual Studio 2008 и .NET Framework. 4.3 [Электронный ресурс]: учебное пособие и практикум/ Казанский А.А.— Электрон. текстовые данные.— М.: Московский государственный строительный университет, ЭБС АСВ, 2011.— 180 c
      7. Уйманова Н.А. Основы объектно-ориентированного программирования [Электронный ресурс]: практикум/ Уйманова Н.А., Таспаева М.Г.— Электрон. текстовые данные.— Оренбург: Оренбургский государственный университет, ЭБС АСВ, 2017.— 156 c.— Режим доступа: http://www.iprbookshop.ru/78808.html.— ЭБС «IPRbooks»
      8. Новиков П.В. Объектно-ориентированное программирование [Электронный ресурс]: учебно-методическое пособие к лабораторным работам/ Новиков П.В.— Электрон. текстовые данные.— Саратов: Вузовское образование, 2017.— 124 c.— Режим доступа: http://www.iprbookshop.ru/64650.html.— ЭБС «IPRbooks»