# Exercises: Operators Expressions and Statements

This document defines in-class exercises from the ["C# Basics" Course @ Software University](http://softuni.bg/courses/csharp-basics/).

### Arithmetic Operators

## Average

Write a program that finds the **average** of the **sum of 3** numbers.

*Hints:*

1. Declare ***four*** variables (*a*, *b*, *c* and *average*).
2. **Read** the user input from the console. (int.Parse(Console.ReadLine());).
3. **Calculate** the **average value** of the variables by the **formulae** (a
4. **Print** the result on the console (Console.WriteLine(average));).

|  |  |  |  |
| --- | --- | --- | --- |
| **a** | **b** | **c** | **Average** |
| 45 | 41 | 20 | 35.33333 |
| 22 | 52 | 60 | 44.66667 |

## Trapezoid

Write a program that finds the **area** of a trapezoid, given the base sides **a**, **b** and height **h**.

*Hints:*

1. Declare ***four*** variables (*a*, *b*, *h* and *area*).
2. **Read** the user input from the console. (int.Parse(Console.ReadLine());).
3. **Calculate** the **area** **of the trapezoid** by the **formulae** (
4. **Print** the result on the console (Console.WriteLine(area));).

|  |  |  |  |
| --- | --- | --- | --- |
| **a** | **b** | **h** | **Area** |
| 5 | 2 | 4 | 14 |
| 8.5 | 4.4 | 2 | 12.9 |

## Last Digit

Write a program that prints the last digit of a number **n**.

*Hints:*

1. Declare ***two*** variables (*n* and *lastDigit*).
2. **Read** the user input from the console. (int.Parse(Console.ReadLine());).
3. **Find** the **last digit** of the **number** by the **formulae** (The word ***mod*** means modular division (or the operator **%** in C#).
4. **Print** the result on the console (Console.WriteLine(lastDigit));).

|  |  |
| --- | --- |
| **n** | **Result** |
| 21 | 1 |
| 139 | 9 |
| 4 | 4 |

## N-th Digit

Write a program that prints the **n**-th digit of a number (from right to left). If no such digit exists, print a dash "**-**".

*Hints:*

1. Declare ***three*** variables (*number*, *n* and *nDigit*).
2. **Read** the user input from the console. (int.Parse(Console.ReadLine());).
3. **Find** the **n-th digit** of the **number** by using the **formulae** ( The word ***mod*** means modular division (or the operator **%** in C#).
4. **Print** the result on the console (Console.WriteLine(area));).

|  |  |  |
| --- | --- | --- |
| **Number** | **n** | **Result** |
| 2174 | 3 | 1 |
| 169 | 2 | 6 |
| 46 | 4 | - |

### Logical Operators

## Big and Odd

Write a program that that prints if the number is both **greater than 20** and **odd**.

*Hints:*

1. Declare ***two*** variables (*n* and *result*).
2. **Read** the user input from the console. (int.Parse(Console.ReadLine());).
3. **Check** if the input **number** is **greater** than **20** and **odd** by using the **logical operators**:
   1. **>** or **<** checks if the **value** **on the** **left** of the operator is **greater/less** than the **value on the right** side of the operator;
   2. Using the **formulae** you check whether the entered number **is odd**. The word ***mod*** means modular division (or the operator **%** in C#);
   3. **&&** checks if the **left** **expression** **AND** the **right expression** both have **true** values;
   4. **Save** the **result** of the verification in the ***result*** variable;
4. **Print** the result on the console (Console.WriteLine(result));).

|  |  |
| --- | --- |
| **n** | **Result** |
| 63 | true |
| 17 | false |
| 22 | false |
| 23 | true |
| 20 | false |

## Pure Divisor

Write a program that prints if a number is **divided** by 9, 11 or 13 **without remainder**.

*Hints:*

1. Declare ***two*** variables (*n* and *result*).
2. **Read** the user input from the console. (int.Parse(Console.ReadLine());).
3. **Check** if the input **number** is **divided** by **9**, **11** or **13** using the **logical operators**:
   1. Using the **formulae:** OR OR)  
      you check whether the entered number **is divided** bythegiven **constants without remainder**. The word ***mod*** means modular division (or the operator **%** in C#);
   2. **||** checks if the **left** **expression** **OR** the **right expression** have **a** **true** value. If **only** **one** has a **true** value the result is **true**;
   3. **Save** the **result** of the verification in the ***result*** variable;
4. **Print** the result on the console (Console.WriteLine(result));).

|  |  |
| --- | --- |
| **n** | **Result** |
| 121 | true |
| 1263 | false |
| 26 | true |
| 23 | false |
| 81 | true |
| 1287 | true |

### Bitwise Operators

## First Bit

Write a program that prints the bit at **position 1** of a number.

*Hints:*

1. Declare ***two*** variables (*n* and *bitAtPosition1*).
2. **Read** the user input from the console. (int.Parse(Console.ReadLine());).
3. **Find** the **value** of the **bit at position 1** (position 1 is the second bit from right to left: [7, 6, 5, 4, 3, 2, **1**, 0]):
   1. **Shift** the number ***n*** times to the **right** (where ***n*** is the position, in this case it is **1**) by using the **>>** operator. In that way the bit we want to check will be at position **0**;
   2. **Find** the bit at **position 0**. Use **& 1** operator expression to extract the value of a bit. By using the following **formulae** () you **check** whether the bit at **position 0** is equal to **1** or **not**. If the bit is **equal** to **1** the **result** is **1** if the bit is **not** **equal** - the **result** is **0**;
   3. **Save** the result in **bitAtPosition1;**
4. **Print** the result on the console (Console.WriteLine(bitAtPostition1));).

|  |  |
| --- | --- |
| **n** | **Result** |
| 2 | 1 |
| 51 | 1 |
| 13 | 0 |
| 24 | 0 |

## p-th Bit

Write a program that prints the bit at position **p** of a number.

*Hints:*

1. Declare ***three*** variables (*n, p* and *bitAtPositionP*).
2. **Read** the user input from the console. (int.Parse(Console.ReadLine());).
3. **Find** the **value** of the **bit at position p**:
   1. **Shift** the number ***p*** times to the **right** (where ***p*** is the position) by using the **>>** operator. In that way the bit we want to check will be at position **0**;
   2. **Find** the bit at **position 0**. Use **& 1** operator expression to extract the value of a bit. By using the following **formulae** () you **check** whether the bit at **position 0** is equal to **1** or **not**. If the bit is **equal** to **1** the **result** is **1** if the bit is **not** **equal** - the **result** is **0**;
   3. **Save** the result in **bitAtPosition1;**
4. **Print** the result on the console (Console.WriteLine(bitAtPostitionP));).

|  |  |  |
| --- | --- | --- |
| **n** | **p** | **Result** |
| 2145 | 5 | 1 |
| 512 | 0 | 0 |
| 111 | 8 | 0 |
| 255 | 7 | 1 |

## Bit Destroyer

Write a program that sets the bit at **position** **p** to **0**. Print the resulting number.

*Hints:*

1. Declare ***four*** variables (*n, p, mask* and *newNumber*).
2. **Read** the user input from the console. (int.Parse(Console.ReadLine());).
3. **Set** the **value** of the **bit at position p** to **0**:
   1. **Shift** the number **1**, ***p*** times to the **left** (where ***p*** is the position) by using the **<<** operator. In that way the bit we want to delete will be at position **p**. Save the resulting value in ***mask***;
   2. **Invert** the **mask** (e.g. we move the number 1, 3 times and we get 00001000, after inverting we get 11110111).
   3. Use **& mask** operator expression to **set** the **value** of a number to **0**. By using the following **formulae** () you **copy** **all** the **bits** of the **number** and you **set** the bit at **position** **p** to **0**;
   4. **Save** the result in **newNumber;**
4. **Print** the result on the console (Console.WriteLine(newNumber));).

|  |  |  |
| --- | --- | --- |
| **n** | **p** | **Result** |
| 1313 | 5 | 1281 |
| 231 | 2 | 227 |
| 111 | 6 | 47 |

## \* Tri-bit Switch

Write a program that inverts the **3 bits** from position **p** to the left with their opposites (e.g. **111** -> **000**, **101** -> **010**). Print the resulting number on the console.

*Hint: Use the following code and complete the comments:*

|  |
| --- |
| long n = long.Parse(Console.ReadLine());  int p = int.Parse(Console.ReadLine());  long mask =  //TODO: Make the mask needed to invert 3 bits at position p  ;  long result =  //TODO: Use the ^ (XOR) operation with the number and the mask to get the result  ;  Console.WriteLine(result); |

1. **Shift** the number **7** (the number 7 has the bits 111 which we use in order to get 3 consecutive values), ***p*** times to the **left** (where ***p*** is the position) by using the **<<** operator. In that way the **3 bits** we want to **invert** will be at position **p**. Save the resulting value in ***mask***;
2. Use **^ mask** operator expression to **invert** the **values** of the **three** **bits** starting from position **p**. By using the following **formulae** (*n ^ mask*) you **copy** all the **bits** of the **number** and you **invert** the bits at position **p**, **p+1** and **p+2**;
3. Save the result in **result**;

|  |  |  |  |
| --- | --- | --- | --- |
| **n** | **p** | **Result** | **Details** |
| 3652421623 | 7 | 3652420727 | **1101 1001 1011 0011 1000 0111 1111 0111**  **1101 1001 1011 0011 1000 0100 0111 0111** |
| 2135693832857285912 | 59 | 2712154585160709400 | **0001 1101 1010 0011 1000 0010 0011 1001 1100 0000 0111 0011 0111 0101 0001 1000**  **0010 0101 1010 0011 1000 0010 0011 1001 1100 0000 0111 0011 0111 0101 0001 1000** |