# Homework: Primitive Data Types and Variables

This document defines homework assignments from the [“C# Basics“ Course @ Software University](http://softuni.bg/courses/csharp-basics/). Please submit as homework a single zip / rar / 7z archive holding the solutions (source code only) of all below described problems.

## Declare Variables

Declare five variables choosing for each of them the most appropriate of the types byte, sbyte, short, ushort, int, uint, long, ulong to represent the following values: 52130, -115, 4825932, 97, -10000. Choose a large enough type for each number to ensure it will fit in it. Try to compile the code. Submit the source code of your Visual Studio project as part of your homework submission.

## Float or Double?

Which of the following values can be assigned to a variable of type float and which to a variable of type double: 34.567839023, 12.345, 8923.1234857, 3456.091? Write a program to assign the numbers in variables and **print** them to ensure no precision is lost.

## Variable in Hexadecimal Format

Declare an integer variable and assign it with the value 254 in hexadecimal format (0x##). Use Windows Calculator to find its hexadecimal representation. Print the variable and ensure that the result is “254”.

## Unicode Character

Declare a character variable and assign it with the symbol that has Unicode code 42 (decimal) using the '\u00XX' syntax, and then print it. Hint: first, use the Windows Calculator to find the hexadecimal representation of 42. The output should be “\*”.

## Boolean Variable

Declare a Boolean variable called isFemale and assign an appropriate value corresponding to **your gender**. Print it on the console.

## Strings and Objects

Declare two **string variables** and assign them with “Hello” and “World”. Declare an **object variable** and assign it with the **concatenation** of the first two variables (mind adding an interval between). Declare a third string variable and initialize it with the value of the object variable (you should perform type **casting**).

## Quotes in Strings

Declare two string variables and assign them with following value:

|  |
| --- |
| The "use" of quotations causes difficulties. |

Do the above in two different ways: with and without using **quoted strings**. Print the variables to ensure that their value was correctly defined.

## Isosceles Triangle

Write a program that prints an isosceles triangle of 9 copyright symbols ©, something like this:

|  |
| --- |
| ©  © ©  © ©  © © © © |

Note that the © symbol may be displayed incorrectly at the console so you may need to change the console character encoding to UTF-8 and assign a Unicode-friendly font in the console. Note also, that under old versions of Windows the © symbol may still be displayed incorrectly, regardless of how much effort you put to fix it.

## Exchange Variable Values

Declare two integer variables a and b and assign them with 5 and 10 and after that exchange their values by using some programming logic. Print the variable values before and after the exchange.

## Employee Data

A marketing company wants to keep record of its employees. Each record would have the following characteristics:

* First name
* Last name
* Age (0...100)
* Gender (m or f)
* Personal ID number (e.g. 8306112507)
* Unique employee number (27560000…27569999)

Declare the variables needed to keep the information for a single employee using appropriate primitive data types. Use descriptive names. **Print** the data at the console.

## Bank Account Data

A bank account has a holder name (first name, middle name and last name), available amount of money (balance), bank name, IBAN, 3 credit card numbers associated with the account. Declare the variables needed to keep the information for a single bank account using the appropriate data types and descriptive names.

## Null Values Arithmetic

Create a program that assigns null values to an integer and to a double variable. Try to print these variables at the console. Try to add some number or the null literal to these variables and print the result.

## \* Comparing Floats

Write a program that **safely compares floating-point numbers** (double) with precision eps = 0.000001. Note that we cannot directly compare two floating-point numbers a and b by a==b because of the nature of the floating-point arithmetic. Therefore, we assume two numbers are equal if they are more closely to each other than a fixed constant eps. Examples:

|  |  |  |  |
| --- | --- | --- | --- |
| **Number a** | **Number b** | **Equal (with precision eps=0.000001)** | **Explanation** |
| 5.3 | 6.01 | false | The difference of 0.71 is too big (> eps) |
| 5.00000001 | 5.00000003 | true | The difference 0.00000002 < eps |
| 5.00000005 | 5.00000001 | true | The difference 0.00000004 < eps |
| -0.0000007 | 0.00000007 | true | The difference 0.00000077 < eps |
| -4.999999 | -4.999998 | false | Border case. The difference 0.000001 == eps. We consider the numbers are different. |
| 4.999999 | 4.999998 | false | Border case. The difference 0.000001 == eps. We consider the numbers are different. |

## \* Print the ASCII Table

Find online more information about [**ASCII**](http://www.ascii-code.com/) (American Standard Code for Information Interchange) and write a program to prints the entire ASCII table of characters at the console (characters from 0 to 255). Note that some characters have a special purpose and will not be displayed as expected. You may skip them or display them differently. You may need to use for-loops (learn in Internet how).

# Exam problems.\*\* Variant 1

All of the problems below are given from the previous C# Basics exams. **You are not obligated** to submit any of them in your homework. We highly recommend you to try solving some or all of them so you can be well prepared for the upcoming exam. You need to learn how to use conditional statements, loops, arrays and other things (learn in internet how or read those chapters in the book “[Fundamentals of computer programming with C#](http://www.introprogramming.info/intro-csharp-book/read-online/)”). If you still find those problems too hard for solving it’s very useful to **check** and **understand** the solutions. You can download all solutions and tests for this variant [here](https://softuni.bg/downloads/svn/csharp-basics/Feb-2014/9.%20CSharp-Basics-Exam-April-2014-Variant-1.zip) or check all [previous exams](https://softuni.bg/trainings/coursesinstances/details/2) (scroll down to the bottom of the page). You can also test your solutions in our automated [judge system](http://judge.softuni.bg/Contests/2/CSharp-Basics-Exam-10-April-2014-Morning) to see if you pass all tests.

## 1 - Joro, the Football Player

Joro loves a lot to play football. He used to play for his local village club “Pantera” Kaloyanovets. However, he is a programmer now and he is very busy. Now he is able to **play only in the holidays** and in the **weekends**. Joro plays in **1/2 of the holidays** and **twice in the weekends**: each **Saturday** and each **Sunday**, but **not every weekend** – only when he is **not tired** and only when he is **not going to his hometown**. Joro goes at his hometown **h** weekends in the year. The other weekends are considered “**normal**”. Joro is **tired in 1/3 of the normal weekends**. When Joro is at his hometown, he always plays football with his old friends once, at **Sunday**. In addition, if the year is **leap**, Joro plays football **3 more times** additionally, in non-weekend days. We assume the year has **exactly 52 weekends**.

Your task is to write a program that calculates **how many times Joro plays football** (rounded down to the nearest integer number).

### Input

The input data should be read from the console. It consists of three input values, each at separate line:

* The string “**t**” for leap year or “**f**” for year that is not leap.
* The number **p** – number of **holidays** in the year (which are not Saturday or Sunday).
* The number **h** – number of weekends that Joro spends in his **hometown**.

The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

* The output data must be printed on the console.
* On the only output line you must print an integer representing how many times Joro plays football for a year.

### Constraints

* The numbers **p** is in range [0...300] and **h** is in range [0…52].
* Allowed working time for your program: 0.25 seconds.
* Allowed memory: 16 MB.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| T  1  2 | 38 | 52 weekends total in the year, split into:   * 2 hometown weekends 🡪 2 Sundays 🡪 **2** plays * 50 normal weekends 🡪 50 \* 2 / 3 🡪 **33.33** plays   1 holiday 🡪 **0.5** plays  Leap years 🡪 additional **3** plays  Total plays = **38.83** plays 🡪 **38** (rounded) |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| F  3  2 | 36 | t  2  3 | 39 | f  10  5 | 41 | t  0  1 | 38 |

## Problem 2 – Half Sum

Nakov likes numbers. He often plays with their sums and differences. Once he invented the following game. He takes a sequence of numbers, splits them into two subsequences with the same number of elements and sums the left and right sub-sums, and compares the sub-sums. Please help him.

You are given a number **n** and **2\*n** numbers. Write a program to check whether **the sum of the first n numbers is equal to the sum of the second n numbers**. Print as result “Yes” or “No”. In case of **yes**, print also the sum. In case of **no**, print also the difference between the left and the right sums.

### Input

The input data should be read from the console.

* The first line holds an integer **n** – the count of numbers.
* Each of the next **2\*n** lines holds exactly one number.

The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

* The output must be printed on the console.
* Print “**Yes, sum=*S***” where ***S*** is the sum of the first **n** numbers in case of the sum of the first **n** numbers is equal to the sum of the second **n** numbers.
* Otherwise print “**No, diff=*D***” where ***D*** is the difference between the sum of the first **n** numbers and the sum of the second **n** numbers. ***D*** should always be a **positive number**.

### Constraints

* The number **n** is integer in range [0...500].
* All other numbers are integers in range [-500 000 ... 500 000].
* Allowed working time for your program: 0.25 seconds.
* Allowed memory: 16 MB.

### Examples

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| **4**  3  4  -1  -1  2  1  1  1 | Yes, sum=5 | **3**  1  2  3  1  2  2 | No, diff=1 | **2**  1  1  0  0 | No, diff=2 |

## Problem 3 – Sunglasses

Do you know that the next solar eclipse will occur on April 29, 2014? It will be observable from South Asia, Australia, the Pacific and the Indian Oceans and Antarctica. Spiro is an entrepreneur who wants to cash in on this natural phenomenon. Help him produce protective **sunglasses of different sizes**. You will get 5% of the profit.

### Input

* The input data should be read from the console.
* You have an integer number **N** (always an **odd number**) specifying the height of sunglasses.

The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

The output should be printed on the console.

You should print the sunglasses on the console. The sunglasses consist of three parts: **frames**, **lenses** and a **bridge** (the connection between the two frames). Each frame's width is double the height N and should be printed using the character '**\***' (asterisk). Print the lenses with the character '**/**'. Finally, connect the two frames with a bridge that is of size N, using the character '**|**'. Leave the rest of the space between the frames blank.

### Constraints

* The number **N** will be a positive **odd integer** number in range [3…101].
* Allowed working time for your program: 0.25 seconds.
* Allowed memory: 16 MB.

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 3 | \*\*\*\*\*\* \*\*\*\*\*\*  \*////\*|||\*////\*  \*\*\*\*\*\* \*\*\*\*\*\* | 5 | \*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*  \*////////\* \*////////\*  \*////////\*|||||\*////////\*  \*////////\* \*////////\*  \*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\* |

## Problem 4 – Nine-Digit Magic Numbers

Petya often plays with numbers. Her recent game was to play with 9-digit numbers and calculate their sums of digits, as well as to split them into triples with special properties. Help her to calculate very special numbers called “***nine-digit magic numbers***”.

You are given two numbers: **diff** and **sum**. Using the **digits from 1 to 7** generate all 9-digit numbers in format **abcdefghi**, such that their sub-numbers **abc**, **def** and **ghi** have a difference **diff** (ghi-def = def-abc = diff), their sum of digits is **sum** and **abc ≤ def** **≤** **ghi**. Numbers holding these properties are also called “***nine-digit magic numbers***”.

Your task is to write a program to print these numbers in increasing order.

### Input

* The input data should be read from the console.
* The number **sum** stays at the first line.
* The number **diff** stays at the second line.

The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

The output should be printed on the console. Print all nine-digit magic numbers matching given difference **diff** and given sum of digits **sum**, in increasing order, each at a separate line. In case no nine-digit magic numbers exits, print “**No**”.

### Constraints

* The number **sum** will be a positive **integer** number in the range [0…100].
* The number **diff** will be a positive **integer** number in the range [0…1000].
* Allowed working time for your program: 0.25 seconds.
* Allowed memory: 16 MB.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 27  46 | 125171217  131177223  221267313 | 1+2+5+1+7+1+2+1+7 = 27; 171-125 = 46; 217-171 = 46  1+3+1+1+7+7+2+2+3 = 27; 177-131 = 46; 223-177 = 46  2+2+1+2+6+7+3+1+3 = 27; 267-221 = 46; 313-267 = 46 |

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 24  103 | 121224327  211314417 | 1+2+1+2+2+4+3+2+7 = 24; 224-121 = 103; 327-224 = 103  2+1+1+3+1+4+4+1+7 = 24; 314-211 = 103; 417-314 = 103 |

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 12  15 | No | No nine-digit magic numbers with sum=12 and diff=15 |

## Problem 5 – Bits Inverter

Once Teodor played the following game: he started from a sequence of white and black balls and flipped the color of the 1st ball, then the color of the 4th ball, then the color of the 7th ball, etc. until the last ball. Flipping was performed by replacing a black b all with a white ball and vice versa. Teodor was a smart mathematician so he wanted to generalize his game in a formal way. This is what he invented.

You are given a **sequence of bytes**. Consider each byte as sequences of exactly 8 bits. You are given also a number **step**. Write a program to invert the bits at positions: **1**, **1 + step**, **1 + 2\*step**, ... Print the output as a sequence of bytes.

Bits in each byte are counted from the leftmost to the rightmost. Bits are numbered starting from 1.

### Input

* The input data should be read from the console.
* The number **n** stays at the first line.
* The number **step** stays at the second line.
* At each of the next **n** lines **n** bytes are given, each at a separate line.

The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

The output should be printed on the console. Print exactly **n** bytes, each at a separate line and in range [0..255], obtained by applying the bit inversions over the input sequence.

### Constraints

* The number **n** will be an **integer** number in the range [1…100].
* The number **step** will be an **integer** number in the range [1…20].
* The **n numbers** will be integers in the range [0…255].
* Allowed working time for your program: 0.25 seconds.
* Allowed memory: 16 MB.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 2  11  45  87 | 173  71 | We have the following sequence of 16 bits (2 bytes):  **0**0101101 010**1**0111  We invert the bits 1 and 12 (step=11). We get:  **1**0101101 010**0**0111 |

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 3  5  45  87  254 | 169  118  246 | We have the following sequence of 24 bits (3 bytes):  **0**0101**1**01 01**0**1011**1** 1111**1**110  We invert the bits 1, 6, 11, 16 and 21 (step=5). We get:  **1**0101**0**01 01**1**1011**0** 1111**0**110 |