0040 Data Type

1. Look at the ranges of the numerical types in C# described in this chapter.

2. Consider the number of digits after the decimal point. Refer to the table that describes the sizes of the types float, double and decimal.

3. Two floating-point variables are considered equal if their difference is less than some predefined precision (e.g. 0.000001): bool equal = Math.Abs(a - b) < 0.000001;

4. Look at the section about Integer Literals. To easily convert numbers to a different numeral system use the built-in Windows calculator. For a hexadecimal representation of the literal use prefix 0x.

5. Look at the section about Character Literals.

6. Look at the section about Boolean Literals.

7. Look at the sections about Strings and Object Data Type.

8. Look at the sections about Strings and Object Data Type. To cast from object to string use typecasting: string str = (string)obj;

9. Look at the section about Character Literals. It is necessary to use the escaping character \" or verbatim strings.

10. Use Console.WriteLine(…), the character 'o' and spaces.

11. Use Console.WriteLine(…), the character © and spaces. Use Windows Character Map in order to find the Unicode code of the sign "©". Note that the console may display "c" instead of "©" if it does not

Chapter 2. Primitive Types and Variables 137

support Unicode. If this happens, you might be unable to do anything to fix it. Some versions of Windows just do not support Unicode in the console even when you explicitly set the character encoding to UTF-8: Console.OutputEncoding = System.Text.Encoding.UTF8;

You may need to change the font of your console to some font that supports the “©” symbol, e.g. “Consolas” or “Lucida Console”.

12. For the names use type string, for the gender use type char (only one char m/f), and for the unique number and age use some integer type.

13. Use third temporary variable for exchanging the variables: int a = 5; int b = 10;

int oldA = a; a = b; b = oldA;

To swap integer variables other solutions exist which do not use a third variable. For example, if we have two integer variables a and b: int a = 5; int b = 10;

a = a + b; b = a - b; a = a - b;

You might also use the XOR swap algorithm for exchanging integer values: http://en.wikipedia.org/wiki/XOR_swap_algorithm.

0050 Operator and Expression

1. Take the remainder of dividing the number by 2 and check if it is 0 or 1 (respectively the number is odd or even). Use % operator to calculate the remainder of integer division.

2. Use a logical "AND" (&& operator) and the remainder operation % in division. You can also solve the problem by only one test: the division of 35 (think why).

3. Divide the number by 100 and save it in a new variable, which then divide by 10 and take the remainder. The remainder of the division by 10 is the third digit of the original number. Check if it is equal to 7.

4. Use bitwise "AND" on the current number and the number that has 1 only in the third bit (i.e. number 8, if bits start counting from 0). If the returned result is different from 0 the third bit is 1: int num = 25; bool bit3 = (num & 8) != 0;

5. The formula for trapezoid surface is: S = (a + b) * h / 2.

6. Search the Internet for how to read integers from the console and use the formula for rectangle area calculation. If you have difficulties see instructions on the next problem.

7. Use the following code to read the number from the console:

162 Fundamentals of Computer Programming with C#

Console.Write("Enter number: "); int number = Convert.ToInt32(Console.ReadLine());

Then multiply by 0.17 and print it.

8. Use the Pythagorean Theorem a2 + b2 = c2. The point is inside the circle when (x*x) + (y*y) ≤ 5*5.

9. Use the code from the previous task and add a check for the rectangle. A point is inside a rectangle with walls parallel to the axes, when in the same time it is right of the left wall, left of the right wall, down from the top wall and above the bottom wall.

10. To get the individual digits of the number you can divide by 10 and take the remainder of the division by 10: int a = num % 10; int b = (num / 10) % 10; int c = (num / 100) % 10; int d = (num / 1000) % 10;

11. Use bitwise operations: int n = 35; // 00100011 int p = 6; int i = 1; // 00000001 int mask = i << p; // Move the 1-st bit left by p positions // If i & mask are positive then the p-th bit of n is 1 Console.WriteLine((n & mask) != 0 ? 1 : 0);

12. The task is similar to the previous one.

13. Use bitwise operations by analogy with the previous two problems. You can reset the bit at position p in the number n as follows: n = n & (~(1 << p));

You can set bits in the unit at position p in the number n as follows: n = n | (1 << p);

Think how you can combine the above two hints.

14. Read about loops in the Internet or in the chapter “Loops”. Use a loop and check the number for divisibility by all integers from 1 to the square root of the number. Since n < 100, you can find in advance all prime numbers from 1 to 100 and checks the input over them. The prime

Chapter 3. Operators and Expressions 163

numbers in the range [1…100] are: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89 and 97.

15. Use 3 times a combination of getting and setting a bit at a given position. The first exchange is given below: int bit3 = (num >> 3) & 1; int bit24 = (num >> 24) & 1; num = num & (~(1 << 24)) | (bit3 << 24); num = num & (~(1 << 3)) | (bit24 << 3);

16. Extend the solution of the previous problem to perform a sequence of bit exchanges in a loop. Read about loops in the chapter “Loops

0060 Console

1. Use the methods Console.ReadLine() and Int32.Parse().

2. Use Math.PI constant and the well-known geometric formulas.

3. Format the text with Write(…) or WriteLine(…) similar to the example with the letter that we looked at.

4. Use the format strings explained in the “Composite Formatting” section and the method Console.WriteLine(). Below is a piece of the code: int hexNum = 2013; Console.WriteLine("|0x{0,-8:X}|", hexNum); double fractNum = -1.856; Console.WriteLine("|{0,-10:f2}|", fractNum);

```
int hexNum = 2013;
Console.WriteLine("|0x{0,-8:X}|", hexNum);
double fractNum = -1.856;
Console.WriteLine("|{0,-10:f2}|", fractNum);
```

5. There are two approaches for solving the problem:

First approach: Use mathematical tricks for optimized calculation based on the fact that every fifth number is divisible by 5. Think how to implement this correctly and about the borderline cases.

The second approach is easier but it works slower. With a for-loop each number within the given range can be checked. You should read in Internet or in the chapter "Loops" how to use for-loops.

6. Since the problem requires a solution, which does not use conditional statements, you should use a different approach. Two possible solutions of the problem include the use of functions of class Math. The greater of the two numbers you can find with the function Math.Max(a, b) and the smaller with Math.Min(a, b).

Another solution to the problem includes usage of the function for taking the absolute value of a number Math.Abs(a):

```
int a = 2011;

int b = 1990;

Console.WriteLine("Greater: {0}", (a + b + Math.Abs(a-b)) / 2);

Console.WriteLine("Smaller: {0}", (a + b - Math.Abs(a-b)) / 2);
```

The third solution uses bitwise operations:

int a = 1990;

int b = 2011;

int max = a - ((a - b) & ((a - b) >> 31));

Console.WriteLine(max);

```
int a = 2011;
int b = 1990;
Console.WriteLine("Greater: {0}", (a + b + Math.Abs(a-b)) / 2);
Console.WriteLine("Smaller: {0}", (a + b - Math.Abs(a-b)) / 2);
```

The third solution uses **bitwise operations**:

```
int a = 1990;
int b = 2011;
int max = a - ((a - b) & ((a - b) >> 31));
Console.WriteLine(max);
```

There is another solution which is partially correct because it uses a hidden conditional statement (the ternary ?: operator):

194 Fundamentals of Computer Programming with C#

int a = 1990; int b = 2013; int max = a > b ? a : b; Console.WriteLine(max);

```
int a = 1990;
int b = 2013;
int max = a > b ? a : b;
Console.WriteLine(max);
```

7. You can read the numbers in five different variables and finally sum them and print the obtained sum. Note that the sum of 5 int values may not fit in the int type so you should use long.

Another approach is using loops. When parsing the consecutive numbers use conditional parsing with TryParse(…). When an invalid number is entered, repeat reading of the number. You can do this through while loop with an appropriate exit condition. To avoid repetitive code you can explore the for-loops from the chapter "Loops".

8. You can use the comparison statement "if" (you can read about it on the Internet or from the chapter "Conditional Statements"). To avoid repeating code you can use the looping construct "for" (you could read about it online or in the chapter "Loops").

9. You should use a for-loop (see the chapter "Loops"). Read the numbers one after another and accumulate their sum in a variable, which then display on the console at the end.

10. Use a combination of loops (see the chapter "Loops") and the methods Console.ReadLine(), Console.WriteLine() and Int32.Parse().

11. More about the Fibonacci sequence can be found in Wikipedia at: http://en.wikipedia.org/wiki/Fibonacci_sequence. For the solution of the problem use 2 temporary variables in which store the last 2 calculated values and with a loop calculate the rest (each subsequent number in the sequence is a sum of the last two). Use a for-loop to implement the repeating logic (see the chapter "Loops").

12. Accumulate the sum of the sequence in a variable inside a while-loop (see the chapter "Loops"). At each step compare the old sum with the new sum. If the difference between the two sums Math.Abs(current_sum – old_sum) is less than the required precision (0.001), the calculation should finish because the difference is constantly decreasing and the precision is constantly increasing at each step of the loop. The expected result is 1.307.

0070 Conditional Statements

1. Look at the section about if-statements.

2. A multiple of non-zero numbers has a positive product, if the negative multiples are even number. If the count of the negative numbers is odd, the product is negative. If at least one of the numbers is zero, the product is also zero. Use a counter negativeNumbersCount to keep the number of negative numbers. Check each number whether it is negative and change the counter accordingly. If some of the numbers is 0, print “0” as result (the zero has no sign). Otherwise print “+” or “-” depending on the condition (negativeNumbersCount % 2 == 0).

3. Use nested if-statements, first checking the first two numbers then checking the bigger of them with the third.

4. First find the smallest of the three numbers, and then swap it with the first one. Then check if the second is greater than the third number and if yes, swap them too.

Another approach is to check all possible orders of the numbers with a series of if-else checks: a≤b≤c, a≤c≤b, b≤a≤c, b≤c≤a, c≤a≤b and c≤b≤a.

A more complicated and more general solution of this problem is to put the numbers in an array and use the Array.Sort(…) method. You may read about arrays in the chapter “Arrays”.

5. Just use a switch statement to check for all possible digits.

6. From math it is known, that a quadratic equation may have one or two real roots or no real roots at all. In order to calculate the real roots of a given quadratic equation, we first calculate the discriminant (D) by the

formula: D = b2 - 4ac. If the discriminant is zero, then the quadratic

equation has one double real root and it is calculated by the formula:

$x_{1,2} = \frac{-b}{2a}$. If the value of the discriminant is **positive**, then the equation has **two distinct real roots**, which are calculated by the formula: $x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$. If the discriminant is **negative**, the quadratic

equation has no real roots.

7. Use nested if statements. You could use the loop structure for, which

you could read about in the “Loops” chapter of the book or in Internet.

8. First input a variable, which indicates what type will be the input, i.e.

by entering 0 the type is int, by 1 is double and by 2 is string.

9. Use nested if statements or series of 31 comparisons, in order to check all the sums of the 31 subsets of the given numbers (without the empty one). Note that the problem in general (with N numbers) is complex and using loops will not be enough to solve it.
10. Use switch statement or a sequence of if-else constructs and at the end print at the console the calculated points.
11. Use nested switch statements. Pay special attention to the numbers from 0 to 19 and those that end with 0. There are many special cases! You might benefit from using methods to reuse the code you write, because printing a single digit is part of printing a 2-digit number which is part of printing 3-digit number. You may read about methods in the chapter “Methods”.