Algorithm

1. Printing subsequent numbers

The first task is a basic, but very important one. Let us create an algorithm to print numbers below each other, from 1 to 10!. The solution should be precise, it should contain all the necessary steps.

How can we describe repetitive steps?

Supplement the pseudo-code with arrows following the sequence of execution!

Solution

Let's consider number 1.

↓

┌> Repeat the following operations:

│ ↓

│ Print the number number to the screen. Start a new line.

│ ↓

│ Let's increment the number by one.

│ ↓

└── Repeat again, if the number is ≤ 10, otherwise go on to the next line.

↓

End.

Let us introduce variable names and transform the repetition used in the previous solution to a proper loop. What we get is very close to a C program.

X = 1

↓

┌> While X ≤ 10, repeat:

│ ↓

│ Print X to the screen. Start a new line.

│ ↓

│ X = X+1

│ ↓

└── End of repeatition.

↓

End.

Let us write a program that prints only the even numbers (between 1 and 10)! Do we need to test whether X is even or odd? How to solve this task with testing and how to solve it without this testing step?

Solution

Testing is not required, it is enough to start the loop from 2 and increment the variable by 2 in each iteration.

Write a program to print all numbers from 1 to 100 that are divisible by 3 but are not divisible by 5.

Solution

In this case we have no choice, we need to go one-by-one and test all numbers whether they satisfy the criteria individually. We introduce conditional branches:

X = 1

↓

┌> While X < 100, repeat:

│ ↓

│ If X is divisible by 3, and it is not divisible by 5, then:

│ ↓

│ Print X. Start a new line.

│ ↓

│ X = X+1

│ ↓

└── End of repetition.

↓

End.

2. Adding up two numbers

Everybody has learnt in elementary school how to add two (potentialy big) numbers together. This is probably the first algorithm that is taught to every child. By this algorithm, the complex task is reduced to a more elementary one, to the addition of numbers smaller than 10. Hence, the problem is broken down to elementary steps. We can consider the addition of single-digit numbers elementary enough, since we can perform it instantly without any aid.

Let us describe this algorithm in a formal way! Let us use arrows again to indicate the order of execution of the steps in the program!

Solution:

1. Start with the least significant decimal digit

↓

┌> 2. Add up the digits below each other

│ ↓

│ 3. If there is one, add up the carry digit from the previous iteration as well.

│ ↓

│ 4. Divide it by 10, write down the remainder below the line.

│ ↓

│ 5. If we have a carry (the integer part of the division), keep it in mind.

│ ↓

└─ 6. Move to a more significant position, If we still have digits (local values) left, jump back to step

3. Prime factorization algorithm

Write a program to perform and print the prime factorization of an integer number!

Provide the pseudo-code (as before) and indicate the order of execution by arrows!

Solution:

1. Ask the user for a number subject to prime factorization.

↓

2. The smallest divisor to try is 2.

↓

┌> 3. Check if the number can be divided by the divisor without remainder.

│ ├─> If yes, then print: number | divisor; and perform the division.

│ └─> If not, then try the next divisor.

│ ↓

└─ 4. While we do not reach 1, do again from line 3.

↓

5. Print 1|.

4.Draw the flow-charts for the two solutions of the prime factoring program!