# Example algorithms

written in **pseudocode**

**Imre Varga Ph.D.**

University of Debrecen

Faculty of Informatics

Department of IT systems and networks

*Last update: 29 August 2021*

*This work was supported by the construction EFOP-3.4.3-16-2016-00021.*
*The project was supported by the European Union, co-financed by the European Social Fund.*

# Example algorithms

## written in pseudocode

This is a collection of algorithms for the students of "**Algorithms and basics of programming**" subject. These algorithms serve just as examples helping your studies. Introduction to the basics of algorithms and pseudocodes is available in classes, so if you follow them, you must be able to understand the meaning of the following sample codes. First understand them, then try to rewrite them alone, finally verify your codes. These examples enrich your algorithmic thinking, you can find different ideas and tricks in them. Their difficulty level spans a wide range. Sometimes you can find alternative algorithms as well to illustrate different solutions. Occasionally these codes are not so effective. The codes are grouped into 3 sections: simple algorithms, algorithms using arrays and algorithms using subroutines. If you have any comments, please contact me.

## 1) Simple algorithms

1. This algorithm calculates and print out the absolute value of an input number.

```
input num
if num<0 then
  num = -1*num
endif
output num
```

2. This algorithm tells whether an integer (given as input) is an even or odd number.

```
output "Give the number"
input Num
if num%2==0 then
  output "Even"
else
  output "Odd"
endif
```

3. This algorithm asks two numbers from user and tells the relation of them (greater, smaller, equal).

```
output "Please, give two numbers"
input a
input b
if a==b then
  output "Equal"
else
  if a>b then
    output "First value is greater."
  else
    output "Second value is greater."
  endif
endif
```

4. This algorithm calculates the average of two numbers given by the user.

```
input a, b
output (a+b)/2
```

5. This algorithm calculates the average of two numbers given by the user.

```
input a
input b
if a==b then
  output a
else
  if a>b then
    max = a
    min = b
  else
    max = b
    min = a
  endif
  output min+(max-min)/2
endif
```

6. This algorithm reads 3 numbers as input and prints out the greatest one.

```
output "Please, give three numbers"
input a
input b
input c
if a>b then
  if a>c then
    output a
  else
    output c
  endif
else
  if b>c then
    output b
  else
    output c
  endif
endif
```

7. This algorithm gets three numbers from the user and prints out the greatest value.

```
output "Please, give three numbers:"
input a
input b
input c
if a>=b and a>=c then
  output a
endif
if b>=a and b>=c then
  output b
endif
if c>=a and c>=b then
  output c
endif
```

8. This algorithm calculates the sum of integers between two different integer values given by the user (including the limits).

```
output "Enter two different integers:"
input x
input y
max = x
min = y
if max<min then
  tmp = max
  max = min
  min = tmp
endif
sum = min+max
i = min+1
while i<max do
  sum=sum+i
enddo
output sum
```

9. This algorithm calculates the sum of integers between two different integer values given by the user (including the limits).

```
output "Enter two different integers:"
input x
input y
if x>y then
  z = -1
else
  z = 1
endif
output (x+y)/2*((y-x)*z+1)
```

10. This algorithm prints all the non-negative even numbers in increasing order which are not greater than a value given by the user.

```
input n
e = 0
while e<=n do
  output e
  e = e+2
enddo
```

11. This algorithm prints all the positive even numbers in increasing order which are below a value given by the user.

```
input n
e = 2
while e<n do
  output e
  e = e+2
enddo
```

12. This algorithm prints all the positive even numbers in decreasing order which are below a real number given by the user.

```
input n
```

```
e = 0
while e<n do
  e = e+2
enddo
e = e-2
while e>0 do
  output e
  e = e-2
enddo
```

13. This algorithm prints the square numbers (so the square of positive integers) not greater than the value given by the user.

```
i = 1
input n
while i*i<=n do
  output i*i
  i = i+1
enddo
```

14. This algorithm tells whether the input value is a square of an integer or not.

```
input N
if N<0 then
  output "No"
else
  i = 0
  while i<N and i*i!=N do
    i = i+1
  enddo
  if i*i==N then
    output "Yes"
  else
    output "No"
  endif
endif
```

15. This algorithm tells whether the input value is a 3-digit even number or not.

```
input X
if X>99 then
  if X<1000 then
    if X%2==0 then
      output "Yes"
    else
      output "No"
    endif
  else
    output "No"
  endif
else
  output "No"
endif
```

16. This algorithm tells whether the input value is a 3-digit even number or not.

```
f1 = 0
```

```
f2 = 0
f2 = 0
input X
if X>99 then
  f1 = 1
endif
if X<1000 then
  f2 = 1
endif
if X%2==0 then
  f3 = 1
endif
if f1+f2+f3==3 then
  output "Yes"
else
  output "No"
endif
```

17. This algorithm tells whether the input value is a 3-digit even number or not.

```
input X
if X>99 and X<1000 and X%2==0 then
  output "Yes"
else
  output "No"
endif
```

18. This algorithm decides that the 3 positive numbers given by the user can be the side length of a triangle.

```
input a,b,c
if a+b>c and a+c>b and b+c>a then
  output "It is a triangle."
else
  output "It cannot be a triangle."
endif
```

19. This algorithm decides that the 3 positive numbers given by the user can be the side length of a right-angled triangle.

```
input a,b,c
if a*a+b*b==c*c or a*a+c*c==b*b or b*b+c*c==a*a then
  output "It is a right-angled triangle."
else
  output "It is not a right-angled triangle."
endif
```

20. This algorithm presents the first 100 elements of the Fibonacci numbers (https://en.wikipedia.org/wiki/Fibonacci_number).

```
f1 = 0
f2 = 1
output f1
outout f2
n = 2
while n<=100 do
  f3 = f1+f2
```

```
  output f3
  f1 = f2
  f2 = f3
  n = n+1
enddo
```

21. This algorithm presents the elements of the Fibonacci numbers which are below 100.

```
f1 = 0
f2 = 1
output f1
outout f2
f3 = f1+f2
while f3<100 do
  output f3
  f1 = f2
  f2 = f3
  f3 = f1+f2
enddo
```

22. This algorithm determines the greatest common divisor of the two positive integers given by the user.

```
output "Please, give two positive integer numbers:"
input a,b
d = 2
D = 1
while a>=d and b>=d do
  if a%d==0 and b%d==0 then
    a = a/d
    b = b/d
    D = D*d
  else
    d = d+1
  endif
enddo
output "Their GCD is ", D
```

23. This algorithm determines the greatest common divisor of the two positive integers given by the user.

```
output "Please, give two positive integer numbers:"
input a,b
while b!=0 do
  t = b
  b = a%b
  a = t
enddo
output "Their GCD is ", a
```

24. This algorithm determines the greatest common divisor of the two positive integers given by the user.

```
output "Please, give two positive integer numbers:"
input a,b
while a!=b do
  if a>b then
```

```
    a = a-b
  else
    b = b-a
  endif
enddo
output "Their GCD is ", a
```

25. This algorithm determines the lowest common multiple of the two positive integers given by the user.

```
output "Please, give two positive integer numbers:"
input a,b
d = 2
D = 1
while a>=d or b>=d do
  if a%d==0 or b%d==0 then
    if a%d==0 then
      a = a/d
    endif
    if b%d==0 then
      b = b/d
    endif
    D = D*d
  else
    d = d+1
  endif
enddo
output "Their LCM is ", D
```

26. This algorithm determines the lowest common multiple of the two positive integers given by the user.

```
output "Please, give two positive integer numbers:"
input a,b
d = 2
D = 1
while a>=d and b>=d do
  if a%d==0 and b%d==0 then
    a = a/d
    b = b/d
    D = D*d
  else
    d = d+1
  endif
enddo
output "Their LCM is ", a*b/D
```

27. This algorithm lists the prime numbers below 1000.

```
n = 2
while n<1000 do
  d = 2
  while n%d!=0 do
    d = d+1
  enddo
  if d==n then
    output n
```

```
  endif
  n = n+1
enddo
```

28. This algorithm tells that whether the input number is prime or not.

```
input N
d = 2
while N%d!=0 and d*d<N do
  d = d+1
enddo
if d*d>=N then
  output "Prime"
else
  output "Not prime"
endif
```

29. This algorithm produces the non-periodic part of the Collatz-conjecture sequence (https://en.wikipedia.org/wiki/Collatz_conjecture) started by a value given by the user.

```
output "Initial value"
input c
while c!=1 do
  output c
  if c%2==0 then
    c = c/2
  else
    c = 3*c+1
  elndif
enddo
output c
```

30. This algorithm raises the B value to exponent E. The B is a real number and E is an integer, both a given by the user.

```
output "Base: "
input B
output "Exponent: "
input E
if E<0 then
  S = 1
  E = -1*E
else
  S = 0
endif
i = 0
P = 1
while i<E do
  P = P*B
  i = i+1
enddo
if S==1 then
  P = 1/P
endif
output "Power: ", P
```

31. This algorithm "pays" an amount by minimum number of coins. (There are 1, 2, 5 and 10 coins.)

```
output "Enter the amount"
input money
coin10 = (money-(money%10))/10
money = money-coin10*10
coin5 = (money-(money%5))/5
money = money-coin5*5
coin2 = (money-(money%2))/2
coin1 = money-coin2*2
output coin10, coin5, coin2, coin1
```

32. This algorithm calculates the reminder of division not using the modulo (%) operator. (Operands are positive integers.)

```
output "Dividend"
input dd
output "Divisor"
input dr
while dd>dr do
  dd = dd-dr
enddo
output "Reminder"
input dd
```

33. This algorithm approximates the square root of a number fulfilling the required accuracy (based on Newton-Raphson method).

```
output "Give the number:"
input Num
output "Give the accuracy:"
input Acc
Old = Num
Diff = Num
while Diff>Acc do
  New = Old-(Old*Old-Num)/(2*Old)
  Diff = Old-New
  Old = New
enddo
output "Square root:", Old
```

34. This algorithm approximates the value of the mathematical Pi constant. (Greater R leads to more precise value.)

```
input R
i = 0
x = 0
while x<=R do
  y = 0
  while y<=R do
    if x*x+y*y<=R*R then
      i = i+1
    endif
    y = y+1
  enddo
  x = x+1
enddo
output 4*i/(R*R)
```

35. This algorithm reverses the order of digits of a positive integer value.

```
input N
R = 0
while N>0 do
  R = R*10+N%10
  N = (N-(N%10))/10
enddo
output R
```

36. This algorithm converts a positive decimal integer to binary.

```
input N
R = 0
P = 1
while N!=0 do
  R = R+(N%2)*P
  P = P*10
  N = (N-(N%2))/2
enddo
output R
```

37. This algorithm calculates the factorial of an integer.

```
input N
if N<1 then
  output "Not defined."
else
  F = N
  while N>1 do
    F = F*N
    N = N-1
  enddo
  output F
enddo
```

38. This algorithm makes the prime factorization of the input positive integer.

```
d = 2
input N
while N>1 do
  if N%d==0 then
    output d
    N = N/d
  else
    d = d+1
  endif
enddo
```

39. This algorithm determines the number of digits in the (positive or negative) integer input.

```
nd = 1
input x
while x*x>=100 do
  x = x/10
  nd = nd+1
enddo
output nd
```

40. This algorithm gives the solution of the y=a*x+b first degree equation.

```
output "Enter the coefficients"
input a,b
if a==0 then
  output "No solutions"
else
  output -b/a
endif
```

41. This algorithm tells that the year given as input is a leap year or not (using Gregorian calendar).

```
input year
if year%4>0 then
  output "Not leap year"
else
  if year%400==0 then
    output "Leap year"
  else
    if year%100==0 then
      output "Not leap year"
    else
      output "Leap year"
    endif
  endif
endif
```

42. This algorithm tells that the year given as input is a leap year or not (using Gregorian calendar).

```
input year
if (year%4==0 and year%100!=0) or year%400==0 then
  output "Leap year"
else
  output "Not leap year"
endif
```

43. This algorithm prints out the three input values in decreasing order.

```
output "Give three numbers"
input a,b,c
if a>=b and a>=c then
  output a
  if b>c then
    output b, c
  else
    output c, b
  endif
endif
if b>=a and b>=c then
  output b
  if a>c then
    output a, c
  else
    output c, a
  endif
endif
if c>=a and c>=b then
```

```
  output c
  if a>b then
    output a, b
  else
    output b, a
  endif
endif
```

44. This algorithm simplifies a fraction specified by a numerator and a dominator.

```
output "Numerator"
input n
output "Dominator"
input d
if n<d then
  m = n
else
  m = d
endif
t = 2
while t<=m do
  if n%t==0 and d%t==0 then
    n = n/t
    d = d/t
  else
    t = t+1
  endif
enddo
output n, "/", d
```

45. This algorithm determines the sum of two fractions. (The inputs and the output are given by a numerator and a dominator.)

```
output "First numerator"
input n1
output "First dominator"
input d1
output "Second numerator"
input n2
output "Second dominator"
input d2
output "Sum numerator: ", (n1*d2+n2*d1)
output "Sum dominator: ", (d1*d2)
```

46. This algorithm converts a positive decimal fraction (real number) to an equivalent fraction (division of integers).

```
input real
m = 1
i = 0
while i==real*m do
  while i<real*m do
    i = i+1
  enddo
  if i>real then
    m = m*10
    i = 0
```

```
  endif
enddo
output "numerator:", real*m
output "dominator:", m
```

47. This algorithm writes 1000 decimal digits in a cyclical way. (That is 0123456789012345678901234567890123456789012...)

```
x = 0
while x<1000 do
  output x%10
  x = x+1
enddo
```

48. This algorithm writes 1000 decimal digits in a cyclical way. (That is 0123456789012345678901234567890123456789012...)

```
a = 1
while a<=1000/10 do
  b = 0
  while b<10 do
    output b
    b = b+1
  enddo
  a = a+1
enddo
```

49. This algorithm prints out integer pairs row by row, where each number is a decimal digit, and each pair is printed once (that is 2,8 is the same as 8,2).

```
x = 0
while x<10 do
  y = 0
  while y<10 do
    if x<=y then
      output x,y,NEWLINE
    endif
    y = y+1
  enddo
  x = x+1
enddo
```

50. This algorithm finds the second most significant digit of a decimal number greater than 9.

```
input n
m = 1
while m<=n do
  m = m*10
enddo
m = m/100
output ((m-n%m)/m)%10
```

51. This algorithm finds the greatest digit of a positive integer number.

```
input Num
Max = 0
while Num>0 do
  if Num%10>Max then
```

```
    Max = Num%10
  endif
  Num = (Num-Num%10)/10
enddo
output Max
```

52. This algorithm counts the number of '5' digits in a positive input number.

```
count = 0
ouput "Give a positive integer."
input x
while x>0 do
  if x%10==5 then
    count = count+1
  endif
  x = (x-(x%10))/10
enddo
output count
```

53. This algorithm read two values from the user and swaps them.

```
input A,B
C = A
A = B
B = C
output A,B
```

54. This algorithm read two integers from the user and swaps them.

```
input A,B
A = A+B
B = A-B
A = A-B
output A,B
```

## 2) Algorithms using arrays

55. This algorithm reads 10 numbers from the user and stores them in an array.

```
i = 0
while i<10 do
  output A[i]
  i = i+1
enddo
```

56. This algorithm reads numbers from the user. The number of values is not fix; 0 value indicates the end of input (so 0 is not a value to be stored).

```
i = 0
x = 1
while x!=0 do
  output x
  if x!=0 then
    A[i] = x
    i = i+1
  endif
enddo
```

57. This algorithm asks a number from the user then it reads this number of values as input and finally it writes back al the numbers in reverse order.

```
input n
i = 0
while i<n do
  output A[i]
  i = i+1
enddo
i = i-1
while i>=0 do
  output A[i]
  i = i-1
enddo
```

58. This algorithm reads 100 integers as input and stores the even and odd numbers in separate arrays finally it tells the number of even/odd values.

```
i_e = 0
i_o = 0
while i_e+i_o<100 do
  input num
  if num%2==0 then
    Evens[i_e] = x
    i_e = i_e+1
  else
    Odds[i_o] = x
    i_o = i_o+1
  endif
enddo
output "Number of even numbers: ", i_e
output "Number of odd numbers: ", i_o
```

59. This algorithm saves the smallest 50 primes into an array.

```
index = 0
num = 2
while index<=49 do
  div = 2
  while num%div!=0 do
    div = div+1
  enddo
  if div==num then
    Primes[index] = num
    index = index+1
    num = num+1
  endif
enddo
```

60. This algorithm writes the squares of positive integers below 1000.

```
i = 1
while i*i<1000 do
  S[i-1] = i*i
  i = i+1
enddo
```

61. This algorithm stores the fisrt 20 Fibonacci numbers in an array.

```
F[0] = 0
F[1] = 1
i = 2
while i<20 do
  F[i] = F[i-1]+F[i-2]
  i = i+1
enddo
```

62. This algorithm fills an array with hundred 0s and 1s alternately.

```
i = 0
while i<100 do
  b[i] = i%2
  i = i+1
enddo
```

63. This algorithm fills an array with hundred 0s and 4s alternately.

```
i = 0
f = 0
while i<100 do
  b[i] = f
  f = 4-f
  i = i+1
enddo
```

64. This algorithm fills an array with positive integers which are dividable by 15. The values must be in decreasing order, and they cannot be greater the maximum value given by the user.

```
output "Give the maximum: "
input t
while t%15!=0 do
  t = t-1
enddo
j = 0
while t>0 do
  T[j] = t
  t = t-15
  j = j+1
enddo
```

65. This algorithm stores the two input values into the first elements of the array and then the further 18 array elements are the average of the previous two values. Finally, it prints the last value.

```
output "Enter two numbers."
input nums[0]
input nums[1]
index = 2
while 2+18>index do
  num[index] = (num[index-1]+num[index-2])/2
  index = index+1
enddo
output num[index-1]
```

66. This algorithm tells the number of days within the year on a given date. The date is specified by the user as year, month, and day (using Julian calendar).

```
M[0] = 31
M[1] = 28
M[2] = 31
M[3] = 30
M[4] = 31
M[5] = 30
M[6] = 31
M[7] = 31
M[8] = 30
M[9] = 31
M[10] = 30
output "Give the year:"
input Dy
output "Give the month:"
input Dm
output "Give the day of the month:"
input Dd
m = 1
Days = 0
while m<Dm do
  Days = Days+M[m-1]
  m = m+1
enddo
Days = Days+Dm
If Dy%4==0 and Dm>2 then
  Days = Days+1
endif
output "This is the ", Days, "th day of the year."
```

## 2.1) Algorithms using predefined arrays

*Just for simplicity, in the remaining example codes, we assume that an array (called A) is already initialized by values and their number is denoted by N. So in the further algorithms the A[0],...,A[N-1] array elements are available and their values are not known by the programmer.*

67. This algorithm calculates the average of the array elements.

```
i = 0
s = 0
while i<N do
  s = s+A[i]
  i = i+1
enddo
output s/N
```

68. This algorithm adds 32 to those array elements which are between 65 and 90 (including the limits).

```
i = 0
while i<N do
  if A[i]>=65 and A[i]<=90 then
    A[i] = A[i]+32
  endif
  i = i+1
```

```
enddo
```

69. This algorithm asks a number form the user and tells whether it is already stored in the (unsorted) array or not. (Using linear/sequential search algorithm.)

```
output "Searching for: "
input x
i = 0
while i<N and A[i]!=x do
  i = i+1
enddo
if i<N then
  output "Found"
else
  output "Not found"
endif
```

70. This algorithm asks a number form the user and tells whether it is already stored in the (unsorted) array or not. (Using linear/sequential search algorithm.)

```
output "Searching for: "
input x
i = 0
while i<N do
  if A[i]==x then
    break
  endif
  i = i+1
enddo
if i<N then
  output "Found"
else
  output "Not found"
endif
```

71. This algorithm asks a number form the user and tells whether it is already stored in the array or not. (Using linear/sequential search with sentinel algorithm.)

```
output "Searching for: "
input x
A[N] = x
i = 0
while A[i]!=x do
  i = i+1
enddo
if i==N then
  output "Not found"
else
  output "Found"
endif
```

72. This algorithm asks a number form the user and tells whether it is already stored in the sorted array or not. (Using linear/sequential search algorithm.) The array elements are already sorted in increasing manner.

```
output "Searching for: "
input x
```

```
i = 0
while i<N and x<A[i] do
  i = i+1
enddo
if i==N or x>A[i] then
  output "Not found"
else
  output "Found"
endif
```

73. This algorithm asks a number form the user and tells where it is in the array (if stored). So the index of the first occurance or the "Not found" text is printed.

```
output "Searching for: "
input x
i = 0
while i<N and A[i]!=x do
  i = i+1
enddo
if i<N then
  output i
else
  output "Not found"
endif
```

74. This algorithm asks a number form the user and tells whether it is already stored in the array or not. (Using binary search algorithm.)

```
output "Searching for: "
input x
s = 0
e = N-1
if N%2==0 then
  m = N/2-1
else
  m = (N-1)/2
endif
while A[m]!=x and s<e do
  if A[m]<x then
    s = m+1
  else
    e = m-1
  endif
  if (e-s)%2==1 then
    m = (s+e-1)/2
  else
    m = (s+e)/2
  endif
enddo
if A[m]==x then
  output "Found"
else
  output "Not found"
endif
```

75. This algorithm overwrites the last occurrence of 32 in the array to 0.

```
i = N-1
while i>=0 and A[i]==32 do
  i = i-1
enddo
if i>=0 then
  A[i] = 0
endif
```

76. This algorithm replaces all the occurrence of 32 with 95 prior the first 0 array element. (We can assume that there is at least one 0 value in the array.)

```
i = 0
while A[i]!=0 do
  if A[i]==32 then
    A[i] = 95
  endif
  i = i+1
enddo
```

77. This algorithm finds the greatest value of the array.

```
i = 1
max = A[0]
while i<N do
  if max<A[i] then
    max = A[i]
  endif
  i = i+1
enddo
output max
```

78. This algorithm finds the index of the lowest value of the array.

```
i = 1
min_i = 0
while i<N do
  if A[min_i]>A[i] then
    min_i = i
  endif
  i = i+1
enddo
output min_i
```

79. This algorithm finds the second lowest value of the array.

```
if A[0]<A[1] then
  min1 = A[0]
  min2 = A[1]
else
  min1 = A[1]
  min2 = A[0]
endif
i = 2
while i<N do
  if A[i]<min2 then
    min2 = A[i]
  endif
```

```
  if A[i]<min1 then
    min2 = min1
    min1 = A[i]
  endif
  i = i+1
enddo
output min2
```

80. This algorithm changes the order of array elements to reverse.

```
i = 0
j = N-1
while i<j do
  k = A[i]
  A[i] = A[j]
  A[j] = k
  i = i+1
  j = j-1
enddo
```

81. This algorithm sorts the array to increasing order. (Bubble sort)

```
last = N-1
while last>0 do
  i = 0
  while i<last do
    if A[i]>A[i+1] then
      temp = A[i]
      A[i] = A[i+1]
      A[i+1] = temp
    endif
    i = i+1
  enddo
  last = last-1
enddo
```

82. This algorithm sorts the array to increasing order. (Bubble sort with 'change' flag.)

```
ch = 1
last = N-1
while ch==1 do
  ch = 0
  i = 0
  while i<last do
    if A[i]>A[i+1] then
      temp = A[i]
      A[i] = A[i+1]
      A[i+1] = temp
      ch = 1
    endif
    i = i+1
  enddo
  last = last-1
enddo
```

83. This algorithm sorts the array to increasing order. (Selection sort.)

```
first = 0
while first<N-1 do
  min = first
  i = min+1
  while i<N do
    if A[min]>A[i] then
      min = i
    endif
    i = i+1
  enddo
  tmp = A[min]
  A[min] = A[first]
  A[first] = tmp
  first = first+1
enddo
```

84. This algorithm sorts the array to increasing order. (Insertion sort.)

```
current = 1
while current<N do
  i = current-1
  while i>=0 and A[i]>A[i+1] do
    tmp = A[i]
    A[i] = A[i+1]
    A[i+1] = tmp
    i = i-1
  enddo
  current = current+1
enddo
```

85. This algorithm sorts the array to increasing order. (Insertion sort with binary search.)

```
current = 1
while current<N do
  Cur = A[current]
  min_i = 0
  max_i = current-1
  if (max_i-min_i)%2==1 then
    middle = (min_i+max_i-1)/2
  else
    middle = (min_i+max_i)/2
  endif
  while A[middle]!=Cur and min_i<max_i do
    if A[middle]<Cur then
      min_i = middle+1
    else
      min_i = middle-1
    endif
    if (max_i-min_i)%2==1 then
      middle = (min_i+max_i-1)/2
    else
      middle = (min_i+max_i)/2
    endif
  enddo
  if A[middle]>Cur then
    pos = middle
  else
```

```
    pos = middle+1
  endif
  i = current-1
  while i>=pos do
    tmp = A[i]
    A[i] = A[i+1]
    A[i+1] = tmp
    i = i-1
  enddo
  current = current+1
enddo
```

86. This algorithm tells how many times the most frequent digit occurs in a given input integer.

```
i = 0
while i<10 do
  digits[i] = 0
  i = i+1
enddo
input num
while num>0 do
  d = num%10
  num = (num-d)/10
  digits[d] = digits[d]+1
enddo
max=digit[0]
i = 1
while i<10 do
  if digits[i]>max then
    max = digits[d]
  endif
  i = i+1
enddo
output "The most frequent digit occurs ",max," times."
```

87. This algorithm determines the median of the array elements.

```
l = N-1
while l>0 do
  i = 0
  while i<l do
    if A[i]>A[i+1] then
      tmp = A[i]
      A[i] = A[i+1]
      A[i+1] = tmp
    endif
    i = i+1
  enddo
  l = l-1
enddo
if N%2==0 then
  output (A[N/2]+A[N/2-1])/2
else
  output A[(N-1)/2]
endif
```

88. This algorithm rearranges the array where all even values are prior any odd values. (We can assume that the array stores only integers.) The order of even number and the order of odd numbers separately are not important.

```
i1 = 0
i2 = N-1
while i1<i2 do
  if A[i1]%2==0 then
    i1 = i1+1
  else
    if A[i2]%2==1 then
      i2 = i2-1
    else
      t = A[i1]
      A[i1] = A[i2]
      A[i2] = A[i1]
      i1 = i1+1
      i2 = i2-1
    endif
  endif
enddo
```

89. This algorithm tells how many times the difference between neighboring array elements is greater than 10.

```
index = 0
count = 0
while index<N-1 do
  if A[i]+10<A[i+1] or A[i+1]+10<A[i] then
    count = count+1
  endif
  index = index+1
enddo
output count
```

90. This algorithm compares two array and tell whether they contain the same values in the same order. Like the predefined A array, the B array also have available elements with cardinality M.

```
i = 0
if N!=M then
  output "They are different arrays."
else
  while i<M and A[i]==B[i] do
    i = i+1
  enddo
  if i==M then
    output "They have the same content."
  else
    output "They are different arrays."
  endif
endif
```

91. This algorithm prints the elements of two sorted arrays. Like the predefined A array, the B array also have available elements with cardinality M. The arrays and the output both must be sorted in increasing manner.

```
i_a = 0
```

```
i_b = 0
while i_a<N and i_b<M do
  if A[i_a]<B[i_b] or i_b==M then
    output A[i_a]
    i_A = i_a+1
  endif
  if A[i_a]>=B[i_b] or i_a==N then
    output B[i_b]
    i_b = i_b+1
  endif
enddo
```

92. This algorithm determines the moving average of a dataset over 5-element subsets. So, each element of the new array is the mean of the 5 consecutive elements of the original array. The new array has N-4 elements because the first 2 and the last 2 values have not enough neighbors.

```
i = 2
while i<N-3 do
  j = -2
  sum = 0
  while j<3 do
    sum = sum+A[i+j]
    j = j+1
  enddo
  MA[i-2] = sum/5
  i = i+1
enddo
```

## 3) Algorithms using subroutines

93. This algorithm implements the mathematical sign function and demonstrates its application.

```
function sign(N)
  if N==0 then
    return 0
  endif
  if N>0 then
    return +1
  endif
  if N<0 then
    return -1
  endif
endfunction

output “Give me a number”
input A
output “Give me another number”
input B
if sign(A)==sign(B) then
  output “They have the same sign.”
else
  output “They have different signs.”
endif
```

94. This algorithm calculates the third angle (in degrees) of a triangle if two is given as parameters. Special case is also managed the in the example.

```
function triangle(alpha,beta)
  if alpha+beta<180 and alpha>0 and beta>0 then
    return 180-alpha-beta
  else
    return -1
  endif
endfunction

output "Give two angles"
input A,B
if triangle(A,B)==-1 the
  output "Wrong values. Not triangle angles.
else
  output "Angles: ", A, B, triangle(A,B)
endif
```

95. This algorithm uses a function for exponentiation, where the exponent is a positive integer. It calculates some values.

```
function power(b,e)
  p = 1
  while e>0 do
    p = p*b
    e = e-1
  enddo
  return p
endfunction

output "Six squared is", power(6,2)
output "Five cubed is", power(5,3)
output "Four squared cubed is", power(power(4,2),3)
```

96. This algorithm defines a function to tell the smallest prime number above its parameter. It prints the first 5 prime numbers.

```
function NextPrime(current)
  next = current+1
  while 1==1 do
    divisor = 2
    while next%divisor!=0 do
      divisor = divisor+1
    enddo
    if next==divisor then
      return next
    endif
    next = next+1
  enddo
endfunction

p1 = NextPrime(1)
p2 = NextPrime(p1)
p3 = NextPrime(p2)
p4 = NextPrime(p3)
p5 = NextPrime(p4)
```

```
output p1, p2, p3, p4, p5
```

97. This algorithm applies a function to tell the length of the given month in days (in case of non-leap years). It tells whether June or July is the longer and then it lists the days of November.

```
function months(M)
  if M==2 then
    return 28
  else
    if M==4 or M==6 or M==9 or M==11 then
      return 30
    else
      return 31
    endif
  endif
endfunction

if months(6)<months(7) then
  output "July is longer than June."
i = 1
while i<=months(11) do
  output i, " November 2021",NEWLINE
  i = i+1
enddo
```

98. This algorithm defines two functions. They give back the perimeter and the area of a circle based on the radius.

```
function perimeter(R)
  return 2*R*3.141592
endfunction
function area(R)
  return R*R*3.141592
endfunction

output "The value of Pi is: ", area(1.0)
output "The value of Pi is: ", perimeter(2/4)
```

99. This algorithm defines a procedure to print minimum 1 and maximum 5 stars (asterisk symbols) according to its parameter.

```
procedure stars(x)
  n = 1
  while x>0 and n<=x and n<=5 do
    output "*"
    n = n+1
  enddo
endprocedure
call stars(5)
```

100. This algorithm contains a function definition to calculate the gross price based on the net price and the VAT (in percentages).

```
function gross(net,VAT)
  toPay = net*(1+VAT/100)
  return toPay
endfunction
```

```
output "Net price of a product "
input x
output "Relevant VAT (%) "
input y
output "The gross price is ", gross(x,y)
```

101. This algorithm tells you how much programming effort you need to be a quite good programmer using a procedure. 😊

```
procedure WhatToDo(N)
  if N<1000 then
    output "You must practice a bit more."
  else
    output "You must practice a LOT.
  endif
endprocedure

output "How many codes have you already written alone?"
input NumOfCodes
call WhatToDo(NumOfCodes)
output "Don't hesitate. Start it."
```

102. This algorithm defines a function to calculate the factorial of a positive integer. It is used in a combinatorial problem.

```
function fact(N)
  f = 1
  while N>1 do
    f = f*N
    N = N-1
  enddo
  return f
endfunction

output "The chance to win on 5/90 lottery "
output fact(5)*fact(90-5)/fact(90)
```

103. This algorithm defines a recursive function to calculate the factorial of a positive integer.

```
function fact(N)
  if N==1 then
    return 1
  else
    return N*fact(N-1)
  endif
endfunction
if fact(5)==1*2*3*4*5 then
  output "It works!"
endif
```

104. This algorithm uses a function to find the middle value from the three parameters.

```
function mid(x,y,z)
  if (x>=y and x<=z) or (x>=z and x<=y) then
    m = x
  endif
```

```
  if (y>=x and y<=z) or (y>=z and y<=x) then
    m = y
  endif
  if (z>=y and z<=x) or (z>=x and z<=y) then
    m = z
  endif
  return m
endfunction

output "Enter three numbers"
input a,b,c
output mid(a,b,c) " is not the greatest and not the lowest."
```

105. This algorithm contains a function definition. It gets a time moment defined by the 3 parameters and returns the number of seconds before the given moment of the day. In the example it returns by 86400.

```
function seconds(hour, min, sec)
  return (hour*60+min)*60+sec
endfunction

output "Number of seconds in a day: ", seconds(24,0,0)
```

106. This algorithm has a paramaterless procedure to print integers from 1 to 100.

```
procedure integers()
  n = 1
  while n<=100 do
    output n, NEWLINE
    n = n+1
  enddo
endprocedure

output "Here are hundred integers: "
call integers()
```

107. This algorithm contains a function definition, which can tell that its parameter is a positive even integer below 10.

```
function IsSmallEven(N)
  if N%2==0 and N<10 and N>0 then
    return 1
  else
    return 0
  endif
endfunction

output "Give a number"
input n
if IsSmallEven(n)==0 then
  output "yes"
else
  output "no"
endif
```

108. This algorithm contains 3 function definitions, to tell that a number is a positive even integer below 10.

```
function IsEven(N)
  if N%2==0 then
    return 1
  else
    return 0
  endif
endfunction

function IsPositive(N)
  if N>0 then
    return 1
  else
    return 0
  endif
endfunction

function IsBelow10(N)
  if N<10 then
    return 1
  else
    return 0
  endif
endfunction

output "Give a number"
input n
if IsEven(n)*IsPositive(n)*IsBelow10(n)==0 then
  output "yes"
else
  output "no"
endif
```

109. This algorithm contains a procedure definition. It prints the exact time (hours, minutes, seconds) based on the number of seconds that have elapsed since that day (given as a parameter). In the example it prints 2:46:40.

```
procedure time(t)
  s=t%60
  m={t/60}%60
  h={t/3600}
  output h, ":", m, ":", s
endprocedure

output "The time after 10,000s is: "
call time(10000)
```

110. This algorithm prints out a multiplication table by a procedure. The size of the output is given as a parameter. Ten different sized structure is printed by calls.

```
procedure MultTab(N)
  r=1
  while r<=N do
    c=1
    while c<=N do
      output c*r
      c=c+1
```

```
    enddo
    output NEWLINE
    r=r+1
  enddo
endprocedure

size = 1
while size<=10 do
  call MultTab(size)
  size = size+1
enddo
```

111. This algorithm prints out a chessboard pattern (using 0s and 1s in a square area) by the help of a procedure and a function.

```
function change(x)
  return 1-x
endfunction

procedure chessboard(x)
  color=0
  s=1
  while s<=x do
    o=1
    while o<=x do
      output color
      color=change(color)
      o=o+1
    enddo
    output NEWLINE
    if x%2==0 then
      color=change(color)
    endif
    s=s+1
  enddo
endprocedure

output "Enter the size "
input S
call chessboard(S)
```

112. This algorithm converts a positive decimal integer into another number system (with base number between 2 and 9). It is implemented in a recursive procedure. The example calls provides 20, 10011 and 354.

```
procedure Convert(N,B)
  if N!=0 then
    call Convert((N-N%B)/B,B)
    output N%B
  endif
endprocedure

call CONVERT(16,8)
output NEWLINE
call CONVERT(19,2)
output NEWLINE
```

```
call CONVERT(832,6)
```

113. This algorithm converts Arabic numerals to Roman numerals. It is limited to 1-12 only (for example tower clocks often use Roman numerals). The algorithm uses a procedure to print the Roman numeral and starts a new output line.

```
procedure Roman(Arabic)
  if Arabic==4 or Arabic==9 then
    output "I"
    Arabic = Arabic+1
  endif
  if Arabic>=10 then
    output "X"
    Arabic = Arabic-10
  endif
  if Arabic>=5 then
    output "V"
    Arabic = Arabic-5
  endif
  while Arabic>0 do
    output "I"
    Arabic = Arabic-1
  enddo
  output NEWLINE
endprocedure

Arabic = 1
while Arabic<=12 do
  call Roman(Arabic)
enddo
```

114. This algorithm defines two function to determine the greatest common divisor and the lowest common multiple of their two parameters.

```
function GCD(N1,N2)
  while N2!=0 do
    Nt = N2
    N2 = N1%N2
    N1 = Nt
  enddo
  return N1
endfunction

function LCM(n1,n2)
  return n1*n2/GCD(n1,n2)
endfunction

output "Enter 2 positive integers"
input x,y
output "Their lowest common multiple is ", LCM(x,y)
if GCD(x,y)==1 then
  output "They are coprime integers."
endif
```

115. This algorithm uses a procedure to print out integers on 10 positions with right alignment. It uses required number of leading spaces (" "). Each number is written into separate lines so

units, tens, hundreds, etc are under each other. The parameter must be less than ten billion of course.

```
procedure RightAlign(Num)
  Num2 = Num
  nd = 0
  while Num2>0 do
    Num2 = (Num2-Num2%10)/10
    nd = nd+1
  enddo
  while nd<=10 do
    output " "
    nd = nd+1
  enddo
  output Num, NEWLINE
endprocedure

call RightAlign(123)
call RightAlign(4)
call RightAlign(5678901)
call RightAlign(234567890)
call RightAlign(12345)
```

116. This algorithm finds the greatest value among 4 numbers by the help of functions.

```
function big2(a,b)
  if a<b then
    return b
  else
    return a
  endif
endfunction

function big4(a,b,c,d)
  if big2(a,b)<big2(c,d) then
    return big2(c,d)
  else
    return big2(a,b)
  endif
endfunction

output "Enter 4 numbers"
input n1,n2,n3,n4
output "The greatest is ", big4(n1,n2,n3,n4)
```

117. This algorithm finds the greatest value among 4 numbers by the help of functions.

```
function big2(a,b)
  if a<b then
    return b
  else
    return a
  endif
endfunction

function big4(a,b,c,d)
  return big2(big2(a,b),big2(c,d))
```

```
endfunction

output "Enter 4 numbers"
input n1,n2,n3,n4
output "The greatest is ", big4(n1,n2,n3,n4)
```

118. This algorithm tells (in a silly way) how many binary combinations can be stored in one byte (using two functions, one of them is a parameterless function).

```
function square(N)
  return N*N
endfunction

function combinations()
  i = 1
  b = 2
  while i<=3 do
    b = square(b)
    i = i+1
  enddo
  return b
endfunction

if combinations()==256 then
  output "It is always true."
endif
```

119. This algorithm tells (in a silly way) how many binary combinations can be stored in one byte (using two functions, one of them is a parameterless function).

```
function square(N)
  return N*N
endfunction

function combinations()
  return square(square(square(2)))
endfunction

if combinations()==256 then
  output "It is always true."
endif
```

120. This algorithm can tell whether a box (rectangular cuboid) can surround another one. Both boxes are specified by their dimensions. The code contains several functions to solve the problem.

```
function biggest(a,b,c)
  if a>=b and a>=c then
    return a
  endif
  if b>=a and b>=c then
    return b
  endif
  if c>=a and c>=b then
    return c
  endif
endfunction
```

```
function smallest(A,B,C)
  if A<=B and A<=C then
    return A
  endif
  if B<=A and B<=C then
    return B
  endif
  if C<=A and C<=B then
    return C
  endif
endfunction

function middle(l,n,k)
  return l+n+k-biggest(l,n,k)-smallest(l,n,k)
endfunction

function ContainBox(x1,y1,z1,x2,y2,z2)
  fb = 0
  fm = 0
  fs = 0
  if biggest(x1,y1,z1)>biggest(x2,y2,z2) then
    fb = 1
  endif
  if middle(x1,y1,z1)>middle(x2,y2,z2) then
    fm = 1
  endif
  if smallest(x1,y1,z1)>smallest(x2,y2,z2) then
    fs = 1
  endif
  if fb+fm+fs==3 then
    return 1
  else
    return 0
  endif
endfunction

if ContainBox(9,5,7,8,6,1) then
  output "The <9,5,7> box can contain the <8,6,1> box."
else
  output "The first box cannot surround the second one."
endif
```

121. This algorithm contains a procedure which can tell the root(s) of a second-degree equation. The coefficients of the general *ax2+bx+c=0* form are the parameters of the subroutine. It uses a 'built-in' `sqrt()` function to calculate the square root of a number.

```
procedure SDE(a,b,c)
  if a==0 then
    if b==0 then
      output "Not equation."
    else
      x=-c/b
      output x
    endif
  else
```

```
    d=b*b-4*a*c
    if d>0 then
      x1=(-b+sqrt(d))/(2*a)
      x2=(-b-sqrt(d))/(2*a)
      output x1, x2
    else
      if d==0 then
        x=-b/(2*a)
        output x
      else
        output ”No solution.”
      endif
    endif
  endif
endprocedure

output “3*x*x-9*x+4=0”
call SDE(3,-9,4)
output “2*x+5=0”
call SDE(0,2,5)
```

122. This algorithm defines a recursive procedure which needs two positive integer parameters. The procedure prints out the integers between 1 and the value of the first parameter. The second parameter tells the direction of the numbers (either in increasing order or in decreasing order).

```
procedure integers(Max,Dir)
  if Dir==1 and Max>1 then
    call integers(Max-1,Dir)
  endif
  output Max
  if Dir==0 and Max>1 then
    call integers(Max-1,Dir)
  endif
endprocedure

output “Enter a positive integer”
input Number
output “Integers in increasing order: ”
call (Number-1,1)
output “Integers in decreasing order: ”
call (Number,0)
```

123. This algorithm calculates the mean and the standard deviation of the first few elements of the Collatz-sequence. It is implemented in a procedure where the first parameter (a positive integer) is the initial value of the sequence, and the second parameter is the number of consecutive elements we are interested in. The generation of the elements are helped by a function and the generated values are stored in an array. The ‘built-in’ `sqrt()` function can be used.

```
function next_Collatz(Current)
  if Current%2==0 then
    return Current/2
  else
    return 3* Current+1
  endif
endfunction
```

```
procedure Collatz_Stat(Init,Num)
  Coll[0] = Init
  i = 1
  while i<Num do
    Coll[i] = next_Collatz(Coll[i-1])
    i = i+1
  enddo
  Sum = 0
  while i>=1 do
    i = i-1
    Sum = Sum+ Coll[i]
  enddo
  Avg = Sum/Num
  output "The mean is ", Avg, NEWLINE
  Dev_Sum = 0
  while i<Num do
    Dev_Sum = Dev_Sum+(Coll[i]-Avg)*(Coll[i]-Avg)
    i = i+1
  enddo
  Std_Dev = sqrt(Dev_Sum/Num)
  output "The standard deviation is ", Std_Dev, NEWLINE
endprocedure

output "Enter the initial value: "
input Start
output "Enter the required length of the analyzed sequence: "
input Num
call Collatz_Stat(Start,Num)
output "I hope you are satisfied."
```

If you were able to write all the above algorithms properly do not worry about your algorithmic competencies, you just must learn the details of a real programming languages and you will be a good programmer.