Course 2: Algorithm description
- Algorithmic language
- Data specification
- Examples
- Succesive refinement technique and subalgorithms

1 Algorithmic language

The pseudocode we will use further allows specification of simple and structured processing as follows.

Assignment. Assigning obtained value through evaluation of an expression with variable v is specified:

$$v \leftarrow \text{< expression >}$$

Expressions are used to describe calculations and consist of operands and operators. Operands can be a variable or a constant. Operators can be:

- *Arithmetic:* + (sum), - (difference), * (multiplication), / (division), ^ (power), DIV (division), MOD (remainder of division).
- *Relational:* = (equal), ≠ or <> (not equal), < (lower than), <= (lower or equal), > (greater than), >= (greater or equal);
- *Logical:* OR (disjunction), AND (conjunction), NOT (negation).

Reading. For assigning a value taken from the input device to *v* variable is specified:

READ *v*

Writing. For sending to output device the value of a variable is specified:

WRITE < expression >

For specification of any algorithm is enough using following structures: sequential, conditional, repetitive.

Sequential structure: O sequence of n processings is specified by:

< processing 1 >

< processing 2 >

.

.

< processing n >

Conditional structure: Is used to specify in what branch the execution goes, decision being taken by whether a condition is satisfied or not. A conditional structure is specified by:

IF < condition > THEN < process 1 > ELSE < process 2 >

Also there is a structure that executes only if the condition is satisfied:

IF < condition > THEN < process 1 >

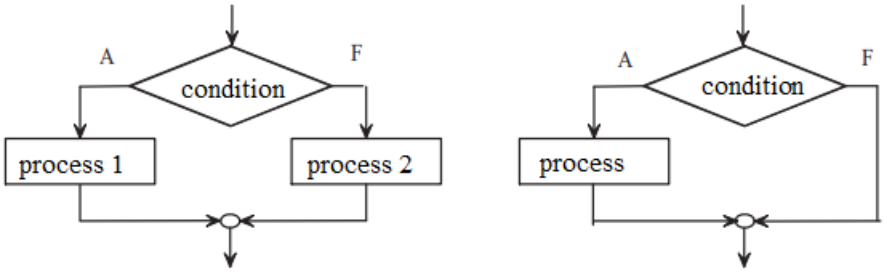

Figure 1. Graphical description of alternative sequences

Repetitive structures: Allow processing description that must be done repeatedly. There exist 2 types of repetitive structure: With pre-condition and with post-condition.

Pre-condition: WHILE < condition > DO < process >

Post-condition: REPEAT < process > UNTIL < condition >

A particular case of repetitive structure is the one with using of a counter, when we know the needed number of repetitions. In that case we need a specific variable for that counter, that will be modified after each repetition. We need an initial value (*v1*) and a final value(*v2*).

FOR $v \leftarrow \overline{v1, v2, step}$ DO < process >

Previous expression is equal to the following(if *step* is positive):

v ←—*v1*
WHILE *v* <= *v2* DO
|| < process >
|| *v* ←—*v* + *step*

2 Data specification

In algorithms we also may need variables also specifying their type. For instance, for specification of simple data can be used following key words: INTEGER (for whole numbers), REAL (for real values), BOOLEAN(for logical values, TRUE or FALSE), CHAR(for single characters).
Declaring variables:

INTEGER a
REAL b
BOOLEAN c
CHAR d

Array data type is specified declaring the nature of elements and the limit of that array. For example, in case of an onedimensional array with whole number we specify: INTEGER t1[n1..n2], and in case of bidimensional array with real numbers we specify: REAL t2[m1..m2,n1..n2]. Elements of an array are specified through their index. For example, the element that is located on *i* position is specified by t1[*i*]. In case of bidimensional array we specifiy the row and the column the element is located, t2[*i,j]*.

3 Examples

Sequential structure.
Find the area of a triangle knowing its sides (a, b and c).
Solving. We ase Herone's formula: area = $\sqrt{\boldsymbol{p}(\boldsymbol{p}-\boldsymbol{a})(\boldsymbol{p}-\boldsymbol{b})(\boldsymbol{p}-\boldsymbol{c})}$, where *p* is semi-perimeter.
Algorithm description.

REAL *a, b, c, p, area* *{ declaring used variables }*

READ *a,b,c* *{ reading input data }*

$p \leftarrow (a + b + c) / 2$ { *semi-perimeter* }

area $\leftarrow \sqrt{\boldsymbol{p(p-a)(p-b)(p-c)}}$ *{ area }*

WRITE *area* *{print the result }*

Conditional structure.

Find the real roots of equation $ax^2+bx+c = 0$, where a,b and c are random real values.

Solving. We use solving method of equation of grade 2 using formulas to find discriminant and root calculation.

Algorithm description.

```
REAL a, b, c, delta, x1, x2    {        declaring used variables        }
READ a, b, c    {       reading input data       }
delta ← b^2 − 4ac    {       finding discriminant    }
{       Analyze discriminant sign       }
IF delta < 0 THEN WRITE   "Equation has no real roots"
ELSE
   ||IF delta = 0 THEN
   ||   ||   ||x1 ← −b/(2a)
   ||   ||   ||WRITE "Common root ", x1
   ||   ||ELSE
   ||   ||   ||x1 ← (−b + √delta)/(2a)
   ||   ||   ||x2 ← (−b − √delta)/(2a)
   ||   ||   ||WRITE x1, x2
```

There are 3 real values of variables *a,b and c.* (a) *Find the least value. (b) Print the three value in ascending order.*

Algorithm description. (a) Comparing values 2 by 2.

```
REAL a, b, c, min
READ a, b, c
IF a < b THEN
        IF a < c THEN min ← a
                 ELSE min ← c
ELSE
        IF b < c THEN min ← b
                 ELSE min ← c
WRITE min
```

Another, more compact variant:

```
REAL a, b, c, min
READ a, b, c
min ← a
IF b < min THEN min ← b
IF c < min THEN min ← c
WRITE min
```

(b) One solution could be checking all 6 existent outcomes:

```
REAL a, b, c, min
READ a, b, c
IF a < b AND b < c THEN WRITE a, b, c
IF a < c AND c < b THEN WRITE a, c, b
IF c < a AND a < b THEN WRITE c, a, b
IF c < b AND b < a THEN WRITE c, b, a
IF b < a AND a < c THEN WRITE b, a, c
IF b < c AND c < a THEN WRITE b, c, a
```

Another variant could be succesive compartion of 2 values

```
REAL a, b, c, min
READ a, b, c
IF a < b THEN
      IF b < c THEN WRITE a, b, c
            ELSE IF a < c WRITE a, c, b
                  ELSE WRITE c, a, b
ELSE
      IF a < c THEN WRITE b, a, c
            IF b < c THEN WRITE b, c, a
            ELSE WRITE c, b, a
```

Repetitive structure. When describing a repetitive structure must be specified: (i) initial values of variables that interfere in repetitive process; (ii) processings that are repeated; (iii) stop condition.

Solve the sums: (a) $\sum_{i=1}^{n} i^3$, n ∈ N*; (b) $\sum_{i=1}^{n} x^i / i!$ for x ∈ (0,1);

Algorithm description. (a) Process that is repeating here is adding the terms. Repetitive processing is finalized when all terms were added. Following 3 descriptions are equivalent:

```
INTEGER S, i, n
READ n
S ← 0
i ← 1
WHILE i ≤ n
    ‖S ← S + i^3
    ‖i ← i + 1
WRITE S
```

```
INTEGER S, i, n
READ n
S ← 0
FOR i ← 1, n DO S ← S + i^3
WRITE S
```

```
INTEGER S, i, n
READ n
S ← 0
i ← 1
REPEAT
    ‖S ← S + i^3
    ‖i ← i + 1
UNTIL i > n
WRITE S
```

(b) Sum can be rewritten as $\sum_{i=1}^{n} T(i)$ where *T(i)* = x^i / i! . We can see that between successive terms exists a relation that allow solving *T(i)* starting from *T(i-1)*:

$$T(i) = \frac{x^i}{i!} = \frac{x^{i-1}}{(i-1)!} \cdot \frac{x}{i} = T(i-1) \cdot \frac{x}{i}$$

```
INTEGER i, n
REAL x, S
READ x, n
S ← 0
T ← 1
i ← 1
REPEAT
T ← T * x/i
S ← S + T
i ← i + 1
UNTIL i > n
WRITE S
```

Arithmetic mean. Find the arithmetic mean of a sequence of finite *n* real numbers: (x_1, x_2, … x_n)

Solving. Arithmetic mean is obtained by formula: $(\sum_{i=1}^{n} x_i)/n$;

Algorithm description. Sequence can be represented as an array x[1..n] and the process there is a repetitive process for which we know the amount of repetition.

```
REAL x[1..n], medie
INTEGER n,i
READ x[1..n]
medie ← 0
FOR i ← 1,n DO
    ||medie ← medie + x[i]
medie = medie/n
WRITE medie
```

4 Succesive refinement technique and subalgorithms

The easiest method of solving an algorithmic problem is by dividing the problem in little subproblems and solve them. Every subproblem will be solved with an (sub)algorithm, and for solving initial problem's algorithm will be used those subalgorithms to build the final result. If the problem is consisted of more subproblems of the same nature, then they can be solved using the same subalgorithm. For that purpose processings applied in a subalgorithm will be used upon some *generic data* that will be replaced with specified data in the time of execution. The transfer of the control of execution from an algorithm to a subalgorithm is called *call*, *generic data* are called *formal parameters,* and their specified values are called *effective parameters.* The effect of an subalgorithm is to return a value to the algorithm.

Structure of an subalgorithm:

```
< subalgorithm name > (< generic data >)
    ||< helping data >
    ||< specific processings >
    ||RETURN < result >
```

Example 1: Determine the largest of the minimums values from each line of a matrix $(a_{ij})_{i=\overline{1,m},j=\overline{1,n}}$ $(\max_{i=\overline{1,m}} \min_{j=\overline{1,n}} a_{ij})$

Solving. We declare an array that contains minimal values of each line: $b_i = min_{j=\overline{1,n}}\ a_{i,j},\ i = \overline{1,m}$ after which we determine maximal value from that array. Problem is supposed to have 2

subproblems: determine minimal value from a finite sequence with *n* real elements, and determine maximal value from a finite sequence with *m* real elements.

Subalgorithms for determining minimal and maximal values:

```
minim (REAL x[1..n])
REAL min
INTEGER i
min ← x[1]
FOR i ← 2, n DO
    ||IF min > x[i] THEN min ← x[i]
RETURN min
```

```
maxim (REAL y[1..m])
REAL max
INTEGER i
max ← y[1]
FOR i ← 2, n DO
    ||IF max < y[i] THEN max ← x[i]
RETURN max
```

Problem's algorithm consist in building an array with minimal values and determining the maximal value of that array:

```
REAL a[1..m, 1..n]
REAL b[1..m]
REAL c
INTEGER m, n, i
READ a[1..m, 1..n]
FOR i ← 2, n DO
    ||b[i] ← minim(a[i, 1..n])
c ← maxim(b[1..m])
WRITE c
```

5. Exercises

1. Solve:

(a) $\prod_{i=1}^{n}(i+1)$; (b) $\sum_{i=1}^{n} 1/(i!)^2$.

2. Approximate, with precision ε :

(a) $\sum_{i=1}^{\infty} x^i/i!$; (b) $\sum_{i=0}^{\infty}(-1)^i x^{2i}/(2i)!$; (c) $\sum_{i=0}^{\infty}(-1)^i x^{2i+1}/(2i+1)!$; (d) $\sum_{i=1}^{\infty}(-1)^{i+1}x^i/i$

3. (a) Determine the sum of all digits of a natural number. (b) Determine all distinct digits from a natural number. (c) Determine the value obtained from inversing the digits of a natural number.