Engineering program development

Edited by Péter Vass

Introduction

Question:

Why engineering program development may be useful for a PhD student in Earth Sciences?

Counter-argument:

In these days a wide range of different types of software is available for us to help our work. Why should we waste time on programming?

Disproof:

There are not cheap (or free) and universal types of software by which even all sorts of scientific and engineering problems can be solved.

Sooner or later each student, researcher or technician encounters a problem which cannot be solved by the available types of software.

Introduction

In such a case, we can select from the following alternatives:

1. searching for a program via the Web
   - the more special the problem is the less probable we will find a suitable one,
2. asking a person to write a program
   - the more special the problem is the more expensive will be the service,
3. creating our own computer program to solve the problem emerged
   - it takes logical thought, professional knowledge regarding the problem and skills in computer programming.

Some basic concepts and definitions

(Computer) program

- is a collection of suitably ordered and coded instructions,
- is written (coded) in some programming language,
- is executable by a computer to perform a specific task.

Software

is a collection of computer programs, subroutine (or function) libraries, related data and pieces of documentation.

A software provides a wider range of services than a single program.

Some basic concepts and definitions

The logical content of a program is based on one or more algorithms.

Algorithm

- is a logical part of a computer program which performs a well-defined task,
- is a self-contained step-by-step set of operations.

Each algorithm has its own *input datum* or *data*.
It processes the input by the execution of its operations and provides the *output datum* or *data* as a result.

Some basic concepts and definitions

Connection between an algorithm and its environment:

Of course, a given task can be solved by using different algorithms (e.g. arranging numbers in increasing order).

<u>The essential requirements of algorithms:</u>

- it must be universal (it must provide the right output data independently of the values of input data),
- it must finish after a finite number of steps,
- it must be a sequence of well-defined operations.

Some basic concepts and definitions

The efficiency of an algorithm is generally analysed by two points of view:

- the *amount of memory*
- and the *time* required for the algorithm to solve its task in the form of a computer program.

In most cases, either of the two indicators above can only be improved at the expense of the other one.

But a computer cannot directly execute an algorithm, because it "*does not understand*" the different types of complex symbolism used for expressing algorithms.
Therefore, we must code the algorithm by using a programming language.
The coded form of an algorithm is already a program.
The steps of a program are called *instructions*.

Some basic concepts and definitions

We must pay particular attention to the correctness of each algorithm because a computer is not able to decide whether a program is correct or not.

Only in the case of correct algorithms, we have a chance to create a correct program.

If a program is based on a wrong algorithm, it will provide a bad result.

Fortunately, we do not need to find out an algorithm for each problem, because a large number of algorithms have been elaborated for solving a wide range of problems.

So, we may use a lot of algorithms from books and web sites as building blocks of our programs.

But we must always understand how the selected algorithm works.

Some basic concepts and definitions

Constants and variables

Data in an algorithm may behave in two different ways.

A datum whose value does not change in an algorithm is called *constant*.

A datum whose value may change in an algorithm is a *variable*.

A variable has three main attributes:

- its name,
- its type,
- and its value.

Some basic concepts and definitions

Variable name

- unambiguously identifies the variable,
- each variable has an individual name in an algorithm.

Variable type

- determines the domain of the variable
- and the set of operations applicable to the variable.

Value of the variable

- a variable always has an actual value in the algorithm,
- it can be changed by the operations.

Fundamental steps of algorithms

Algorithms can generally be made up of the following elemental steps:

- read,
- write,
- assignment,
- conditional branch,
- loop.

Read

an algorithm gets the input datum or data by means of one or more *read* steps

Write

an algorithm provides the output datum or data by means of one or more *write* steps (also known as *print* and *display*) for the outworld.

Fundamental steps of algorithms

Assignment

The symbol of assignment operator is generally the equation sign (=).

None but a variable may stand on the left hand side of an assignment.

The variable on the left hand side gets a value from the right hand side where variables and constants may appear.

In a simple case, the right hand side contains only a single variable or constant.

In a more complex case, different types of *operators* make connections between the neighbouring variables and/or constants (an operator is the symbol of a mathematical or logical operation).

That combination of variables, constants and operators is called *expression*.

Fundamental steps of algorithms

Examples:

```
x = 4
x = y + 5
```

<u>The following rules refers to the assignments:</u>

- before using a variable in an expression (on the right hand side), a value must be given to it in some previous step of the algorithm,
- the assignment which gives the first value to a variable in an algorithm is called *initialization* (of the variable),
- if a variable previously initialized stands on the left hand side, it will lose its value and receive a new value coming from the right hand side.

Example

```
y = 4
x = 5 * y
x = x -1
```

Fundamental steps of algorithms

Conditional branch

A conditional branch contains a condition and provides two potential branches for proceeding.

When the run of an algorithm arrives a conditional branch, the condition will be evaluated.

If the condition is fulfilled, the run of algorithm will continue along the true branch.

On the contrary, the false branch shows the way foreward.

Example:

Condition: $x \equiv 0$

if it is true, $y = 2$

if it is false, the value of y does not change

Fundamental steps of algorithms

Loop

Loops are used in algorithms to repeat a specific step or block of steps.

A loop has two parts:

- a *loop body* which contains one or more sequential steps (instructions)
- and a *loop control statement* which contains a condition.

After each execution of the loop body, the condition of the control statement will be evaluated.

If the condition is true, the loop body will be executed again.

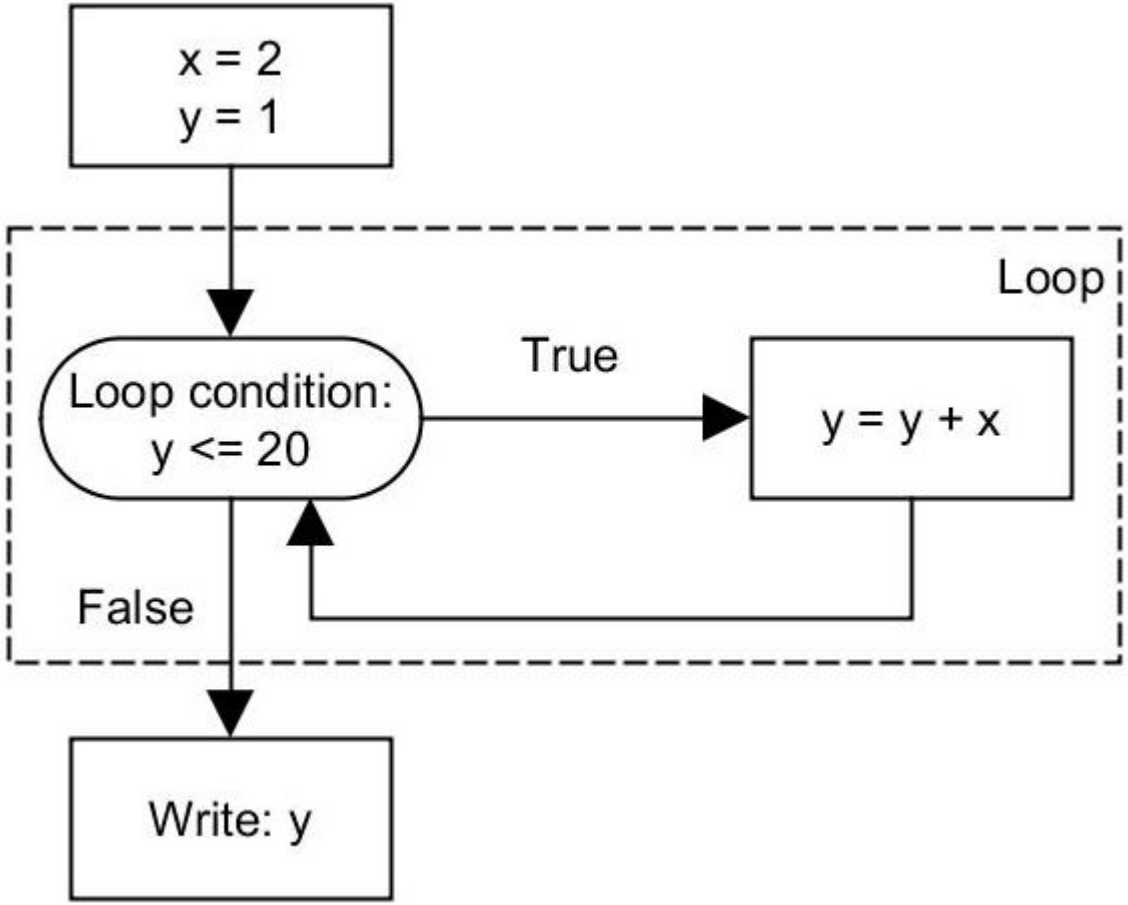
The condition generally contains a variable whose value changes in the loop body.

So, after each repetition of the loop body, another value of the variable will act in the condition.

The body of the loop will be repeated until the condition has become false.

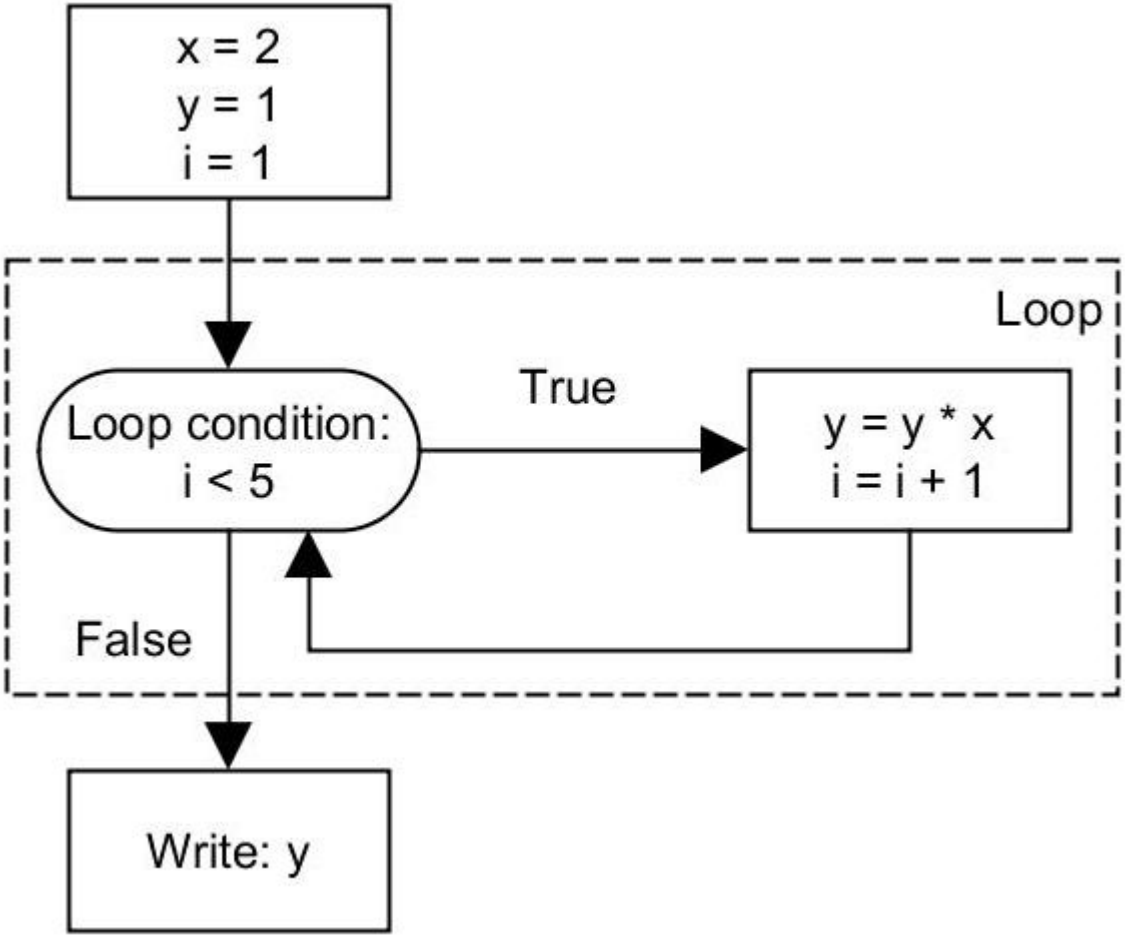
Fundamental steps of algorithms

Loop example 1



Fundamental steps of algorithms

Loop example 2



Ways of expressing algorithms

There are several ways of expressing algorithms.
The most frequently used ones are:

- description by means of natural languages,
- pseudocodes,
- and flowcharts.

Natural languages

A natural language is any language which has evolved naturally in human communities (e.g. Hungarian, Greek, Japanese etc.).
Natural languages are considered as unprecise but quick ways of expressing algorithms.
They are not suitable for the description of complex algorithms because they are verbose and ambiguous.

Ways of expressing algorithms

Example:

we would like to compute the value of the expression below for different values of the variable *x*:

$$y = Ax^3+Bx^2+Cx+D$$

step 1: take the values of constants *A*, *B*, *C* and *D*
step 2: take the value of *x*
step 3: compute the value of y
step 4: print the value of y
step 5: do you want to take another value of *x*?
step 6: if you want it, go to step 2, otherwise the algorithm stops

Ways of expressing algorithms

Unlike natural languages, pseudocodes and flowcharts are structured ways of expressing algorithms which help us to avoid the ambiguities common in natural language statements.

Pseudocodes

A pseudocode forms a transition between a natural (informal) language and a programming (formal) language.

It already takes into account the structural conventions of a normal programming language, but it is more comprehensible than a programming language.

A pseudocode concentrates on the steps of algorithms and it typically neglects the details essential for machine understanding of the algorithm (e.g. variable declarations, system-specific code and subroutines).

There is not any standard for pseudocodes, therefore they vary widely in style depending on their users.

Ways of expressing algorithms

The structures most frequently used in pseudocodes for process control:

Conditional branches:

if *condition* then

instruction block

meaning

if the condition is fulfilled, the instructions in the block will be executed

if *condition* then

instruction block 1

else

instruction block 2

meaning

if the condition is fulfilled, the instructions in block 1 will be executed, and if it is not, the instructions in block 2 will be executed.

Ways of expressing algorithms

Loops:

pre-test loop

while *condition* do

instruction block

meaning

while the condition is being fulfilled, the execution of instruction block repeats cyclically.

post-test loop

do

instruction block

while *condition*

meaning

the execution of instruction block repeats cyclically while the condition is being fulfilled.

Ways of expressing algorithms

Differences between pre-test and post-test loops:

- for pre-test loops, the loop control statement (includes the condition) precedes the loop body (includes the instruction block),
- for post-test loops, the loop control statement follows the loop body,
- if the condition is not fulfilled at the first test of a pre-test loop, the instruction block will never be executed,
- the instruction block of a post-test loop is executed on one occasion at least, because the first test of the condition follows the block,
- if the condition is fulfilled the block will repeat,
- if it is not, the process of the algorithm continue with the next step after the loop.

Ways of expressing algorithms

Loops:

for-loop

for *control variable* = *initial value* to *last value* do

instruction block

meaning

until the value of control variable belonging to the for-loop has reached the last value, the instruction block repeats.

The control variable begins with the initial value, and its value increases or decreases by one (depending on the last value) after each execution of the block.

There is another variant of for-loops which specifies the interval of increment or decrement for the control variable.

Ways of expressing algorithms

Example 1:
we would like to compute the value of the expression below for different values of the variable *x*:

$$y = Ax^3+Bx^2+Cx+D$$

read *A*, *B*, *C*, *D*
do
    read *x*
    $y = Ax^3+Bx^2+Cx+D$
    write *y*
    write „Do you want to take another value of x ? [yes/no]”
    read *i*
while *i* == „yes”

Ways of expressing algorithms

Example 2:
Let us compute the factorial of a positive integer *N*.

read *N*
fact = 1
for *i* = 1 to *N* do
        fact = *fact* * *i*
write *fact*

Ways of expressing algorithms

Flowchart

A flowchart is a type of diagram which represents an algorithm by showing its steps with geometrical shapes and the sequential connections with arrows.

The most frequently used symbols in flowcharts are the following

a rounded rectangle or an ellipse called terminal

It denotes the beginning and the end of algorithms.

It contains the word „START” or „STOP”.

Ways of expressing algorithms

a parallelogram called input/output

It symbolizes the points of an algorithmic workflow where receiving data and displaying processed data occure.

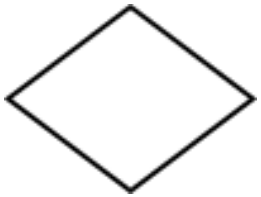
It contains the word „INPUT" or „OUTPUT" and the names of received or displayed variables.

a rectangle called operation

It is used to show that an operation is performed at a given point of the algorithm.

It contains the description of the operation (e.g. an assignment).

Ways of expressing algorithms



a diamond (rhombus) called decision

It shows the point of an algorithm where making a decision is required. It contains commonly a Yes/No question or a condition which implies a True/False test.

Generally, two arrows come out of a decision symbol. One of them starts from the bottom apex and the other joints to the right apex. These arrows correspond to the branches belonging to Yes/No or True/False responses. The arrows are always labelled.

an arrow called flow line

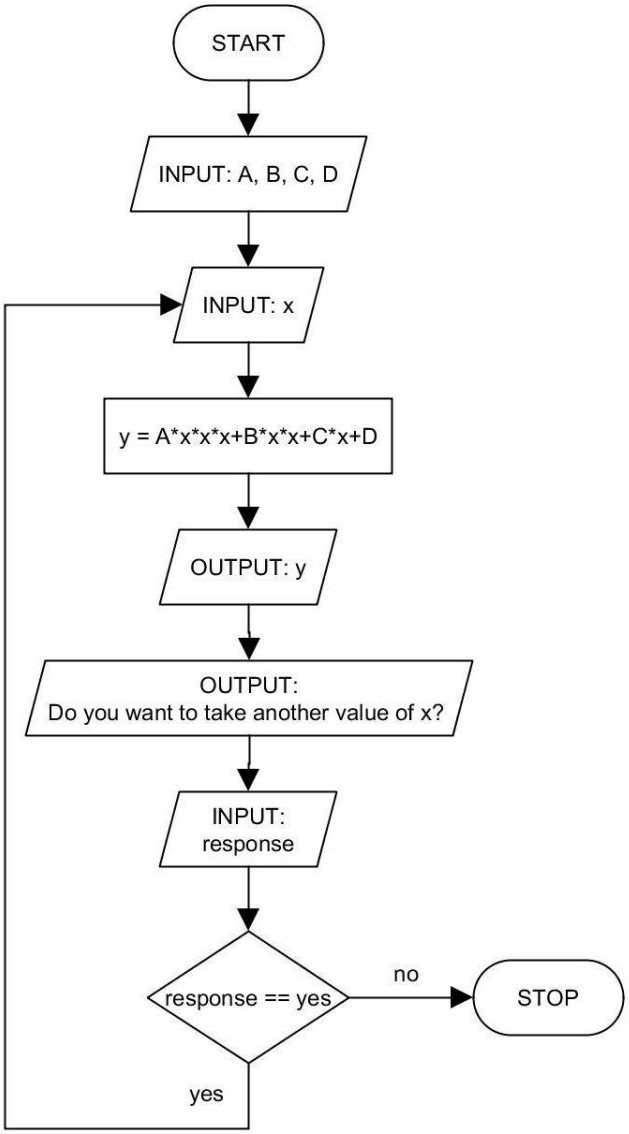
It starts from a symbol and ends at the following symbol.

It shows the order of the steps inside an algorithms.

Ways of expressing algorithms

Example 1:
we would like to compute the value of the expression below for different values of the variable x:

$y = Ax^3+Bx^2+Cx+D$



Ways of expressing algorithms

Example 2:

Let us compute
the factorial of a positive
integer *N*.