MODULE 2

Swati S
Assistant Professor
Department of CSE, GECSKP

SERIES 1 Exam
Time : 2 hrs
Total marks : 40

PART A : 12 marks

- MODULE 1 – 2 Questions (3 marks each)
- MODULE 2 – 2 Questions (3 marks each)

SERIES 1 Exam

PART B : 28 marks

- MODULE 1 – 1 Question : Including 2 sub parts (Total 14 marks)
- MODULE 2 – 1 Question : Including 2 sub parts (Total 14 marks)

- ALGORITHM AND PSEUDOCODE REPRESENTATION:- Meaning and Definition of Pseudocode, Reasons for using pseudocode, The main constructs of pseudocode - Sequencing, selection (if-else structure, case structure) and repetition (for, while, repeat-until loops), Sample problems*

- FLOWCHARTS** :- Symbols used in creating a Flowchart - start and end, arithmetic calculations, input/output operation, decision (selection), module name (call), for loop (Hexagon), flow-lines, on-page connector, off-page connector.

ALGORITHM

- An algorithm describes a systematic way of solving a problem. It is a step-bystep procedure that produces an output when given the necessary inputs.

- An algorithm uses pure English phrases or sentences to describe the solution to a problem.

- A Pseudocode is a high-level representation of an algorithm that uses a mixture of natural language and programming language-like syntax.
- It is more structured than an algorithm in that it uses mathematical expressions with English phrases to capture the essence of a solution concisely.
- You can also use programming constructs in a pseudocode which are not permitted in an algorithm in a strict sense.

- As the name indicates, pseudocode is not a true program and thus is independent of any programming language.
- It is not executable rather helps you understand the flow of an algorithm.

Algorithm vs Pseudocode

- Confused between algorithms and pseudocodes? Let us take an example.
- We will now write an algorithm and a pseudocode to evaluate an expression, say $d = a + b * c$.
- Here a, b, c, and d are known as variables.
- Simply put, a variable is a name given to a memory location that stores some data value. First, let us look at the algorithm for the expression evaluation.

Algorithm

```
Evaluate-Algo
1 Start
2 Read the values of a, b and c.
3 Find the product of b and c.
4 Store the product in a temporary variable temp.
5 Find the sum of a and temp.
6 Store the sum in d.
7 Print the value of d.
8 Stop.
```

- PS: Replace line numbers with step numbers

Pseudocode

The following is the pseudocode to evaluate the same expression.

```
Evaluate-Pseudo
1 Start
2 Read(a, b, c)
3 $d = a + b * c$
4 Print(d)
5 Stop
```

Syntax of a Pseudocode

- In a pseudocode, Read is used to read input values.
- Print is used to print a message.
- The message to be printed should be enclosed in a pair of double quotes. For example, Print("Hello folks!!") prints Hello folks!!
- To print the value of a variable, just use the variable name (without quotes). In the above example, Print(d) displays the value of the variable d.

- Although pseudocode and algorithm are technically different, these words are interchangeably used for convenience.

Advantages of Pseudocodes

1. Ease of understanding: Since the pseudocode is programming language independent, novice developers can also understand it very easily.

2. Focus on logic: A pseudocode allows you to focus on the algorithm's logic without bothering about the syntax of a specific programming language.

3. More legible: Combining programming constructs with English phrases makes pseudocode more legible and conveys the logic precisely.

4. Consistent: As the constructs used in pseudocode are standardized, it is useful in sharing ideas among developers from various domains

- 5. Easy translation to a program: Using programming constructs makes mapping the pseudocode to a program straightforward.
- 6. Identification of flaws: A pseudocode helps identify flaws in the solution logic before implementation

Constructs of a pseudocode

- A good pseudocode should follow the structured programming approach.
- Structured coding aims to improve the readability of pseudocode by ensuring that the execution sequence follows the order in which the code is written.
- Such a code is said to have a linear flow of control. Sequencing, selection, and repetition (loop) are three programming constructs that allow for linear control flow.
- These are also known as single entry – single exit constructs

- In the sequence structure, all instructions in the pseudocode are executed (exactly) once without skipping any.
- On the other hand, with selection and loop structures, it is possible to execute certain instructions repeatedly or even skip some.
- In such structures, the decision as to which statements are to be executed or whether the execution should repeat will be determined based on the outcome of testing a condition.

- We use special symbols called relational operators to write such conditions. The various relational operators are listed in Table 2.1.
- It is also possible to combine two or more conditions using logical operators like "AND" (&&), "OR" (||). Eg: a > b AND a > c

Sequence

This is the most elementary construct where the instructions of the algorithm are executed in the order listed. It is the logical equivalent of a straight line. Consider the code below.

```
S1
S2
S3
.
.
Sn
```

The statement S1 is executed first, which is then followed by statement S2, so on and so forth, Sn until all the instructions are executed.

Decision or selection

Example 2.1. The pseudocode CheckPositive(x) checks if an input value x is positive.

```
CheckPositive(x)
1 if (x > 0)
2         Print(x," is positive")
3 endif
```

if else

```
if (condition)
        true_instructions
else
        false_instructions
endif
```

This structure contains two blocks of statements. If the test condition is met, the first block (denoted by true_instructions) is executed and the algorithm skips over the second block (denoted by false_instructions).

If the test condition is not met, the first block is skipped and only the second block is
executed.

if else

Example 2.2. The pseudocode PersonType(age) checks if a person is a major or not.

```
PersonType(age)
if (age >= 18)
    Print("You are a major")
else
    Print("You are a minor")
endif
```

if else if else structure

When a selection is to be made out of a set of more than two possibilities, you need to use the if else if else structure, whose general form is given below:

```
if (condition1)
        true_instructions1
else if (condition2)
        true_instructions2
else
        false_instructions
endif
```

```
CompareVars(x, y)
1 if (x > y)
2         Print(x,“is greater than”,y)
3 else if (x < y)
4         Print(y,“is greater than”,x)
5 else
6         Print(“The two values are equal”)
7 endif
```

Case Structure

The case structure is a refined alternative to if else if else structure. The pseudocode representation of the case structure is given below.

- The general form of this structure is:

```
caseof (expression)
case 1 value1:
        block1
case 2 value2:
        block2
.
.
.
default :
        default_block
endcase
```

The pseudocode PrintDirection(dir) prints the direction name based on the value of a character called dir.

```
PrintDirection(dir)
caseof (dir)
    case ‘N’:
        Print(“North”)
        break
    case ‘S’:
        Print(“South”)
        break
    case ‘E’:
        Print(“East”)
        break
    case ‘W’:
        Print(“West”)
        break
    default :
        Print(“Invalid direction code”)
endcase
```

Repetition or loop

- When a certain block of instructions is to be repeatedly executed, we use the repetition or loop construct. Each execution of the block is called an iteration or a pass.
- If the number of iterations (how many times the block is to be executed) is known in advance, it is called definite iteration.
- Otherwise, it is called indefinite or conditional iteration. The block that is repeatedly executed is called the loop body. There are three types of loop constructs as discussed below:

while loop

- A while loop is generally used to implement indefinite iteration. The general
- form of the while loop is as follows:

```
while (condition)
        true_instructions
endwhile
```

Repeat until

- the loop body is repeated as long as condition evaluates to False. When
- the condition evaluates to True, the loop is exited.

- The pseudocode form of
- repeat-until loop is shown below.

```
repeat
        false_instructions
until (condition)
```

- There are two major differences between while and repeat-until loop constructs:
- 1. In the while loop, the pseudocode continues to execute as long as the resultant of the condition is True; in the repeat-until loop, the looping process stops when the resultant of the condition becomes True.
- 2. In the while loop, the condition is tested at the beginning; in the repeatuntil loop, the condition is tested at the end.

- For this reason, the while loop is known as an entry controlled loop and the repeat-until loop is known as an exit controlled loop.

- You should note that when the condition is tested at the end, the instructions in the loop are executed at least once

For loop

- All three for loop constructs use a variable (call it the loop variable) as a counter that starts counting from a specific value called begin and updates the loop variable after each iteration. The loop body repeats execution until the loop variable value reaches end. The first for loop variant can be written in pseudocode notation as follows:

```
for var = begin to end
        loop_instructions
endfor
```

Here, the loop variable (var) is first assigned (initialized with) the value begin.

Then the condition var <= end is tested. If the outcome is True, the loop body is executed. After the first iteration, the loop variable is incremented (increased by 1).

The condition var <= end is again tested with the updated value of var and the loop is entered (loop body is executed), if the condition evaluates to True.

This process of updating the loop variable after an iteration and proceeding with the execution if the condition (tested with the updated value of the loop variable) evaluates to True continues until the counter value becomes greater than end. At that time, the condition evaluates to False and the loop execution stops.

In the second for loop variant, whose pseudocode syntax is given below, the loop variable is decremented (decreased by 1) after every iteration.

Pseudocode Syntax - for loop

```
for var = begin to end by step
    loop_instructions
endfor
```

2.3 Solved problems - Algorithms and Flowcharts

Problem 2.1 To find simple interest.

Solution:

See Figure 2.1 for the algorithm and flowchart.

```
SIMPLEINTEREST
1   Start
2   READ($principal, rate, years$)
3   $SI = (principal * rate * years)/100$
4   PRINT($SI$)
5   Stop.
```

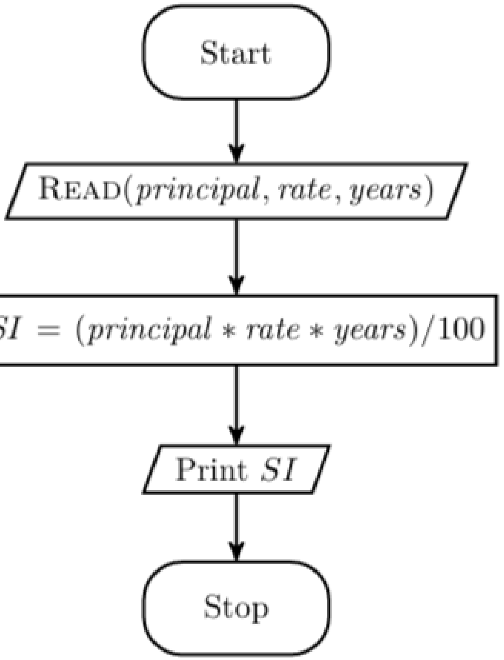

Start
READ(principal, rate, years)
SI = (principal * rate * years)/100
Print SI
Stop

Figure 2.1: To find simple interest

```
LargerTwo
1  Start
2  Read($a, b$)
3  if ($a > b$)
4        $large = a$
5  else
6        $large = b$
7  endif
8  Print($large$)
9  Stop.
```

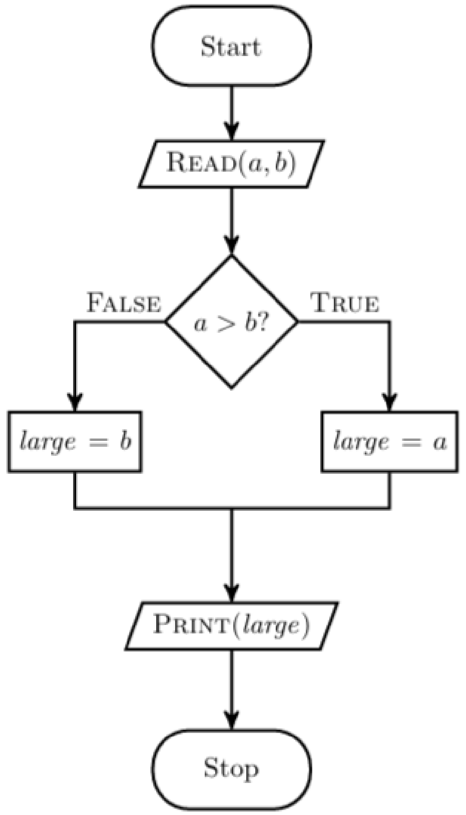


Figure 2.2: To find the larger of two numbers

```
SMALLESTTHREE
1    Start
2    READ(a, b, c)
3    if (a < b)
4        small = a
5    else
6        small = b
7    endif
8    if (c < small)
9        small = c
10   endif
11   PRINT(small)
12   Stop.
```

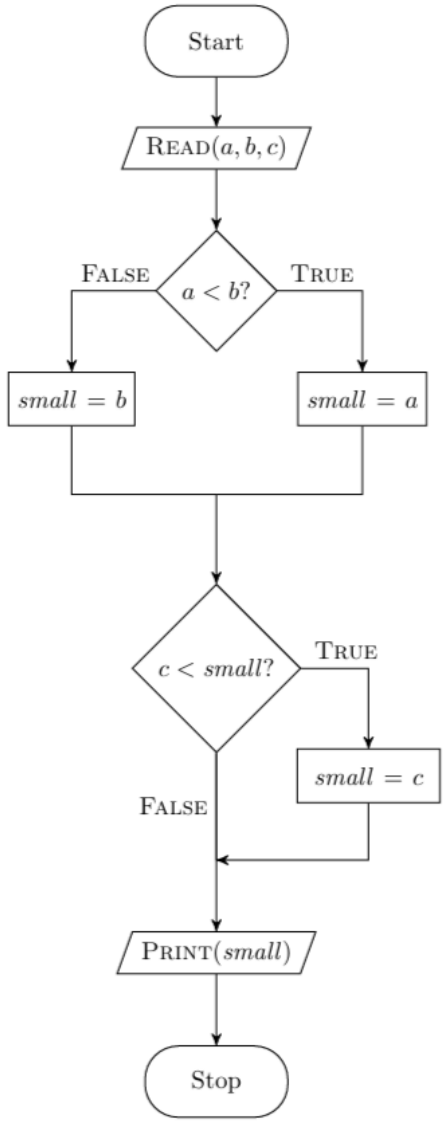


Figure 2.3: To find the smallest of three numbers

See Figure 2.4 for the pseudocode and flowchart.

```
TICKETFARE
1   Start
2   READ(age)
3   if (age < 10)
4           fare = 7
5   else if (age < 60)
6           fare = 10
7   else
8           fare = 5
9   endif
10  PRINT(fare)
11  Stop
```

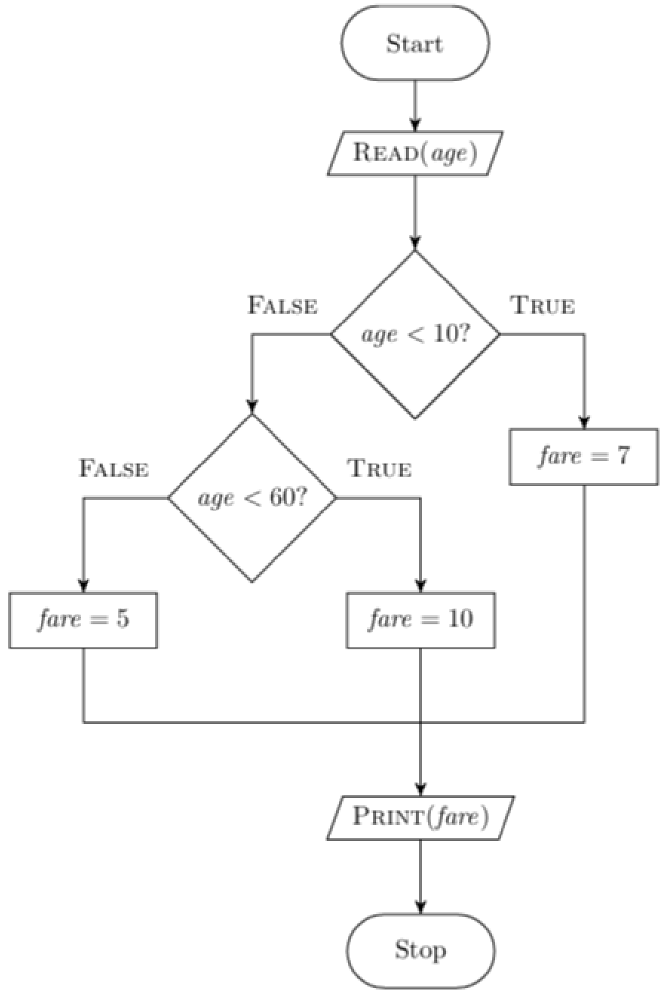


Figure 2.4: To determine the entry fare in a zoo

```
PrintColour
Start
Read(code)
caseof (code)
    case 'R':
        Print("Red")
        break
    case 'G':
        Print("Green")
        break
    case 'B':
        Print("Blue")
        break
    default :
        Print("Wrong code")
endcase
Stop
```

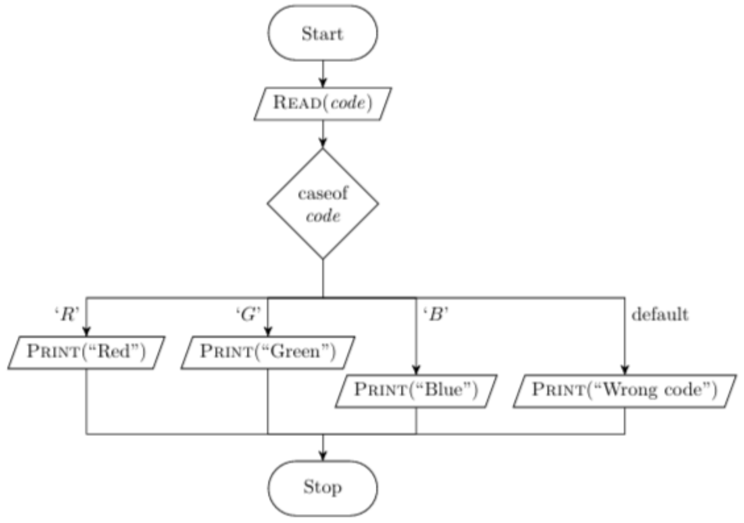


Figure 2.5: To print colors based on a code value

Problem 2.6 To print the numbers from 1 to 50 in descending order.

Solution:

See Figure 2.6 for pseudocode and flowchart.

```
PRINTDOWN
1  Start
2  for count = 50 downto 1
3      PRINT(count)
4  endfor
5  Stop
```

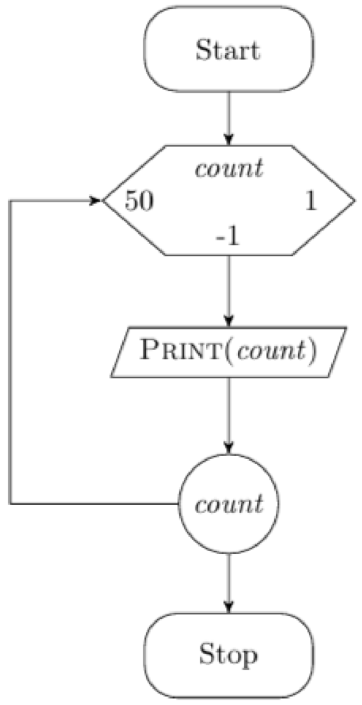


Figure 2.6: To print numbers in descending order

Problem 2.7 To find the factorial of a number.

Solution: The factorial of a number n is defined as $n! = n \times n-1 \times \cdots\cdots \times 2 \times 1$.

See Figure 2.7 for the pseudocode and flowchart.

```
FACTORIAL
1  Start
2  READ(n)
3  fact = 1
4  for var = n downto 1
5        fact = fact * var
6  endfor
7  PRINT(fact)
8  Stop
```

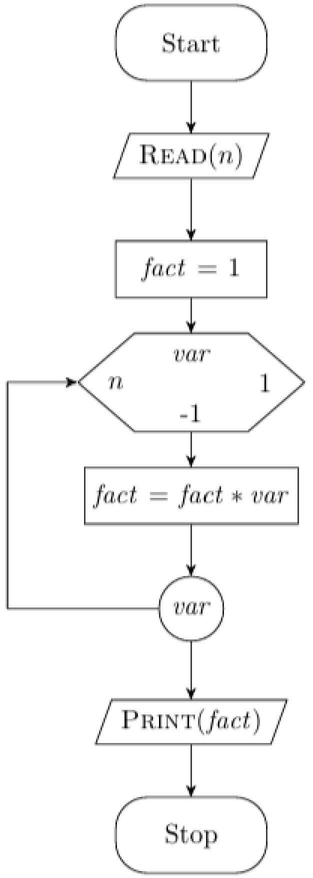


Figure 2.7: To find the factorial of a number

Problem 2.8 To determine the largest of n numbers.

Solution: See Figure 2.8 for pseudocode and flowchart.

```
LARGEN
1   Start
2   READ($n$, $num$)
3   $large = num$
4   for $count = 1$ to $n - 1$
5       READ($num$)
6       if ($num > large$)
7           $large = num$
8       endif
9   endfor
10  PRINT($large$)
11  Stop
```

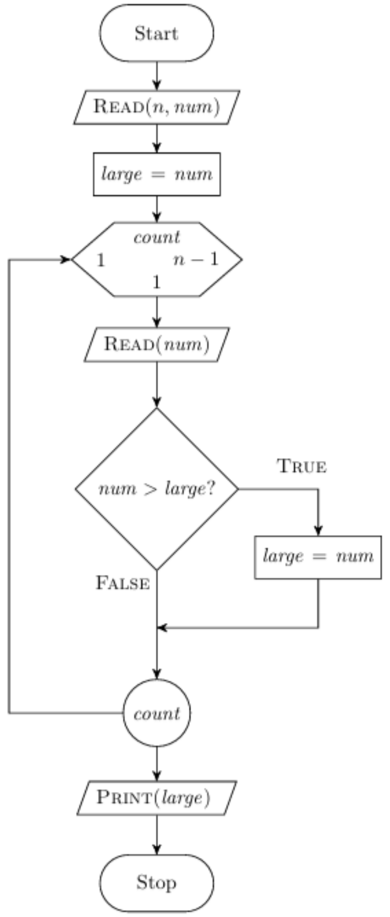


Figure 2.8: To find the largest of n numbers

Problem 2.9 To determine the average age of students in a class. The user will stop giving the input by giving the age as 0.

Solution: See Figure 2.9 for pseudocode and flowchart.

```
AVERAGEAGEV1
1   Start
2   $sum = 0$
3   $count = 0$
4   READ($age$)
5   while ($age$!=0)
6       $sum = sum + age$
7       $count = count + 1$
8       READ($age$)
9   endwhile
10  $average = sum/count$
11  PRINT($average$)
12  Stop
```

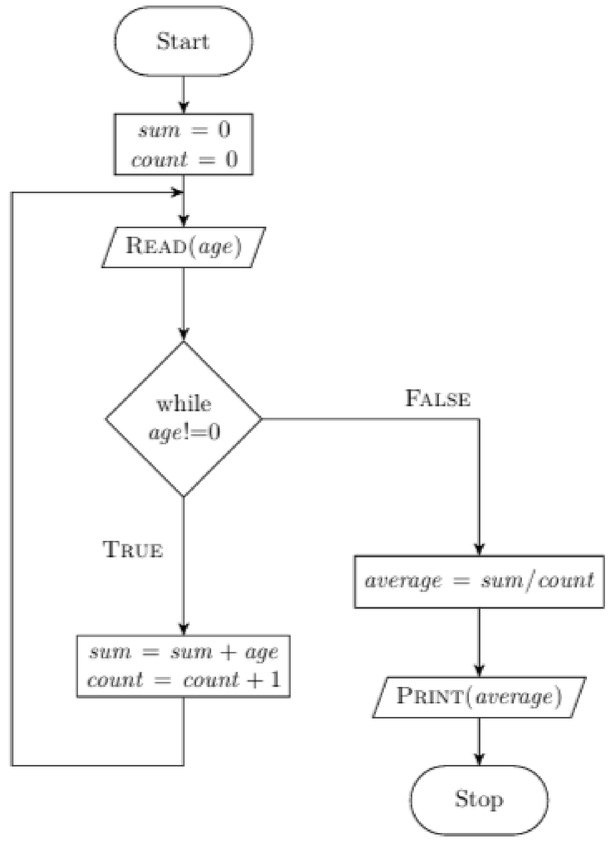
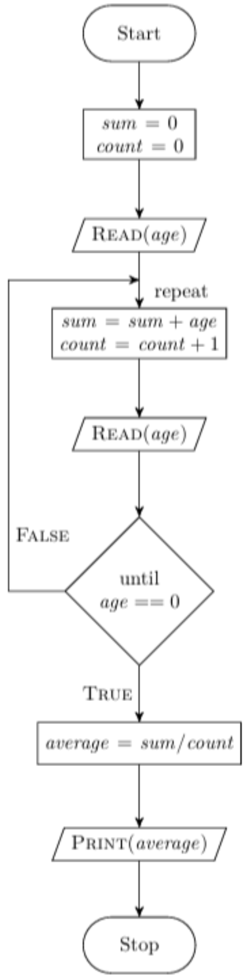


Figure 2.9: To determine the average age using **while** loop

Problem 2.10 Redo Problem 2.9 using repeat-until loop construct.

Solution: See Figure 2.10 for flowchart and pseudocode.

```
AVERAGEAGEV2
1   Start
2   $sum = 0$
3   $count = 0$
4   READ($age$)
5   repeat
6       $sum = sum + age$
7       $count = count + 1$
8       READ($age$)
9   until ($age == 0$)
10  $average = sum/count$
11  PRINT($average$)
12  Stop
```


Start
sum = 0
count = 0
READ(age)
repeat
sum = sum + age
count = count + 1
READ(age)
FALSE
until
age == 0
TRUE
average = sum/count
PRINT(average)
Stop