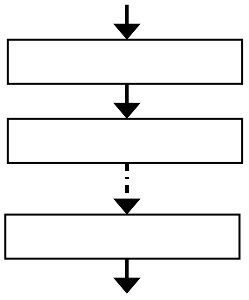
1. Basic Algorithmic Control Structures

   1.1. Sequence Control Structure

A sequence control structure executes a set of instructions one after the other from the first to the last in the order they are given.

Syntax:
```
Instruction 1
Instruction 2
…
Instruction n
```

```
Get a Get
b c ← a
+ b
Print "Sum =", c
End
```



Flow chart representation: Sequence

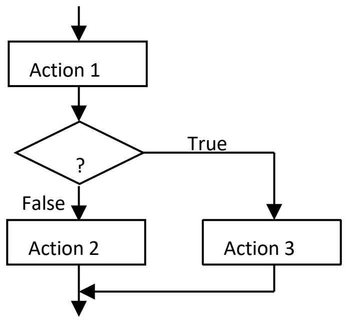
   1.2. Selection Control Structure

A selection control structure (condition control structure) chooses the instruction or set of instructions to be executed based on the validity of a certain condition. Examples: If …then else and case … of.

      1.2.1. If… then … else

Syntax:
```
IF condition THEN
    Instruction 1
ELSE
    Instruction 2
```

Ex.
```
Get a, b
If a = 0 then
  Print "Error: division by zero"
Else
  Print b/a
```



Explanation *Condition* is a Boolean expression meaning that it can take only one of two values true or false. The condition is evaluated, if it is true, instruction 1 is executed. If it is false, instruction 2 is executed. Note that instructions 1 and 2 could be compound instructions.

It is possible to nest many selection structures.

Syntax:
```
If condition1 then
   If condition2 then
      Instruction 1
   Else
      Instruction 2
Else
   Instruction 3
```

E x.
```
Get a, b
If a <> 0 then
  If b <> 0 then
    Print b/a
  Else
    Print "Answer is 0"
Else
  Print "Error: division by 0"
End
```

Explanation If condition1 is true, we move to condition2. If condition2 is true, then instruction 1 is executed otherwise, instruction 2 is executed. If condition 1 is false, instruction 3 is executed. Instruction 1 or instruction 2 will be executed if and only if condition 1 is true.

1.2.2. Case … of

Syntax:
```
Case variable of
      Case 1: Instruction 1
      Case 2: Instruction 2
        …
      Case n: Instruction n
   End
```

E x.
```
Get day
Case day of
   1: print "today is Monday"
   2: print "today is Tuesday"
           …
   7: print "today is Sunday"
End
```

Explanation The value of variable is evaluated, if it matches with case 1, instruction 1 is executed. If it matches with case 2, instruction 2 is executed and so on. Case…of is a multiple selection structure. It is used when an important number of choices are to be considered depending on the value of a variable.

1.3. Repetition Control Structure

The repetition (iteration) control structure executes an instruction or set of instructions many times until a certain condition is reached or while a condition is true. Repetition structures define the order of operations and the number of repetitions. They are also known as loops. Examples are, while…do, repeat…until, for…to…do.

1.3.1. While Loop

Syntax:
```
While condition do
      Instruction(s)
   End while
```

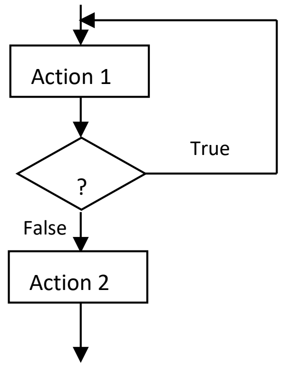
E x.
```
Get n
 i ← 1
 While i <= n do
     Print "this is a while loop"
     i ← i + 1
   End while
```

Explanation The condition is evaluated, if it is true instruction(s) is/are executed. Instruction(s) is/are executed as long as condition remains true. When the condition becomes false, the loop stops.
The condition for the loop to stop comprises of a variable called control or iteration variable whose value must change at the end of each execution of the loop.
In the example above, the control variable is "i".

1.3.2. Repeat Loop

Syntax:
```
Repeat
      Instruction(s)
   Until condition
```

E x.
```
Get n
 i ← 1
 Repeat
     Print "this is a repeat loop"
     i := i + 1
   Until i <= n
```



Explanation The instruction or set of instructions is executed and the condition is evaluated.
If it evaluates to false, the instruction or set of instructions is executed again. If condition evaluates to true, the program exits the loop.

Remark! The Repeat until loop must be executed at least once as the condition is evaluated only at the end of the loop.

1.3.3. For Loop

Syntax:

```
For var := low_limit to hi_limit do
    Instruction(s)
```

Or

```
For var := hi_limit downto low_limit do
    Instruction(s)
```

E x.

```
Get n
For i := 1 to n do
  Print “this is a for loop”
End For
                    Or
Get n
For i:= n downto 1 do
  Print “this is a for loop”
End For
```

Explanation var (variable) is given a value low limit or hi_limit depending on the loop, which is automatically incremented or decremented (by 1) after any execution of the loop. The loop stops when low_limit becomes greater than hi_limit. In both cases, if hi_limit is less than low_limit, the loop body is not executed at all.

Exercise! Exercise! Exercise!

1) Write an algorithm that reads a value n and writes the first n numbers.
2) Write an algorithm to calculate the area of a circle.
3) Write an algorithm to solve a linear equation
4) Write an algorithm that reads a person’s sex and writes good morning sir if it’s male and good morning madam if it’s female
5) Write an algorithm that reads two numbers and an operator and returns the value of the operation.

2. Recursion

Some problems are recursive in nature. The solution to such problems involves the repeated application of the solution to its own values until a certain condition is reached. Algorithms for such problems are known as recursive algorithms.
A recursive algorithm is an algorithm that calls (invokes) itself during its execution. Examples are the factorial function and the sum function.
Recursion can be defined as the calling of a procedure by itself, creating a new copy of the procedure.

2.1. Factorial Function

Factorial is defined as:
$1! = 1$
$2! = 2 \times 1 = 2$
$3! = 3 \times 2 \times 1 = 6$
$4! = 4 \times 3 \times 2 \times 1 = 24$
$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$
…
$n! = n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1$

By studying the above equations closely, we see that the factorial of any number n can be calculated by multiplying the number by the factorial of the preceding number.
We therefore have:

$1! = 1$
$2! = 2 \times 1!$
$3! = 3 \times 2!$
$4! = 4 \times 3!$
$5! = 5 \times 4!$
…
$n! = n \times (n-1)!$

Factorial is defined recursively as:

$$\Bigg\{$$

```
Get n
If $n = 0$ or $n = 1$ then
  Print "answer is 1"
Else
  $fact := n \times fact(n-1)$
  Print fact
```

$\boxed{\text{Remark:}}$ $n! = 1$ is known as the base case. Every recursive problem must always have some base case which can be solved without recursion. For cases that are to be solved recursively, the recursive call must always be a case that makes progress towards the base case.

2.2. The Sum Function

The sum function is a function that calculates the sum of the first n integers. For example we want to calculate the sum of the first 5 integers 1, 2, 3, 4 and 5. Their sum is calculated as follows:

$Sum = 1 + 2 + 3 + 4 + 5$

We can see that for any number n, the sum is the number n plus the sum of the previous numbers. The sum function can therefore be defined recursively as:

$sum(1) = 1$
$sum(2) = 2 + sum(1)$
$sum(3) = 3 + sum(2)$
$sum(4) = 4 + sum(3)$
…
$sum(n) = n + sum(n-1)$

The base case is $n = 1$ which gives $sum(1) = 1$

```
Get $n$
If $n = 1$ then
  Print "sum is 1"
Else
  $sum := n + sum(n-1)$
  Print sum
```