

Programming

A-Level Computer Science 9618 · Topic 11 revision notes

Everything from the 4 lessons of this topic on a few pages: the key idea of each lesson, the diagram to remember, the points to learn and the traps examiners set. Read it, cover it, test yourself.

1. Variables, constants and built-in functions
2. Selection and iteration constructs
3. Procedures and functions
4. Passing parameters by value and by reference

thelucidstem.com

1 Variables, constants and built-in functions

Syllabus 11.1

A **variable** is a named memory location whose value can change while the program runs; declare it with `DECLARE Name : TYPE` and give it a value with the **assignment** operator `←`. A **constant** is declared once with `CONSTANT Name = value`, using a **literal** only, and can never be changed. **Expressions** combine values with arithmetic operators (`+`, `-`, `*`, `/`, `DIV`, `MOD`), relational operators and the logical operators **AND**, **OR**, **NOT**. **Built-in functions** and **library routines** are pre-written, pre-tested subroutines such as `LENGTH`, `MID` and `INT`.

each flowchart symbol becomes one pseudocode statement

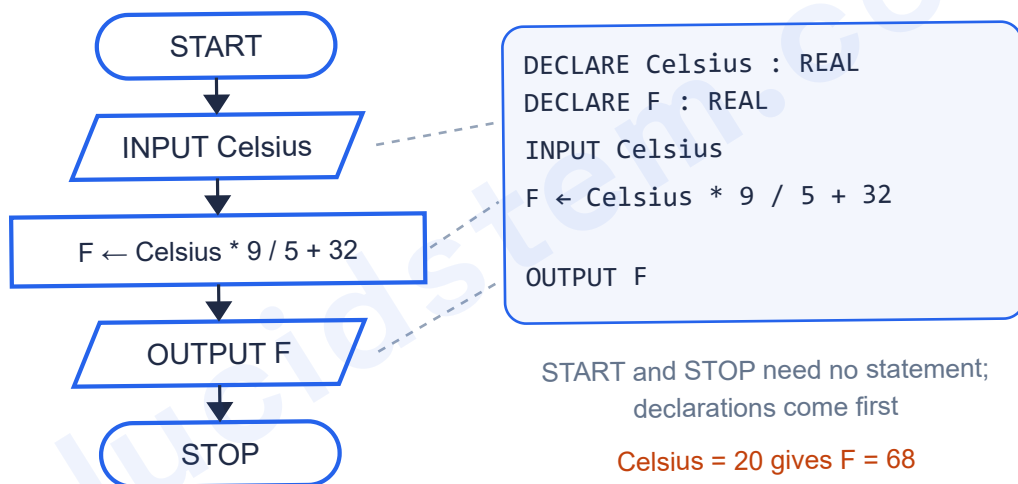


Fig. 1: converting a program flowchart into pseudocode, one symbol at a time

Worked example: Structured English to pseudocode

Convert this design into pseudocode. 1. SET the hourly rate to 12.50 (it never changes). 2. INPUT the hours worked. 3. CALCULATE the pay as hours multiplied by the rate. 4. OUTPUT the pay. Then state the output when 6 hours are entered.

- Step 1 is a value that never changes, so it is a constant: `CONSTANT HourlyRate = 12.50`.
- Declare the variables with suitable types: `DECLARE Hours : INTEGER` and `DECLARE Pay : REAL`.
- Steps 2 and 3: `INPUT Hours` then `Pay ← Hours * HourlyRate`.
- Step 4: `OUTPUT "Pay: ", Pay`. With 6 hours, $6 \times 12.50 = 75.00$.

Answer:

```
CONSTANT HourlyRate = 12.50
DECLARE Hours : INTEGER
DECLARE Pay : REAL
INPUT Hours
Pay ← Hours * HourlyRate
OUTPUT "Pay: ", Pay
```

With 6 hours the output is Pay: 75.00 (many languages show it as 75.0).

Literal, operator, function.

- ✓ **Constant:** `CONSTANT Name = literal`, declared at the start, never assigned again.
- ✓ **Division:** `/` gives a REAL; `DIV` gives the whole-number quotient; `MOD` gives the remainder.
- ✓ **Strings:** positions start at 1; `MID(s, start, length)`; join with `&`.
- ✓ **Random integers:** `INT(RAND(n))` gives 0 to $n - 1$; add the lowest value you want.

Watch out!

`CONSTANT VAT = Price * 0.2` is not allowed: a constant must be a literal. And a constant can never appear on the left of `←` later in the program.

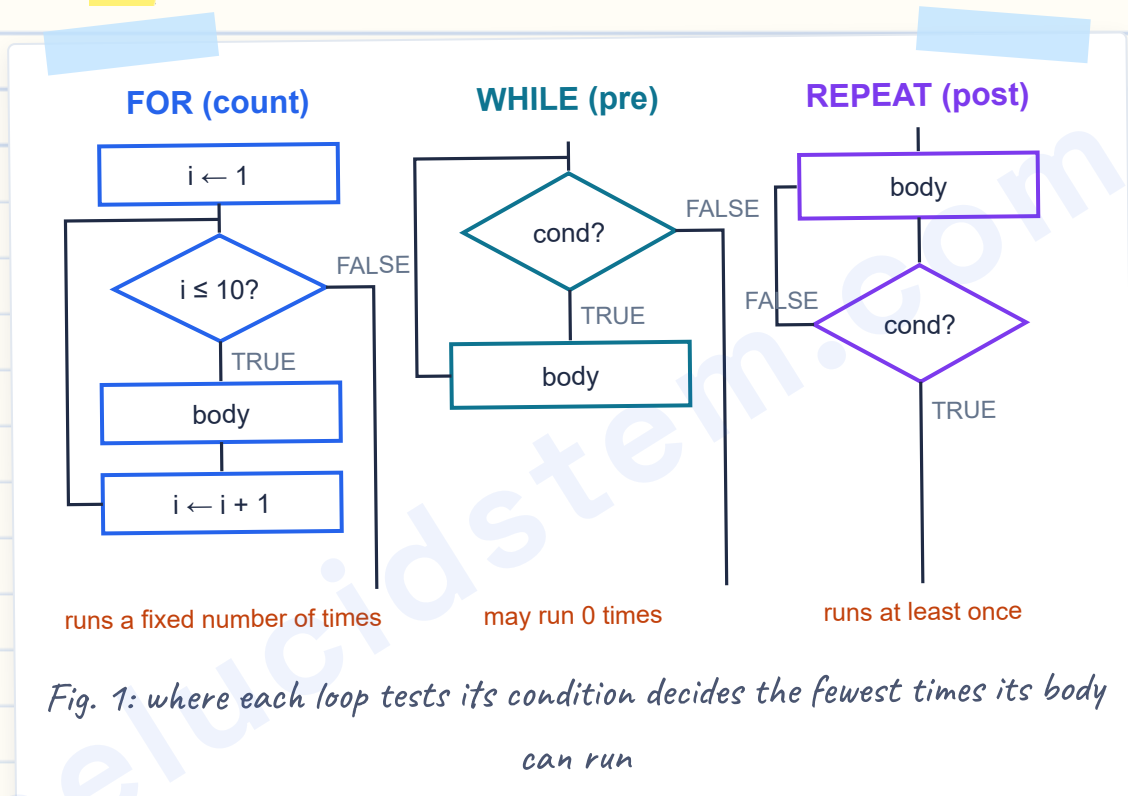
`17 / 5` is 3.4, not 3. If a question says the result must be an INTEGER, the mark is for `DIV` (or `INT`), not `/`.

In the 9618 guide `UCASE` and `LCASE` take a single `CHAR`, and string positions start at 1. To change a whole string, loop through it with `MID` one character at a time.

2 Selection and iteration constructs

Syllabus 11.2

Selection uses **IF ... THEN ... ELSE ... ENDIF** (which can be **nested**) or **CASE OF ... OTHERWISE ... ENDCASE**, whose clauses are tested in order until one matches. **Iteration** has three forms: a **count-controlled** loop (**FOR ... NEXT**) repeats a known number of times; a **post-condition** loop (**REPEAT ... UNTIL**) tests at the end, so it always runs **at least once**; a **pre-condition** loop (**WHILE ... ENDWHILE**) tests at the start, so it may run **zero** times.



Worked example: Dry-run the delivery charge

Give the output for (a) Weight 14.0, Express FALSE and (b) Weight 2.0, Express TRUE.

1. (a) $14.0 \leq 2.0$ is FALSE, so the ELSE runs; $14.0 \leq 10.0$ is FALSE, so the inner ELSE runs.
2. $\text{Charge} \leftarrow 7.50 + (14.0 - 10.0) \times 0.50 = 7.50 + 2.00 = 9.50$. Express is FALSE, so no change.
3. (b) $2.0 \leq 2.0$ is TRUE (it is a boundary value), so $\text{Charge} \leftarrow 4.00$.
4. Express is TRUE, so $\text{Charge} \leftarrow 4.00 \times 2 = 8.00$.

Answer: (a) Delivery charge: 9.50 (b) Delivery charge: 8.00

Test first or test last.

- ✓ **IF:** THEN on the IF line, one ENDIF per IF; check the boundary values.
- ✓ **CASE:** one variable, values or ranges, first match wins, OTHERWISE last.
- ✓ **FOR** = known count; **REPEAT** = at least once; **WHILE** = maybe zero times.
- ✓ **Justify** a loop by saying why its test position suits this problem, not just what the loop is.

Watch out!

Check boundary values. Changing $\leq$ to $<$ moves a 2.0 kg parcel into the next price band, and examiners test exactly that value.

CASE stops at the first match. Overlapping ranges are not an error, but every value is caught by the earlier range, so the later clause is dead code.

The UNTIL condition is the condition to stop; the WHILE condition is the condition to keep going. Converting one loop into the other means reversing the condition.

3 Procedures and functions

Syllabus 11.3

A **procedure** is a named block of code that performs a task; it may have none, one or more **parameters** and is run with a **CALL** statement. A **function** also performs a task but **returns a single value**; its header ends **RETURNS** data type, it uses **RETURN**, and it is used **inside an expression**, where the **return value replaces the call**. The values passed in a call are **arguments**; the identifiers in the header that receive them are **parameters**. Subroutines give **modular**, reusable code that is easier to test and maintain.

a procedure call is a statement; a function call is part of an expression

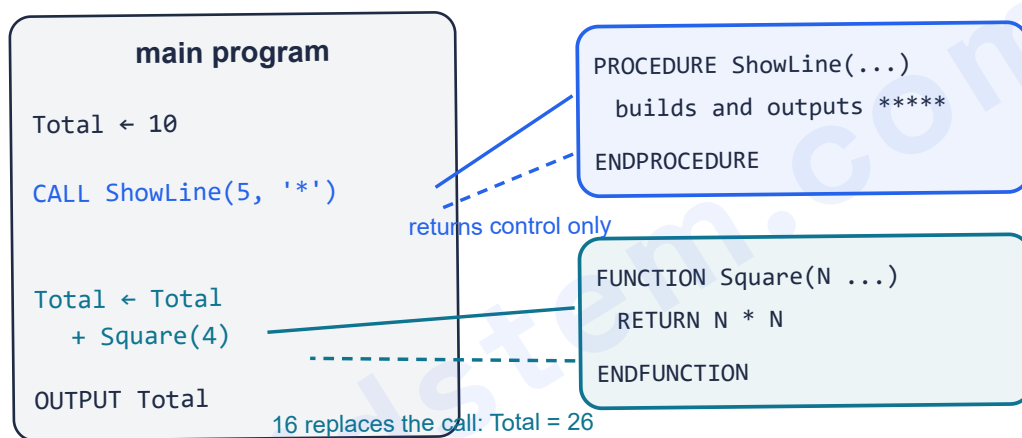


Fig. 1: control passes to the subroutine and back; only a function brings a value back with it

Worked example: Follow the calls

State exactly what the last two lines of the program above output.

1. **CALL ShowBanner()** runs ShowBanner, which first calls ShowLine with Length 20 and Symbol '='.
2. ShowLine joins '=' to Line 20 times and outputs a row of 20 = signs.
3. ShowBanner then outputs the title and calls ShowLine(20, '=') again.
4. Control returns to the main program: **CALL ShowLine(5, '*')** outputs *****.

Answer: A row of 20 = signs, the line Weekly quiz results, another row of 20 = signs, then *****

Statement or expression.

- ✓ **Procedure:** `PROCEDURE Name(...)`
... `ENDPROCEDURE`; run with `CALL`
`Name(...)`.
- ✓ **Function:** header ends `RETURNS`
`type`; uses `RETURN`; used inside an
expression, never with `CALL`.
- ✓ **Parameter** is in the header; **argument**
is in the call; they match in number,
order and type.
- ✓ **Efficient:** stop loops early, do not
repeat calculations, replace copied code
with a subroutine.

Watch out!

A procedure does not return a value, so `X ← ShowLine(5, '*')` is wrong. If the main program needs a result back, write a function instead.

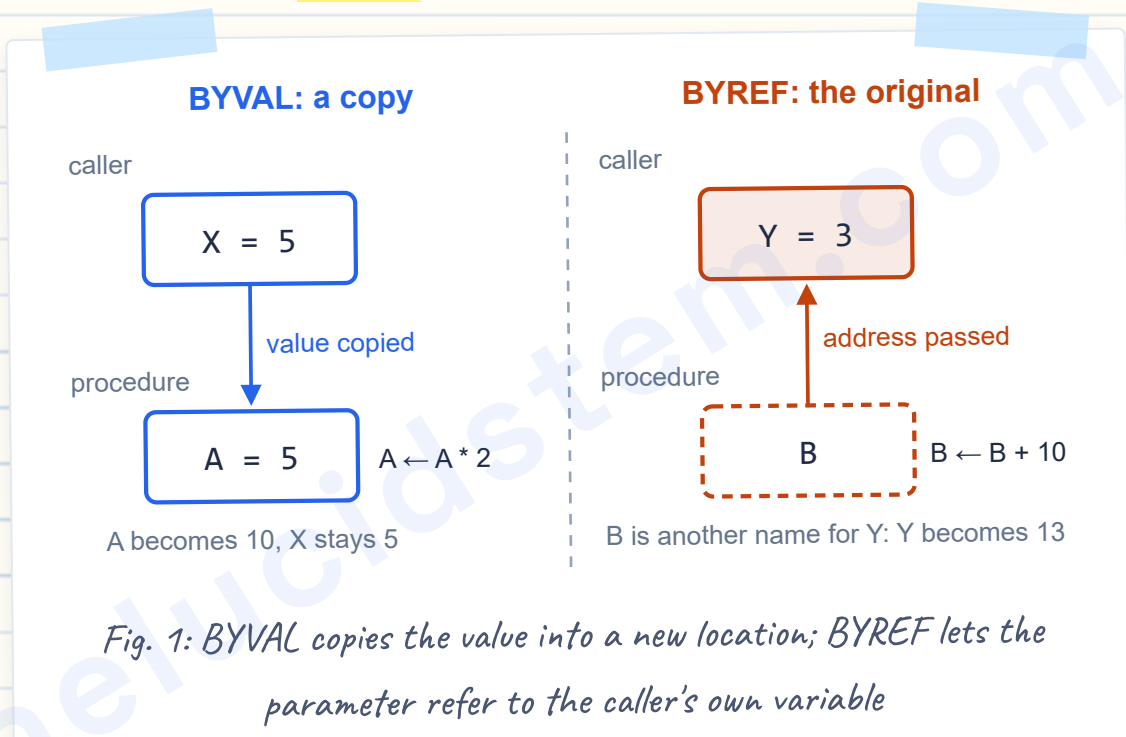
`CALL CountChar("banana", 'a')` loses the mark: a function is never `CALL`ed, and its return value would be thrown away.

Do not mix up parameter and argument. The parameter is the name in the header (`Length`); the argument is what is passed in the call (`4.0`).

4 Passing parameters by value and by reference

Syllabus 11.3

Passing **by value** (BYVAL) gives the subroutine a **copy** of the argument's value: changes to the parameter are **local** and the caller's variable is **unchanged**. Passing **by reference** (BYREF) passes the **address** of the caller's variable, so the parameter is another name for the same memory location and **any change is made to the original**. If neither keyword is given, passing **by value** is assumed. In 9618 pseudocode, parameters should **not** be passed by reference to a **function**.



Worked example: Trace a BYVAL call

Trace the program above and give both lines of output.

1. Score ← 7 in the main program.
2. CALL Double(Score): the value 7 is copied into the parameter N.
3. N ← 7 × 2 = 14 and "Inside: 14" is output. Only the copy changed.
4. The procedure ends, N is discarded, and "Outside: 7" is output.

Answer:

Score N OUTPUT

7

7

14 Inside: 14

Value in, reference out.

- ✓ **BYVAL:** a copy; changes stay inside the subroutine; the default if nothing is stated.
- ✓ **BYREF:** the address; the parameter is the caller's variable; the argument must be a variable.
- ✓ **One keyword** applies to all the following parameters until the other keyword appears.
- ✓ **Functions** take BYVAL parameters only and return one value; use BYREF in a procedure to pass back several values.

Watch out!

"The procedure changes N, so Score changes" is the classic wrong answer. With BYVAL the procedure only ever touches its own copy.

In `PROCEDURE Swap(BYREF X : INTEGER, Y : INTEGER)`, Y is passed by reference too. Saying Y is passed by value loses the mark.

If a question says a procedure must "return" two values, the answer is BYREF parameters, not two RETURN statements: a procedure has no RETURN value, and a function returns only one value.