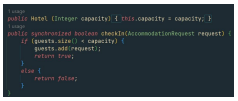
Programming

AS Computer Science ▪ Topic 11

A-Level Computer Science

What we will cover

1. Programming basics
2. Selection
3. Iteration
4. Procedures & functions
5. Efficient code
6. Recap



1 Basics

From design to code

Turn a **flowchart** 流程图 or structured English into **pseudocode** 伪代码, then into a language.

input or output box	→ `INPUT / OUTPUT`
decision diamond	→ `IF…ELSE…ENDIF`
loop arrow	→ `WHILE / FOR / REPEAT`
process box	→ assignment or calculation

> Then trace a small input to check. Find the variables and their types first.

Design, then code

flowchart symbol	pseudocode
input / output	`INPUT / OUTPUT`
decision	`IF...THEN or CASE`
process	`x = expression`
loop arrow	`WHILE / FOR / REPEAT`

Structure diagram, then pseudocode, then a language – each step commits to a little more detail, and only the last one is language-specific.

> Work out the algorithm before you type. A design fixed on paper costs minutes; the same fix after the code is written costs hours.

Variables, operators, functions

- **constant** 常量 fixed; **variable** 变量 changes
- assign with ←; `DIV`, `MOD`
- **precedence** 优先级: `NOT` > ×/ > +− > compare > `AND` > `OR`

```
s =  1 2 3 4 5 6 7 8
     C O M P U T E R

LENGTH(s) = 8      RIGHT(s, 2) = "ER"
LEFT(s, 3) = "COM"  UCASE(s) = "COMPUTER"
MID(s, 4, 3) = "PUT"  LCASE(s) = "computer"
```

library routines 库例程: *LEFT, MID, LENGTH*

When to use a subroutine

- the same logic appears in **more than one place** – write it once, call it many times.
- a block has a **clear named purpose** – the name documents what it does.
- the program is **complex** – break it into parts (**decomposition** 分解).
- you want to **test a piece in isolation**.

Terminology

- **definition** – the `PROCEDURE ... ENDPROCEDURE` (or function) block.
- **call** – where it is invoked. **argument** – a value passed in. **parameter** – the variable that receives it.
- **return value** – what a function passes back.
- **procedure/function header** – the first line giving the name and parameters (`PROCEDURE Name(params)` or `FUNCTION Name(params) RETURNS type`).

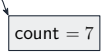
The words a program specification uses

Data types	**Data types** 数据类型: INTEGER, REAL, CHAR, STRING, BOOLEAN, DATE – and the type decides what operations are legal.
Array	An **array** 数组 holds many values of one type under one identifier, reached by index – which is what lets a loop process them all.
Arguments	Parameters are named in the definition; **arguments** 实参 are the values passed at the call. By value copies; by reference lets the routine change the original.
Linear search	A **linear search** 线性查找 checks each element in turn – slow on large data, and the only option when the list is unsorted.

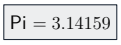
Declare the type, name the parameters, say how they are passed. A specification missing any one of those cannot be implemented from.

Assignment is not equality

```
count = count + 1
```



variable
(can change)



constant
(fixed)

The right-hand side is evaluated first, then stored in the variable named on the left – which is why `x ← x + 1` makes sense.

`x ← 5` stores 5 in x; `x = 5` asks whether x holds 5. Using one where the other belongs is the commonest slip in a trace table.

Operator precedence, and the built-in functions

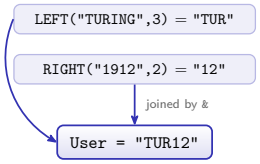
precedence 优先级	`NOT` → `*` `/` `DIV` `MOD` → `+` `-` → comparisons → `AND` → `OR`
the advice	use brackets whenever the order is not obvious to a reader
numeric	`INT(x)`, `ROUND(x)`, `ABS(x)`, `MOD(a,b)`, `RANDOM()`
string	`LENGTH`, `SUBSTRING`, `UCASE`, `LCASE`, and concatenation with `&`
conversion	`STR_TO_NUM` and `NUM_TO_STR` – needed whenever input is read as text
selection 选择	chooses which steps run: `IF` for a condition, `CASE` for many values of one variable

`AND` binds tighter than `OR`, so `a OR b AND c` means `a OR (b AND c)`. That is the precedence question the paper actually sets.

Worked example: build a username

```
Name ← "TURING"
Year ← "1912"
User ← LEFT(Name,3)
       & RIGHT(Year,2)
```

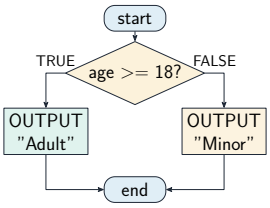
What ends up in `User`?



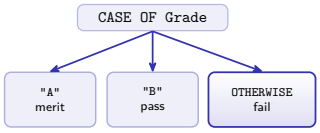
String positions count from 1. `LEFT`/`RIGHT`/`MID` cut pieces; `&` joins them.

2 Selection

Selection



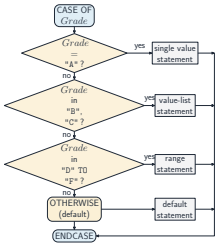
IF/ELSE: one branch runs



CASE: match one value, many options

Use CASE instead of deep **nested** 嵌套 IFs when testing one value against many.

CASE picks one branch by value

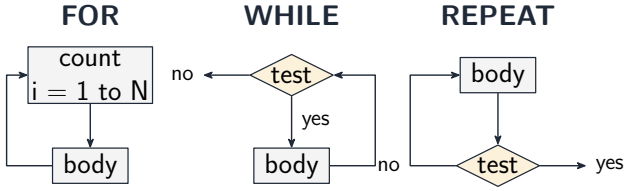


> A CASE statement runs the branch that matches the value

3 Iteration

Three loops

- FOR – a **count-controlled loop** 计数循环: the count is known up front
- WHILE – a **pre-condition loop** 前测循环: tests first, may run zero times
- REPEAT – a **post-condition loop** 后测循环: tests after, runs at least once



4 Procedures and functions

Procedures and functions

- **procedure** 过程 – does an action, returns nothing
- **function** 函数 – returns a value for an expression

Named **subroutines** 子程序 make **structured programming** 结构化编程.

procedure

call Greet("Ada")

does an action: prints "Hello, Ada"

returns **no** value

function

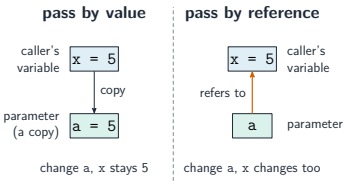
set y = Square(5)

computes a value: $5 \times 5 = 25$

returns 25

now y holds 25

Parameters and scope



value copies; reference shares

- **by value** 传值 – a copy; caller unchanged
- **pass by reference** 传引用 – can change the caller's variable
- prefer **local** 局部变量 over **global** 全局变量 (**scope** 作用域)

Where a name is visible

global scope (the whole program)

`Total` global — visible everywhere

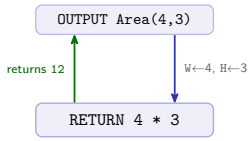
PROCEDURE Calc (local scope)

`temp` local — exists only while Calc runs

A **local variable** 局部变量 exists only inside its routine; a **global variable** 全局变量 is visible everywhere, which is exactly why it is easy to break by accident. The **signature** 签名 of a routine is its name plus its parameter list.

Worked example: a function that returns

```
FUNCTION Area(W : INTEGER,
    H : INTEGER) RETURNS INTEGER
  RETURN W * H
ENDFUNCTION
OUTPUT Area(4, 3)
```



prints **12**

A function call sits inside an expression because it returns a value; a procedure call is a statement on its own (`CALL`).

5 Efficiency

Writing efficient pseudocode

slower

```
FOR i = 1 TO n
  x = slow()  // every pass
  use(x, i)
NEXT i
```

faster

```
x = slow()  // once
FOR i = 1 TO n
  use(x, i)
NEXT i
```

- move **invariants** 不变量 out of loops
- exit a search early once found
- meaningful names; initialise before use
- comment the intent, not the mechanics

Choosing the loop, and writing it efficiently

count known in advance	`FOR` – the count is in the header, so it cannot be forgotten
may run zero times	`WHILE` – the test comes first
always at least one pass	`REPEAT...UNTIL` – the test comes last
how to justify	say whether the count is known and whether the body must run once. A scenario question wants both halves
avoid redundant work	store a result and reuse it rather than recomputing it inside the loop
and	leave the loop early once the answer is found, and do nothing inside the loop that does not change per pass

> Work that does not depend on the loop variable does not belong *inside* the loop. Moving it out is the commonest efficiency mark on the paper.

Work that does not belong in the loop

slower

```
FOR i = 1 TO n
  x = slow() // every
pass
  use(x, i)
NEXT i
```

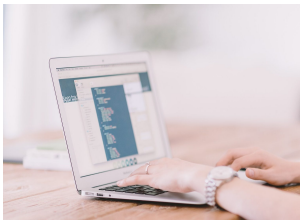
faster

```
x = slow() // once
FOR i = 1 TO n
  use(x, i)
NEXT i
```

Anything whose value cannot change should be computed once, before the loop.

> Anything whose answer does not change between passes belongs outside the loop. Compute it once, store it, and use the stored value.

Code and test together, not test at the end



> Testing as you write catches an error while you still remember what you meant. Leaving it to the end turns one bug into a hunt through everything.

Worked example – which loop suits each task?

(a) print the 12 times table; (b) keep reading numbers until the user enters 0; (c) ask for a password until it is correct.

- (a) The count is known in advance, so a **FOR** loop.
- (b) The count is unknown and the *first* input might already be 0, so the test must come before the body: **WHILE**, which runs zero or more times.
- (c) The count is unknown but you must ask at least once, so the test comes after: **REPEAT…UNTIL**, which runs one or more times.

> Two questions decide it: how many times does the body run, and when does the test happen?

Exam tips

- Distinguish a **procedure** (no return value) from a **function** (returns a value); know **pass by value vs by reference**.
- Choose the right loop: **count-controlled (FOR)** when the number of repeats is known, **condition-controlled (WHILE/REPEAT)** otherwise.
- Distinguish **local vs global** variables and scope; prefer local variables in reusable modules.

6 Recap

Topic 11 – key takeaways

1. **Selection**
   IF/ELSE; CASE for one value, many options
2. **Iteration**
   FOR / WHILE (zero) / REPEAT (at least one)
3. **Subroutines**
   procedure acts; function returns a value
4. **Parameters**
   by value copies; by reference shares

Check yourself

1. Which loop may run zero times?
2. Procedure or function: which returns a value?
3. Which parameter passing changes the caller's variable?
4. `7 MOD 2` = ?

Answers **1.** `WHILE` **2.** a function **3.** by reference **4.** 1