

Programming

A-Level Computer Science

Programming basics

```
1 usage
public Hotel (Integer capacity) { this.capacity = capacity; }
1 usage
public synchronized boolean checkIn(AccommodationRequest request) {
    if (guests.size() < capacity) {
        guests.add(request);
        return true;
    }
    else {
        return false;
    }
}
```

Programming turns a design into instructions written as code

Image: Ilyaaaaa, CC BY-SA 4.0 (commons.wikimedia.org)



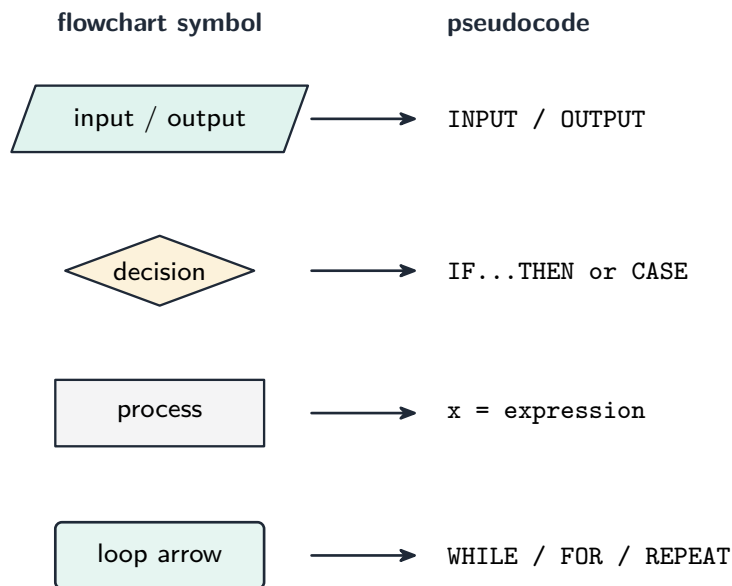
A programmer writes the code and tests it as they go

Image: Crew crew, CC0 (commons.wikimedia.org)

From design to code

You should be able to turn a design —a **flowchart** 流程图 (**program flowchart**) or **structured English** 结构化英语—into **pseudocode** 伪代码, and then into a real language:

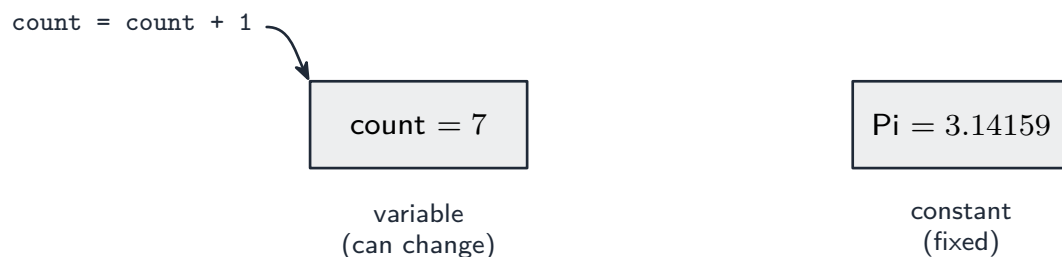
1. find the **variables** 变量 and their **data types** 数据类型.
2. turn input/output boxes into INPUT / OUTPUT.
3. turn decision diamonds into IF...ELSE...ENDIF (or CASE).
4. turn loop arrows into WHILE, REPEAT...UNTIL, or FOR.
5. turn process boxes into assignments or calculations.
6. check by tracing a small input.



Each flowchart symbol becomes a pseudocode keyword

Constants and variables

A **constant** 常量 holds a value that **never changes**; a variable holds one that may change. Declare them with a type:



A variable's value can change; a constant stays fixed

```

CONSTANT Pi ← 3.14159
DECLARE Radius : REAL
DECLARE Area : REAL

```

```
Radius ← 5
Area ← Pi * Radius * Radius
```

Use constants for fixed values that recur (Pi, MaxScore); they make code clearer and easy to change in one place.

Assignment and expressions

Use ← for **assignment** 赋值:

```
Total ← Total + 1
Average ← Sum / Count
```

Expressions use **operators** 运算符:

- arithmetic + - * / , plus DIV (integer division) and MOD (remainder): 7 DIV 2 = 3; 7 MOD 2 = 1.
- comparisons =, <>, <, >, <=, >=.
- logic AND, OR, NOT.

Precedence 优先级 (highest to lowest): NOT → * / DIV MOD → + - → comparisons → AND → OR. Use brackets when unsure.

Input and output

```
OUTPUT "Enter your name:"
INPUT Name
OUTPUT "Hello, ", Name
```

Built-in functions and library routines

Many tasks have ready-made **library routines** 库例程, so you need not write them:

- string: LENGTH(s), LEFT(s, n), RIGHT(s, n), MID(s, start, len), UCASE(s), LCASE(s).
- numeric: INT(x), ROUND(x), ABS(x), MOD(a, b), RANDOM().
- conversion: STR(x) (number → string), VAL(s) (string → number).

Use the exact names from the question paper's reference list.

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

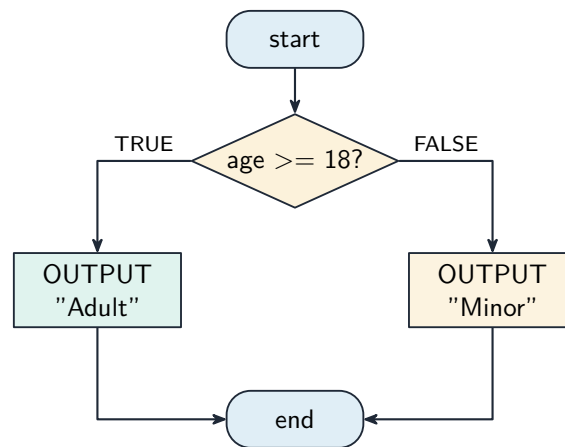
LENGTH(s) = 8	RIGHT(s, 2) = "ER"
LEFT(s, 3) = "COM"	UCASE(s) = "COMPUTER"
MID(s, 4, 3) = "PUT"	LCASE(s) = "computer"

The common string routines acting on s = "COMPUTER" (positions 1–8)

Selection

Selection 选择 chooses which steps run.

```
IF age >= 18 THEN
    OUTPUT "Adult"
ELSE
    OUTPUT "Minor"
ENDIF
```

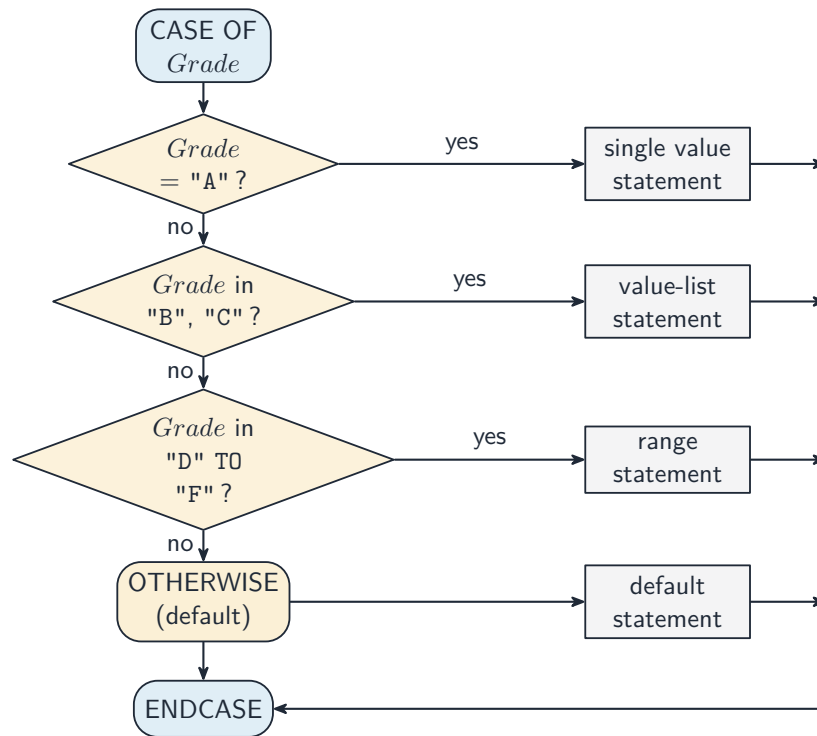


An IF...ELSE tests the condition once, then runs exactly one branch

For more than two cases you can use a **nested** 嵌套 IF, but deep nesting is hard to read —a **CASE** is cleaner when testing one value against several options:

```
CASE OF Grade
    "A": OUTPUT "Excellent"
    "B": OUTPUT "Good"
    OTHERWISE: OUTPUT "Try again"
ENDCASE
```

Cambridge CASE allows single values, value lists (1, 2, 3:), and ranges (1 TO 5:).



A CASE statement runs the branch that matches the value

Iteration

Iteration 迭代 repeats a block. Three loops differ in how many times the body runs.

Count-controlled (FOR) loop

A **count-controlled loop** 计数循环—use it when you know how many times to repeat:

```
FOR i ← 1 TO 10
  OUTPUT i
NEXT i
```

A STEP can change the count (e.g. FOR i ← 10 TO 1 STEP -1). Best for a fixed number of repeats or processing each element of an **array** 数组.

Pre-condition (WHILE) loop

A **pre-condition loop** 前测循环 tests the condition **before** each pass, so it may run **zero** times:

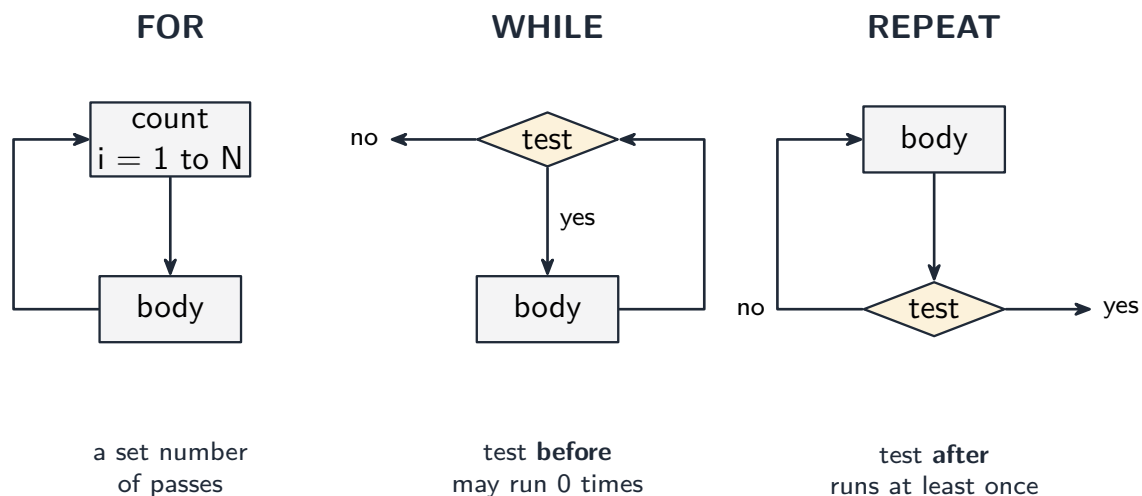
```
WHILE total < 100 DO
  INPUT n
  total ← total + n
ENDWHILE
```

Post-condition (REPEAT...UNTIL) loop

A **post-condition loop** 后测循环 tests the condition **after** each pass, so it always runs at least once:

```
REPEAT
  INPUT password
UNTIL password = correctPassword
```

Choosing the right loop



The three loops differ in where the condition is tested —before the body (WHILE), after it (REPEAT), or a set number of times (FOR)

- count known up front → **FOR**.
- may need zero passes → **WHILE**.
- always at least one pass → **REPEAT...UNTIL**.

Justify your choice by whether the count is known and whether the body must run at least once. A typical question gives a scenario ("ask for a password until correct, but always ask at least once") and asks which loop fits.

Procedures and functions

Structured programming 结构化编程 builds a program from small named **subroutines** 子程序, each with one job.

Procedure

A **procedure** 过程 is a named block that **does an action**; it may take **parameters** 参数 but does **not return a value**.

```

PROCEDURE Greet(name : STRING)
    OUTPUT "Hello, ", name
ENDPROCEDURE

CALL Greet("Ada")

```

Function

A **function** 函数 is like a procedure but it **returns a value** that becomes part of an expression.

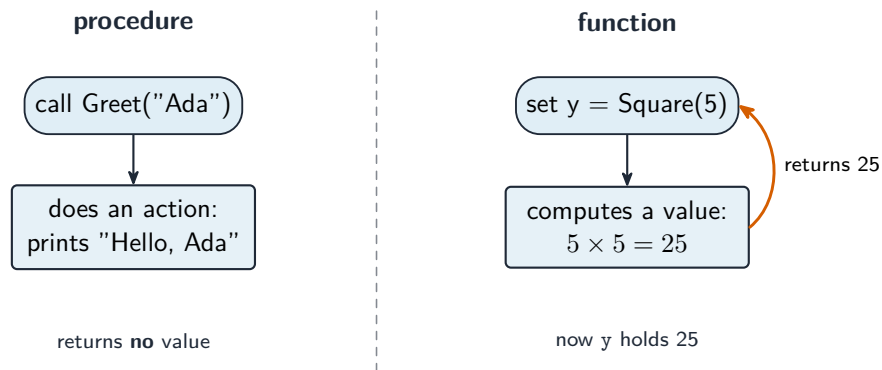
```

FUNCTION Square(x : INTEGER) RETURNS INTEGER
    RETURN x * x
ENDFUNCTION

result ← Square(5) + 1    // result = 26

```

Use a **procedure** when the subroutine performs an action; use a **function** when it computes a value for the caller.

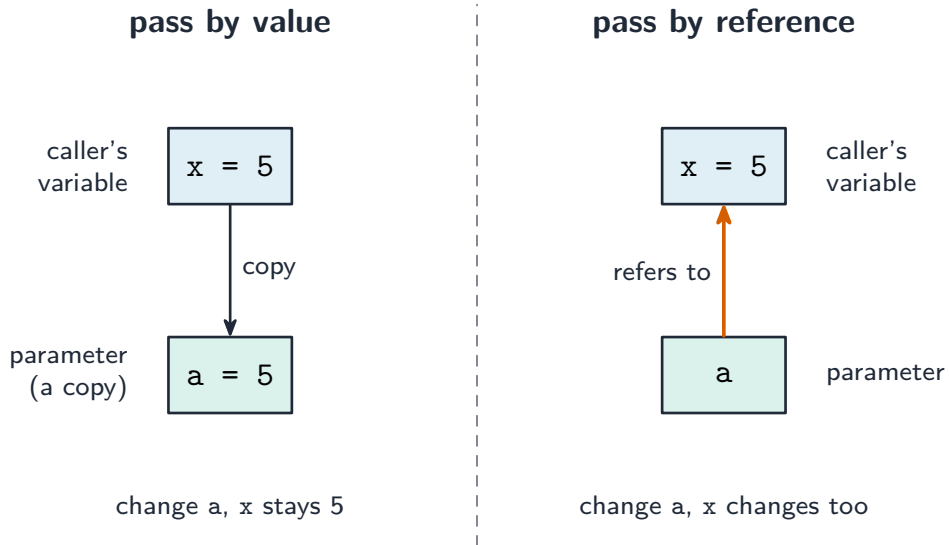


A procedure does an action and returns nothing; a function returns a value you use in an expression

Parameters

A **parameter** is a variable a subroutine declares to receive input; the values the caller supplies are **arguments** 实参. Two ways to pass them:

- **pass by value** 传值—the routine gets a **copy**; changes inside it do not affect the caller. Use for inputs it only reads.
- **pass by reference** 传引用—the routine gets a **reference** to the caller's variable; changes **do** affect the caller. Use when it must update a parameter.

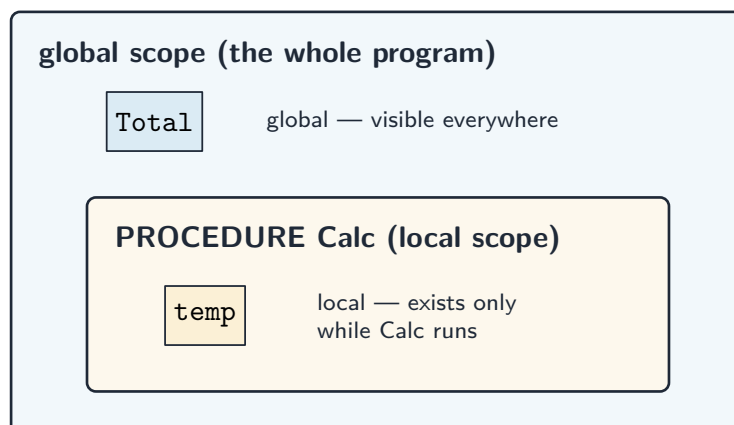


Pass by value copies the value into a new box; pass by reference lets the routine change the caller's own variable

```
PROCEDURE Swap(BYREF a : INTEGER, BYREF b : INTEGER)
  DECLARE temp : INTEGER
  temp ← a
  a ← b
  b ← temp
ENDPROCEDURE
```

Local vs global variables

A **local variable** 局部变量 is declared inside a subroutine and exists only while it runs. A **global variable** 全局变量 is declared outside and is visible everywhere. Prefer locals and parameters —heavy use of globals makes code hard to follow and test. (The region where a name is visible is its **scope** 作用域.)



A global variable is visible everywhere; a local variable exists only inside its own procedure

When to use a subroutine

Use a subroutine when:

- 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**.

Don't make them so tiny that the call costs more than the work inside.

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`).
- **procedure/function interface / signature** 签名—name + parameters + return type: what a caller must know to use it.

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. Choose by asking **how many times** the body runs and **when the test happens**. (a) The count is known in advance (12), so use a **FOR** loop. (b) The count is unknown, and the very first input might already be 0 - so the test must come **before** the body: a **WHILE** loop, which runs **zero** or more times. (c) The count is unknown, but you must always ask **at least once** before there is anything to test - so the test comes **after** the body: a **REPEAT...UNTIL**, which runs **one** or more times. The deciding question is whether the body must run at least once: **WHILE** may run zero times, **REPEAT** always runs once.

Writing efficient pseudocode

- **move invariants out of loops** —if a value (an **invariant** 不变量) does not change with the loop counter, compute it once before the loop.
- **exit a loop early** when the answer is found (stop a **linear search** 线性查找 as soon as the target appears).
- **avoid redundant work** —store a result and reuse it instead of recomputing.
- **choose the right data structure** —an array beats many separate variables when the items belong together.
- **replace deep nested IFs with CASE** when testing one value against many.
- **comment the intent**, not the mechanics (`// validate the postcode`, not `// loop 6 times`).
- **use meaningful names** (`numberOfPupils`, not `n`) and **initialise variables** before use.

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 unchanging work out of the loop so it runs once

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.