

Selection & Iteration

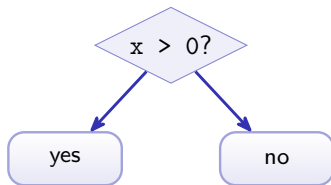
AP Computer Science A ■ Unit 2

AP Computer Science A



What we will cover

- 1 Boolean expressions
- 2 The if statement
- 3 Compound & nested conditions
- 4 while & for loops
- 5 Nested iteration & run-time





Boolean expressions & the if statement

A **Boolean expression** evaluates to true or false using **relational operators** ($>$, $<$, $==$, $!=$).

if (cond)
run a block

if / else
pick one of two

else if
three or more paths

This ability to choose a path is **selection**. Trace which **branch** runs for a given input.



Assembly loop

The three families of operators

arithmetic

+ **-** ***** **/**

MOD **DIV**

give a
number

relational

= **<>** **<** **>**

<= **>=**

compare →
true or **false**

logical

AND **OR**

NOT

join conditions →
true or **false**

The three families of operators: arithmetic, relational, and logical

Compound & nested conditions

`&&` (AND)
both true

`||` (OR)
at least one

`!` (NOT)
reverse

Short-circuit evaluation stops early: a `&& b` skips `b` if `a` is false. Compare **objects** with `.equals()`, not `==` (which compares references).



Traffic light if

Nested and chained conditions: only the first match runs

nesting

an if inside another – the inner one is reached only when the outer condition was true

chaining

else if tests several cases **in order**

the rule

only the first matching branch runs; the rest are skipped even if they also match

so order matters

put the **most specific** test first – score ≥ 50 before score ≥ 80 gives everyone the lower grade

else

catches everything no earlier branch matched – it needs no condition of its own

Tracing a chain means reading **top to bottom and stopping at the first true**. That is the whole semantics, and most multiple-choice items test only this.



while & for loops

while loop

repeat while a condition
holds – count of repeats
not known in advance

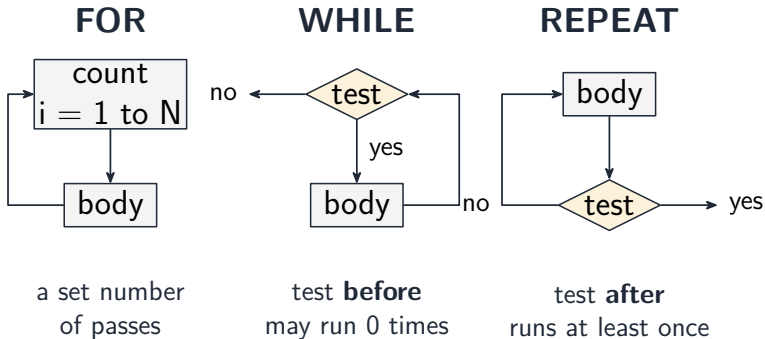
for loop

```
for (int i=0; i<n; i++)
```

– a known number of times

Every **loop** needs progress toward its stop condition, or it becomes an **infinite loop**. Trace variables pass by pass.

while & for loops, in detail



The three loop types differ in where the condition is tested

Every for loop is a while loop

These two do identical work. Be able to convert either way.

```
// for: init, condition, update
for (int i = 0; i < n; i++) {
    sum += i;
}
```

```
// the same, written as a while
int i = 0;
while (i < n) {
    sum += i;
    i++;
}
```

Use for when you **know the count** in advance, while when you do not. Notice where the three parts of the for header land in the while version – and that omitting `i++` is exactly what makes an **infinite loop**.

Nested iteration & run-time

```
for (int i=0; i<n; i++)  
    for (int j=0; j<n; j++)  
        // runs n x n times
```

A **nested loop** puts one loop inside another. **Informal run-time analysis** : count how many times the inner statement runs – here $n \times n = n^2$, so it grows fast.

The integer patterns the exam tests directly

reading digits

repeatedly take $n \% 10$ for the **last digit**, then $n /= 10$ to drop it
– loop while $n > 0$

why it works

$\%$ gives the remainder and integer $/$ truncates, so together they peel one digit per pass

a running total

declare the accumulator **outside** the loop, add inside it

a counter

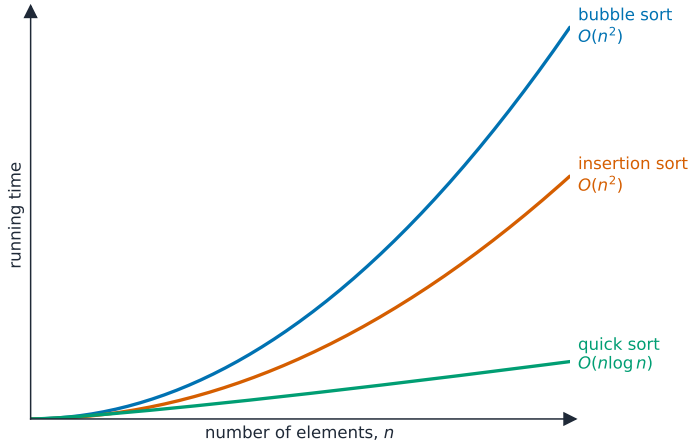
the same shape, but incrementing on a condition rather than adding a value

a flag

a boolean recording **whether something happened** – set inside the loop, tested after it

Total, count, max/min and flag cover most free-response loop questions. Decide which the wording is asking for before writing anything.

Nested iteration & run-time, in detail



How the running time grows with the number of elements n

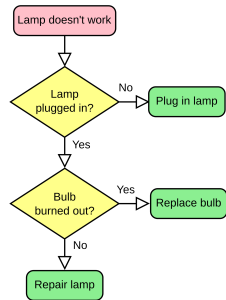
String algorithms

Loop over a String's indices to process each character.

Traverse a String

```
for (int i = 0; i < s.length();  
i++)  
char c = s.charAt(i);
```

Count vowels, reverse text, or find a substring – all by visiting each index in turn.



A flowchart with a decision diamond: selection chooses which path the algorithm takes

The standard String tasks

count occurrences

loop the indices, compare `s.substring(i, i+1)` to the target, increment a counter

build a copy

start from an empty String and concatenate the characters you want, in the order you want them

reverse

the same build, walking the index **downward**

filter

the same build, concatenating only characters that pass a test

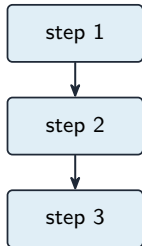
contains

`s.indexOf(t) >= 0` – remembering that `-1` means absent

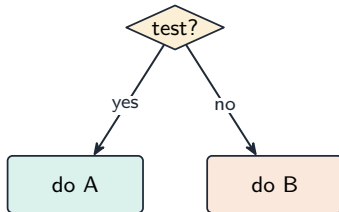
All five are one loop with a different body. Because a String is **immutable**, each builds a *new* String rather than editing the old one.

The three control structures

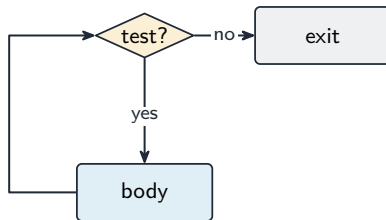
Sequence



Selection



Iteration



The three control structures: sequence, selection, and iteration

Every algorithm is built from three structures

sequence

steps executed **in order**, one after another

selection

choosing a path – if, if/else, else if

iteration

repeating – while and for

logical operators

&&, ||, ! combine conditions; the first two short-circuit, so the right side may never be evaluated

the claim

any algorithm can be written with just these three – a theorem, not a style rule

Name the structure before you write code. Most “I don’t know where to start” moments are really “I have not decided whether this repeats”.

Key points

if / else if / else
choose a **branch** (selection)

&&, ||, ! build compound conditions

while & for loops **iterate**; avoid infinite loops

A **nested loop** runs n^2 times – growth matters

Exam tips

- Get boundary conditions right: use `<` vs `<=` deliberately, and watch the first and last iteration of every loop (off-by-one is the classic bug).
- Build compound conditions with `&&`, `||`, `!` and remember **short-circuit** evaluation (put the null check first).
- Trace nested loops by counting how many times the **inner** body runs in total.
- Choose the right structure – `if/else if` for ranges, a loop for repetition – and avoid an infinite loop by updating the loop variable.
- Apply **De Morgan's laws** when you simplify or negate a boolean condition.



Unit 2 – key takeaways

1

Bounds

$<$ vs $\leq$; first and last iteration

2

Short-circuit

null check first in $\&\&$

3

Nested

count how many times the inner body runs

4

De Morgan

use it to simplify or negate

Check yourself

- 1 A loop runs n times inside a loop that runs n times. How many iterations?
- 2 `while` or `for`: which suits a known count?
- 3 What does `&&` do when the left side is false?
- 4 Name the two structures that control program flow.

Answers 1. n^2 2. `for` 3. short-circuits – the right side never runs 4. selection and iteration