

Topic 11: Programming

Why $0.1 + 0.2$ is not 0.3 , how integers overflow, what the call stack really holds and how conditions are short-circuited.

Optional stretch material. This goes beyond the 9618 syllabus, so you will not be asked about it directly in the exam. Its purpose is to deepen ideas you already know. **Builds on:** 11.1 Programming basics, 11.2 Constructs, 11.3 Structured programming. Try the challenge questions on the last pages before you read the answers.

Floating-point traps

Computers store real numbers in binary. Just as $1/3$ cannot be written exactly in decimal ($0.3333\dots$), values like **0.1** cannot be written exactly in binary. Each is stored as a very close approximation:

```
>>> 0.1 + 0.2
0.30000000000000004
>>> 0.1 + 0.2 == 0.3
False
```

Rules of thumb: never test real numbers for exact equality; compare with a tolerance, e.g. IF $ABS(A - B) < 0.00001$ THEN. Store money as whole pence (integers) or a decimal type, not as floats.

Integer overflow

Integers in most languages have a fixed number of bits, so arithmetic wraps around. In 8-bit **unsigned** arithmetic $255 + 1 = 0$. In 8-bit **two's complement** $127 + 1 = -128$. Real incidents: the Ariane 5 rocket (1996) was lost when a 64-bit floating-point value was converted into a 16-bit integer that could not hold it. Python's integers grow without limit, but most languages do not.

What the call stack holds

Each time a procedure or function is called, a **stack frame** is pushed holding its parameters, local variables and the **return address**. When it returns the frame is popped. This is how a program remembers where to go back to.

```
FUNCTION Fact(N) RETURNS INTEGER
  IF N <= 1 THEN RETURN 1
  RETURN N * Fact(N - 1)
ENDFUNCTION
```

```
Stack while evaluating Fact(3):  Fact(1)  <- top
                                Fact(2)
                                Fact(3)
```

The stack has a limited size. A recursive call with no base case keeps pushing frames until it overflows: a **stack overflow**.

Short-circuit evaluation and functions as values

In languages such as Python, Java and C, $A \text{ AND } B$ does not evaluate B at all if A is already false. This is used deliberately: IF $(i \leq N) \text{ AND } (Data[i] = Target)$ is safe because $Data[i]$ is never read once i exceeds N . (Always check how your exam language behaves; do not rely on it in pseudocode answers.)

In many languages a function can be passed to another function. Python's `sort` accepts a rule for ordering: `students.sort(key=lambda s: s.grade)`, a first taste of the functional style in topic 20 extension.

Challenge questions

1. In 8-bit two's complement, what is the result of $100 + 50$, and why?
2. A program contains `Total <- 0.1 + 0.2` followed by `IF Total = 0.3 THEN ...`. Why is this risky? Show a safer version.
3. A recursive function `Sum(N)` returns 0 when $N = 0$ and $N + \text{Sum}(N - 1)$ otherwise. How many stack frames exist at the deepest point when `Sum(3)` is called?

Worked answers

Only look once you have had a proper go at each question.

1. 150 is larger than 127 (the biggest 8-bit value), so it overflows. $150 = 10010110$ in binary, which read as two's complement is $150 - 256 = -106$.
2. 0.1, 0.2 and 0.3 are not stored exactly; the sum is 0.30000000000000004, so the test fails. Safer: `IF ABS(Total - 0.3) < 0.000001 THEN.`
3. Frames for Sum(3), Sum(2), Sum(1) and Sum(0): **4** frames. Sum(0) returns first, then each frame is popped as the results unwind (0, 1, 3, 6).