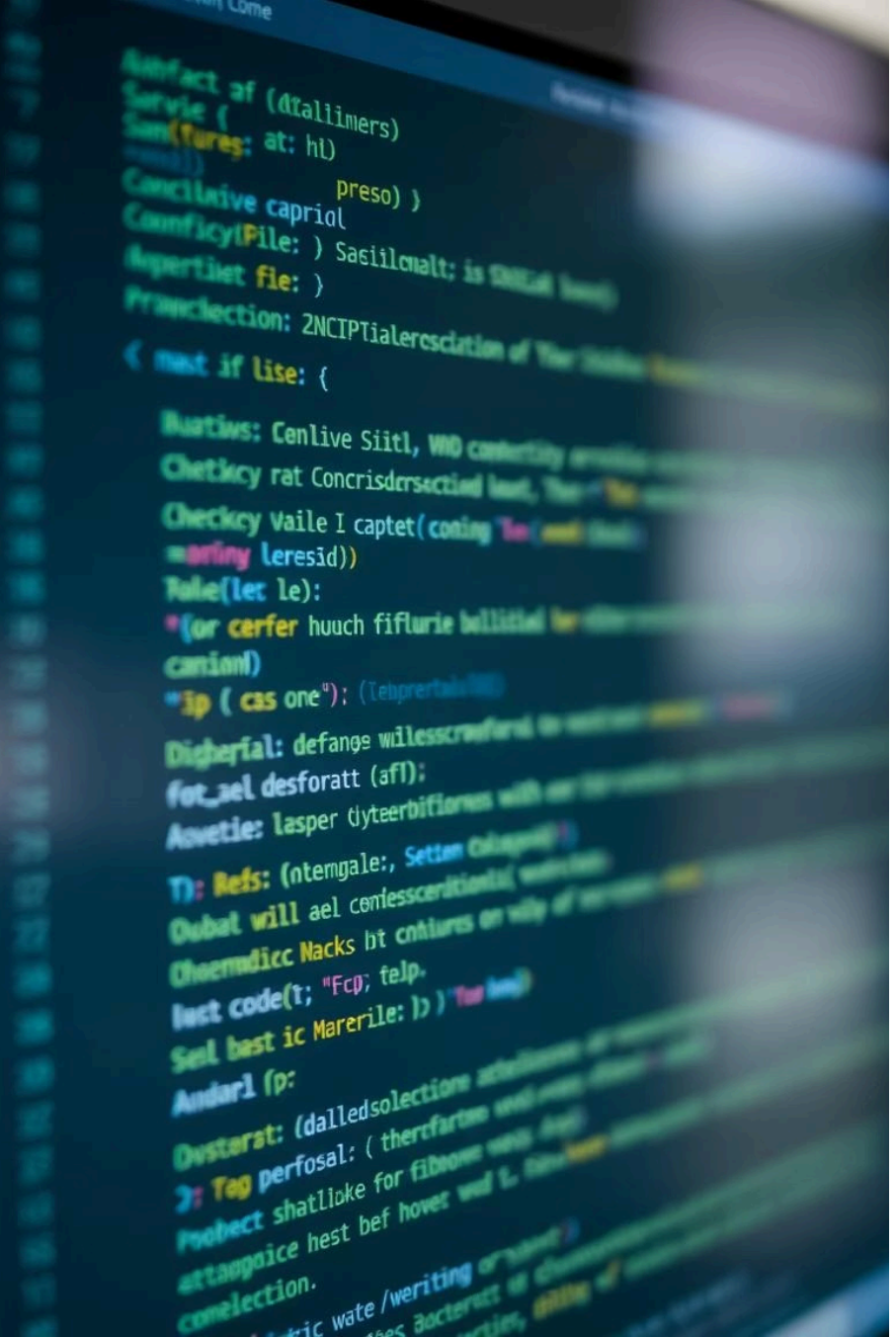


C Language: Typedef, Bit Fields, Enums, and Unions

by Hamdi Emad

```c
#include <stdio.h>

typedef int INTEGER;

typedef struct employee {
    INTEGER id;
    char name[50];
    float salary;
} manager;
```

Typedef: Defining Custom Data Types

Purpose

Creates aliases for existing data types to improve readability.

Benefits

Enhances code maintainability and clarity.

Example

typedef int student_id; defines a new alias for int.

Bit Fields: Memory-Efficient Data Storage

Declaration

Use syntax like **unsigned int flag : 1;** inside structs.

Size Limits

Bit field sizes are limited by the underlying type size.

Advantage

Reduces memory footprint by compact packing of bits.

```c
#include <stdio.h>

struct test {
    unsigned a : 1;
    unsigned b : 2;
    unsigned c : 3;
    unsigned : 0;
};
```

Bit Field Overflow Behavior

1

Implementation Defined

Behavior varies across compilers and platforms.

2

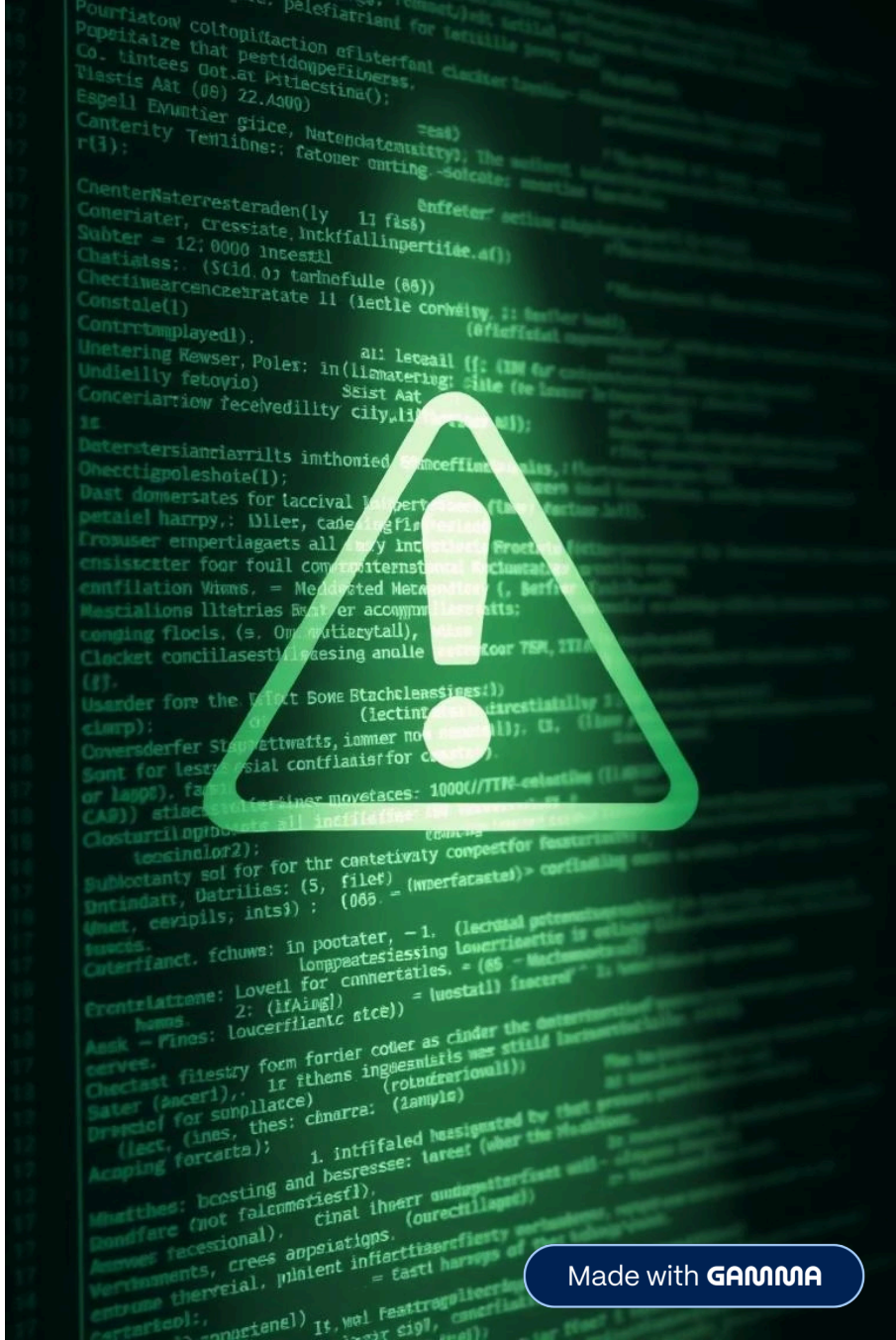
Potential Issues

Overflow may cause data truncation (unsigned) or unexpected results(signed).

3

Design Care

Careful bit field sizing is essential to avoid errors.



Typedef with Structs and Unions

Simplifies Declarations

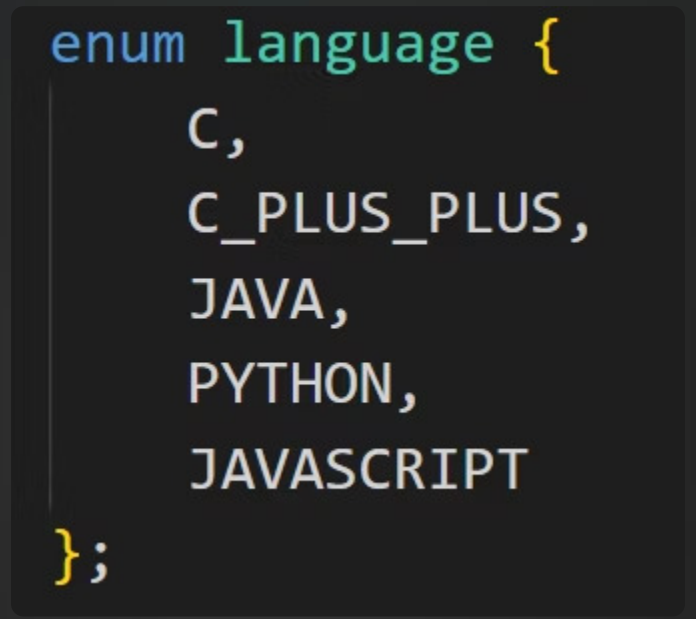
Makes complex

types easier to name and use in code.

Example

typedef struct { int x, y; } point_t;

Reduces verbosity while enhancing clarity.



Enums: Enumerated Types

Default Type

Enums default underlying type is **int**.

Explicit Typing

Can specify types like **char** for storage efficiency.

Purpose

Represent named sets of integral constants clearly.

Unions vs. Structs: Key Differences

Structs

- Members stored sequentially in separate memory areas
- Total size equals sum of all members

Unions

- All members share the same memory location
- Size is that of the largest member only

Memory Efficiency with Unions

Single Active Member

Only stores one member at any time, saving space.

Use Case

Ideal to represent multiple data formats in one variable.

Benefit

Significantly reduces memory consumption versus multiple vars.

Code Example: Using a Union

```
typedef union {
  int i;
  float f;
} Number;

Number num;
num.i = 42;
// Only num.i or num.f is valid at a time
```

This code defines a union storing either int or float. Only one member is active at once, optimizing memory.