What is the purpose of typedef?

The typedef keyword in C is used to create a new name (alias) for an existing data type. It helps improve code readability and makes complex types like structs, unions, and pointers easier to manage.

example:

```
typedef unsigned int uint;
```

How are bit fields declared and what are their size limitations?

Bit fields are declared within a struct using the syntax:

```
struct example {
    unsigned int a : 3;
};
```

This means a uses only 3 bits of memory. The size of a bit field cannot exceed the size of its underlying type (usually int), and the exact behavior might depend on the compiler and system architecture.

What happens if a bit field overflows?

If a value assigned to a bit field exceeds its allowed size, the higher bits are truncated, meaning only the least significant bits are stored. This can cause incorrect or unexpected values if not handled carefully.

How is typedef used with complex types like structs and unions?

typedef simplifies the syntax for complex types.

example:

```
typedef struct {
    int x;
    int y;
} Point;
```

Similarly, for unions:

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

What is the default underlying type of an enum?

By default, the underlying type of an enum in C is int. This means each enumerator is stored as an int value unless explicitly specified (which is allowed in some C standards like C99 and later).

How is a union different from a struct?

In a struct, each member has its own separate memory location, so the total size is the sum of all members. In a union, all members share the same memory space, and the size of the union is equal to the size of its largest member.

When is using a union more memory-efficient?

A union is more memory-efficient when you need to store different data types but only one at a time. Since all members share the same memory space, a union reduces memory usage compared to a struct, which allocates memory for all members.