Summary

What is a macro in C, and how is it defined?

A macro in C is a fragment of code that is given a name. It is defined using the #define directive and is replaced by the preprocessor before compilation. For example: #define PI 3.14

What is the difference between macros and functions?

Macros are processed by the preprocessor and involve textual substitution, while functions are compiled code that is executed at runtime. Macros can be faster but are harder to debug and error-prone compared to functions.

What do #ifdef, #ifndef, and #endif do?

#ifdef checks if a macro is defined, #ifndef checks if it is not defined, and #endif marks the end of the conditional directive. They are used for conditional compilation.

What does malloc() do, and what type does it return?

malloc() allocates a specified number of bytes in memory and returns a void pointer (void*) to the allocated space.

What is the difference between malloc, calloc, and realloc?

malloc allocates uninitialized memory, calloc allocates zero-initialized memory for an array, and realloc resizes previously allocated memory.

Why must we always call free() after dynamic allocation?

Calling free() is necessary to release dynamically allocated memory back to the system and prevent memory leaks.

What is a header guard, and what problem does it solve?

A header guard prevents a header file from being included multiple times, which avoids duplicate definitions and compiler errors.

What is the typical format of a header guard?

A typical header guard format is:

```
#ifndef HEADER_NAME_H
#define HEADER_NAME_H
... // content
#endif
```

How does the preprocessor handle nested includes?

The preprocessor expands nested includes recursively and uses header guards to prevent multiple inclusions of the same file.