Object Oriented Design and C

David Sackstein

Agenda

- C and C++ interoperability
- A gradual migration to C++
- Calling C code from C++
- Calling C++ code from C
- Defining an object in C
- Interfaces in C

Interop

- C++ is a superset of C
- The two languages can be used in the same project (executable, shared library).
- This makes it easy to migrate a C project to C++ in small steps.

Calling C code from C++

- In C++ function names are not what they seem.
- They are mangled to include information about the arguments and their types.
- You can prevent the compiler from mangling names using extern “C” {}
- To call the functions in c.h from c++ you need to wrap the inclusion of c.h with extern “C” {}

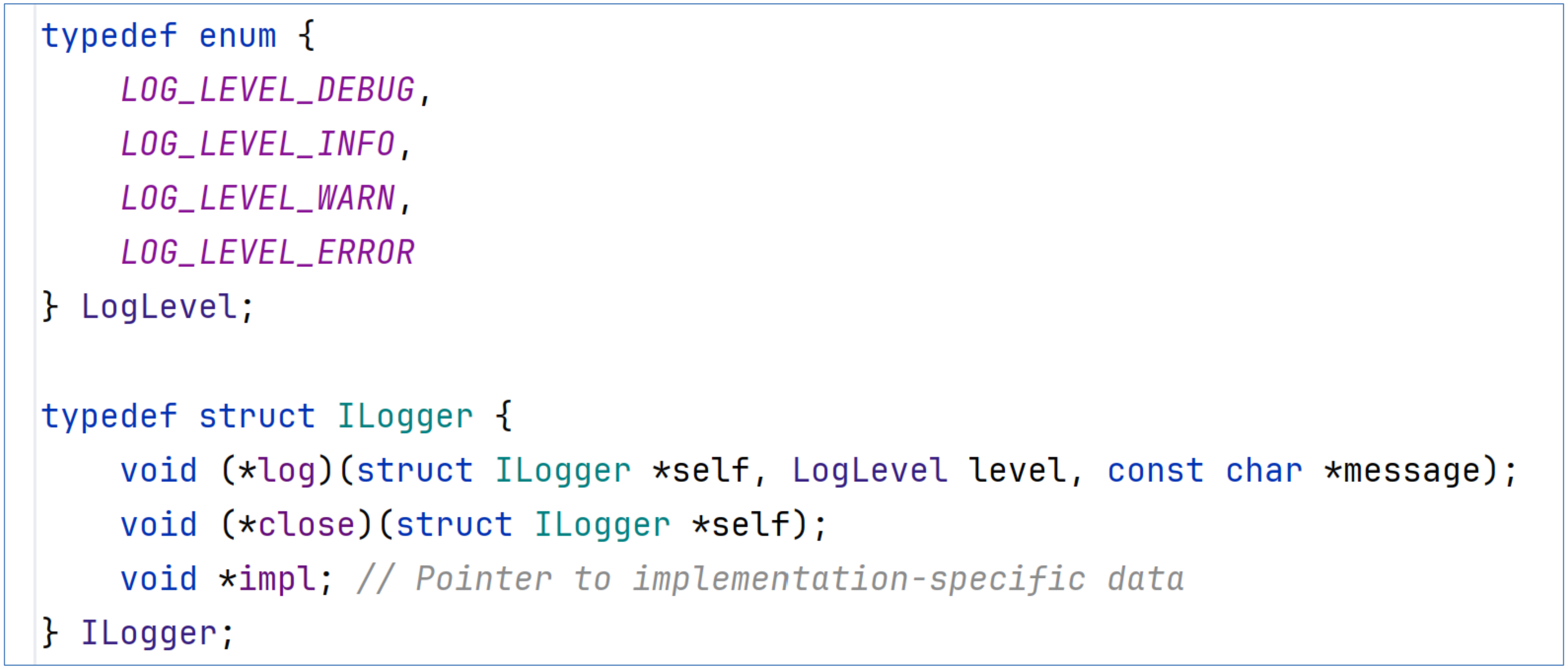
Calling C++ code from C

- For the same reason, you can only call a C++ method from C if it is not name-mangled (or you know it mangled name).
- For C++ methods wrap the declaration in the header and the implementation in extern "C"
- As extern "C" is invalid syntax in C, wrap its use with ifdef __cplusplus

Defining an object in C

- In a header define a struct.
- In a c file with the same name, define methods that accept a pointer to the struct as their first parameter (you can call it ‘this’).
- Define a constructor and a destructor
- In the header declare the methods you want public.
- Do not declare the methods you want private
- In the source file mark the methods you want private as static.
- You can define an interface as a struct containing pointers to functions.

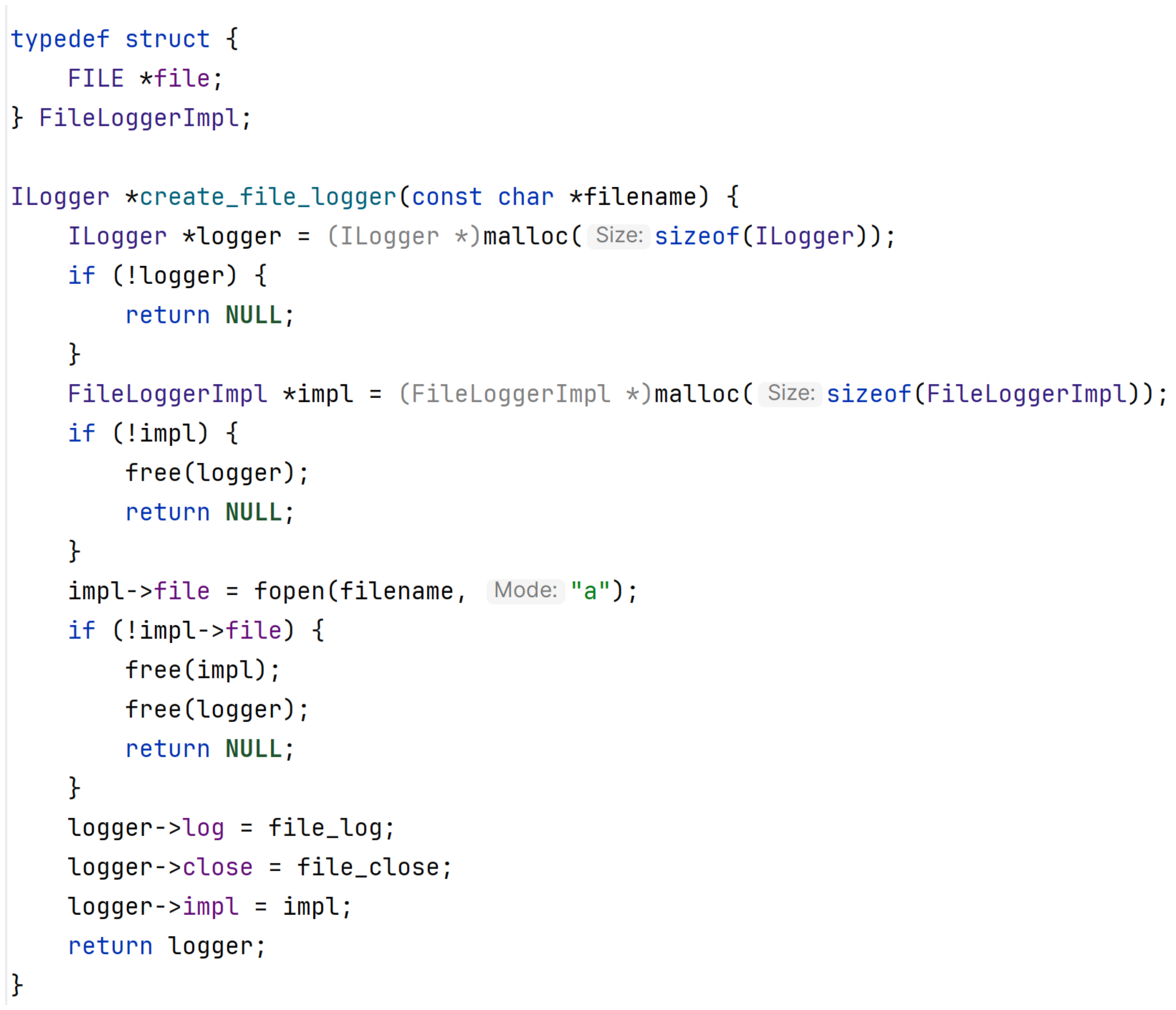
The ILogger interface



```c
typedef enum {
    LOG_LEVEL_DEBUG,
    LOG_LEVEL_INFO,
    LOG_LEVEL_WARN,
    LOG_LEVEL_ERROR
} LogLevel;

typedef struct ILogger {
    void (*log)(struct ILogger *self, LogLevel level, const char *message);
    void (*close)(struct ILogger *self);
    void *impl; // Pointer to implementation-specific data
} ILogger;
```

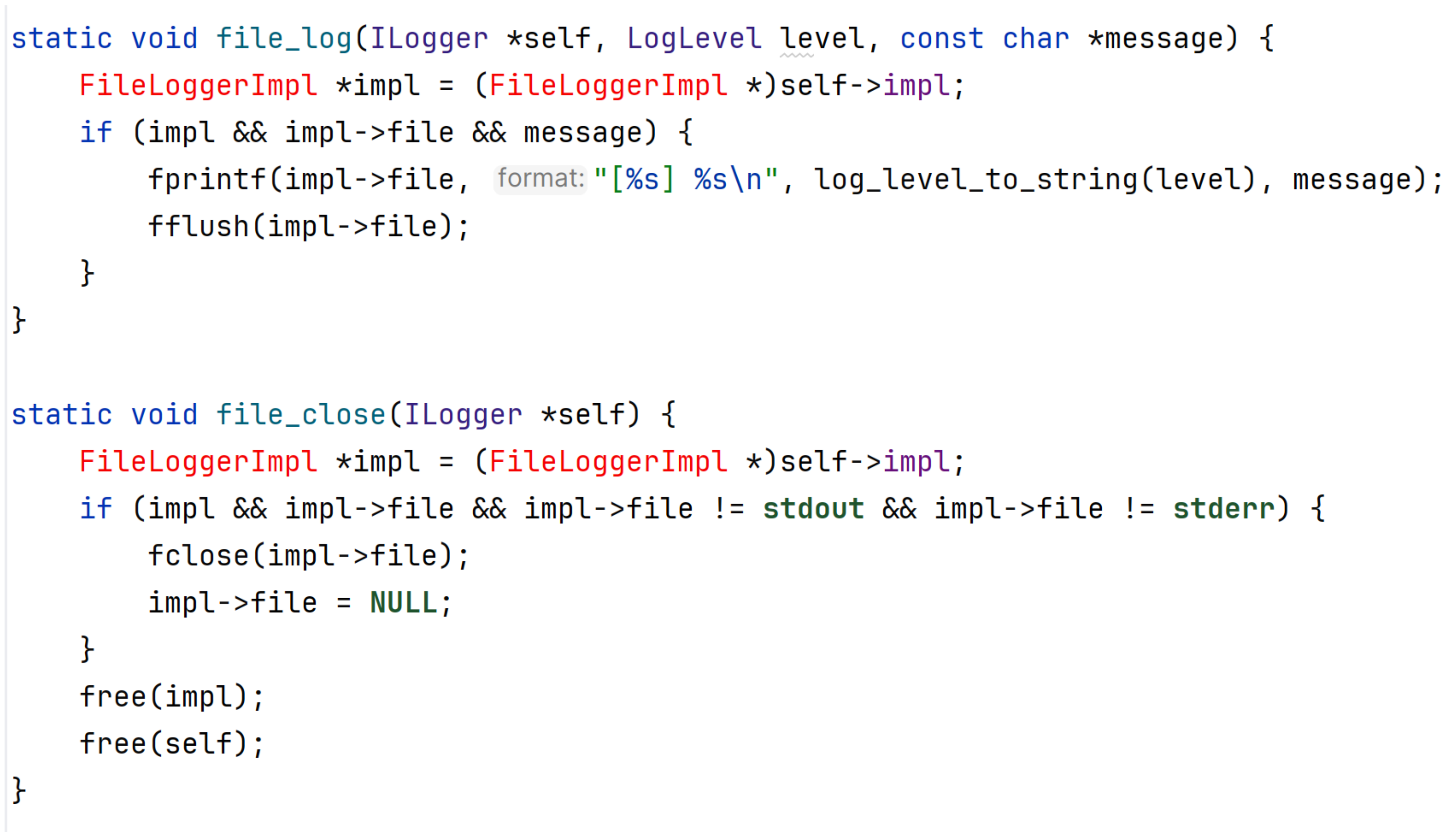
The FileLogger implementation



```c
typedef struct {
    FILE *file;
} FileLoggerImpl;

ILogger *create_file_logger(const char *filename) {
    ILogger *logger = (ILogger *)malloc( Size: sizeof(ILogger));
    if (!logger) {
        return NULL;
    }
    FileLoggerImpl *impl = (FileLoggerImpl *)malloc( Size: sizeof(FileLoggerImpl));
    if (!impl) {
        free(logger);
        return NULL;
    }
    impl->file = fopen(filename,  Mode: "a");
    if (!impl->file) {
        free(impl);
        free(logger);
        return NULL;
    }
    logger->log = file_log;
    logger->close = file_close;
    logger->impl = impl;
    return logger;
}
```

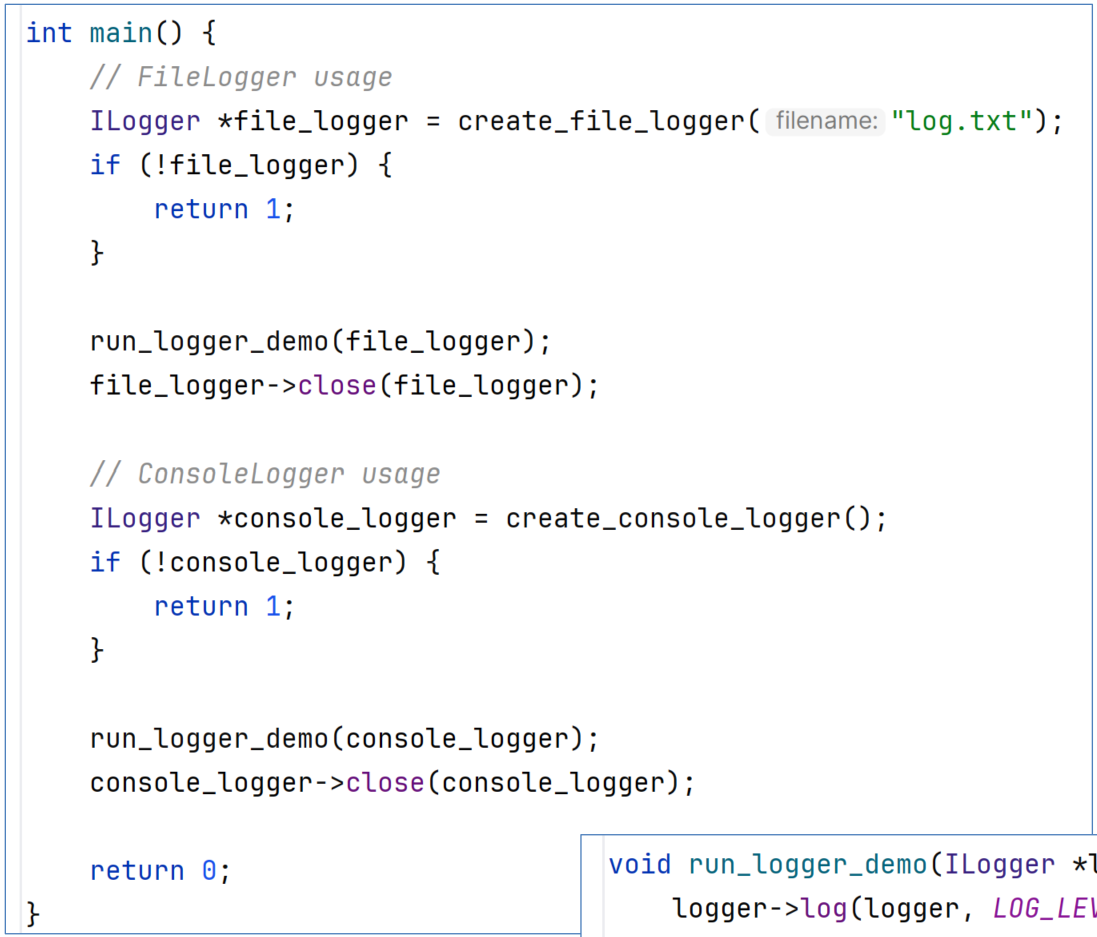
The FileLogger implementation



```c
static void file_log(ILogger *self, LogLevel level, const char *message) {
    FileLoggerImpl *impl = (FileLoggerImpl *)self->impl;
    if (impl && impl->file && message) {
        fprintf(impl->file,  format: "[%s] %s\n", log_level_to_string(level), message);
        fflush(impl->file);
    }
}

static void file_close(ILogger *self) {
    FileLoggerImpl *impl = (FileLoggerImpl *)self->impl;
    if (impl && impl->file && impl->file != stdout && impl->file != stderr) {
        fclose(impl->file);
        impl->file = NULL;
    }
    free(impl);
    free(self);
}
```

Use like in C++



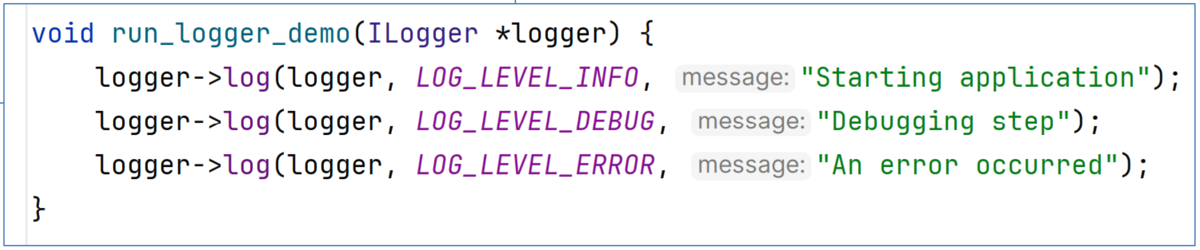
```c
int main() {
    // FileLogger usage
    ILogger *file_logger = create_file_logger( filename: "log.txt");
    if (!file_logger) {
        return 1;
    }

    run_logger_demo(file_logger);
    file_logger->close(file_logger);

    // ConsoleLogger usage
    ILogger *console_logger = create_console_logger();
    if (!console_logger) {
        return 1;
    }

    run_logger_demo(console_logger);
    console_logger->close(console_logger);

    return 0;
}
```



```c
void run_logger_demo(ILogger *logger) {
    logger->log(logger, LOG_LEVEL_INFO,  message: "Starting application");
    logger->log(logger, LOG_LEVEL_DEBUG,  message: "Debugging step");
    logger->log(logger, LOG_LEVEL_ERROR,  message: "An error occurred");
}
```

Key Takeaways

- The principles of OOD are useful in C too.
- The problem is that you need to do a lot of work yourself, and also the compiler does not force you to write correctly.
- But you should write constructors, destructors and use the static keyword to hide symbols
- You can implement interfaces as structs of function pointers.
- Due to interop of C and C++ you can migrate incrementally (you can use shared libraries to help here).