Embedded Systems Programming Laboratory

Object Orientated C and Code Modularization

Alex Hoffman

Technical University of Munich
Chair of Real-Time Computer Systems (RCS)

November 22, 2022

Lehrstuhl für
Realzeit-Computersysteme

Example code

Code examples to accompany these slides can be found **here**

Object Orientated (OO) Programming

Programming paradigm:

- "Objects" contain data and methods
- Methods often modify data contained within object
- Objects interact with each other
- Objects can be passed between other objects or modules of a program
- Sets clear boundaries around data and interfaces

Example languages:

- Java
- C++
- Python
- JavaScript
- Objective-C (not to be confused with C utilizing object orientated concepts)

Possible problems with C

- No enforced rules for writing to objects through defined channels of thought and action
- No clear lines of independent behaviours between object/modules of a program
- Hidden interdependencies between subsystems can appear creating less modular and harder to debug code

Solution:

- Developers must use OO practises that they must enforce as their is no compiler protection
  - OO is not bound by the language or compiler, it is simply a way of thinking
- Establish rules as to how OO code will be written in your program
- Stick to these rules!

Example of OO in C - Private and Public Variables

In an OO language such as Java the data of an object can be hidden from the rest of the program, only accessible through defined set and get methods.

```
1  public class EncapTest {
       private int age;

       public int getAge() {
          return age;
6      }
       public void setAge( int newAge) {
          age = newAge;
       }
   }
```

The set and get methods are used to interact with the object's data. These methods can enforce correct usage of the object's data, eg. you cannot set the age to be a string.

Private and Public Variable Rules

Public (exposed to the rest of the program) decelerations of variables and functions are placed in a separate file to private variable and functions. This becomes the API to your object.

Thus,
RULE 1: Place public declarations of functions and variables in a separate file from those private to the module

RULE 2: Header files act as the API to the module or object stored in the respective C file

Header Files

A common weak area for students

- Are "copy pasted" everywhere the `#include` directive is used
- Thus used to share C function declarations, variable declarations and macro definitions between source files (C files).
- System header files (eg. `#include <stdio.h>`) and user header files (eg. `#include "main.h"`)
- Avoid blanket inclusions
  - Leads to circular inclusion
  - Include only what is necessary for that file
    If a C file needs an include do not put it in the header of that C file and include that file's header
  - Easier to debug build errors

Essentially be a minimal and small-scoped as possible with includes

Include Example

```c
/* screen.c */
#include "error.h"
#include "screen_API.h"
#include "config.h"

#ifdef FREERTOS
#include "cmsis_os.h"
#endif

#include <stdlib.h>
#include <string.h>

typedef struct screen_device {
  int rows;
  int cols;
...
```

Example from `https://github.com/alxhoff/STM32-Mechanical-Keyboard/blob/master/V2/Src/screen.c`

Include Example

```c
/* screen.h */
signed char screen_init(void);
char **screen_get_buffer(void);
int screen_get_cursor_x(void);
int screen_get_cursor_y(void);
signed char screen_add_line(char *line);
signed char screen_add_line_at_index(unsigned char
    index, char *line);
void screen_move_cursor_left(void);
void screen_move_cursor_right(void);
```

- Frame buffer cannot be directly accessed, pseudo private
- No unnecessary includes in header file

Example from `https://github.com/alxhoff/STM32-Mechanical-Keyboard/blob/master/V2/Inc/screen.h`

Private Variable Example in C

```c
/* hello_object.c */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

char *public_message = "Hello World";
char message[100] = {0}; // <-- Private
int message_length = 0; // <-- Private

void set_message( char *str ){...}
char *get_string_copy( char *str ){...} // <-- Private
char *get_message( void ){...}
int get_message_len( void ){...}
```

Private Variable Example in C

From lecture example code: `https://github.com/alxhoff/lecture_example_code/tree/master/objective_c/1`

```
/* hello_object.h */
extern char *public_message;

void set_message( char *str );
char *get_message( void );
int get_message_len( void );

```

Cannot set message length manually, hello_object API makes sure this value is internally computed and thus always valid. Also the `set_message` function makes sure that the provided string is not too long for the internal buffer thus avoiding memory issues.

What to "Expose" Through Header Files

- Only put declarations in headers that are required elsewhere in your code base. Students have the very bad habit of exposing ALL functions via headers.
- Functions that are not exposed are thus "private"
- Similarly only expose defines, macros and data structures/types via the header that are required elsewhere.

Private Structures/Data Types Example in C

- We can hide either entire instances of structures, similar to previous example
- We can just return a "generic" handle to a created object such that the user is forced to use API functions to interact with the object
  - The "generic" handle is simply a void pointer typecast that provides zero information to the underlying data structure
  - Actual memory structure of the object is hidden in the C file

Private Structures/Data Types Example in C

We can hide either entire instances of structures, similar to previous example. Example code is available **here**.

```
struct String
{
    int len;
    char *str;
} my_string;
```

This will create a structure called `my_string` that is accessible via the exposed functions

Private Structures/Data Types Example in C

For example, `setString`

```
int setString(char *str)
{
    char *new_str = strdup(str);

    if (new_str == NULL)
        return -1;

    my_string.str = new_str;
    my_string.len = strlen(new_str);

    return 0;
}
```

Private Structures/Data Types Example in C

Drawbacks to this approach, or advantages maybe, depending on context

- You are only dealing with a single data object
  - This might be all you need, eg. list of specific objects, eg. states in a state machine
- A reference to the object can not be passed around as one is unable to access it's location
- Cannot embed the object inside other objects

Private Structures/Data Types Example in C

- A more robust solution is to return usable objects to the user that have their functionality/data hidden and/or controlled, this is analogous with true OO programming
- But how?
  - Returning void pointers gives the user the where but not the what
  - An object's description is stored inside the C file such that the API functions and other private functions know how the data looks
  - void pointers are usually cast into a human readable handle that is then passed to the API functions
- This removes the possibility that the user can use the objects directly
- Allows for user to create multiple objects and even embed them into other structures
- My recommendation for most OO solutions in C

Private Structures/Data Types Example in C

More in-depth example available **here**.

- The handle is a typecasted void pointer

```
typedef void *String_t;
```

- Inside the C functions the object can be typecasted as the structure definition is inside the C file and therefore accessible

```
struct String *cast_string = (struct String *)
    string_handle;
```

- The casted handle is then usable as a normal struct
- The user can create multiple objects, to manipulate objects a handle should be passed into the API functions