



API Design and Concerns

CSCI 3410 Spring 2023

Prof. Sabin Mohan

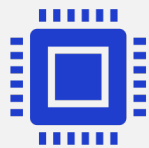
Application Programming Interface [API]



Method for computer programs to communicate with each other



software **interfaces**



offering **services** to other software/code

Note: **not** a user interface!

Main
Concerns
[for now]

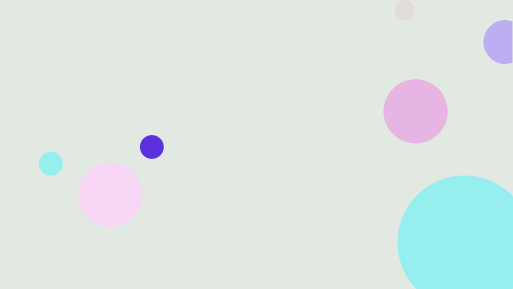
Return Values

Errors

Memory Ownership



Return Values

- Functions may need to return more than one value/result
 - actual return (results of computation, memory allocation, etc.)
 - how to interpret the return value?
 - errors (if any)
 - meaning of errors (if any)
 - additional information, say for debugging or understanding context
 - create multiple entities and return them
- 

Return Values in C

- explicit return [return by value]

```
int sum( int l, int r )...
```

- return using (incoming) pointer [return by reference]

```
strcpy( destination_pointer, source_pointer ) ;
```

- return using global variable/macro/environment variable [“implicit” return]

```
errno -1
```

- return multiple entities [use **structs**]

```
struct ret_type{  
    int a;  
    char* carray ;  
} ;
```

```
struct ret_type my_function(...)  
struct ret_val_ptr* my_other_func(...)
```

Errors!



How to Detect Errors

- functions that return integers —→ common for **negative values** to indicate errors
- functions that return pointers —→ **NULL** value indicate errors
- custom struct can include error information

```
struct ret_type{  
    int a;  
    char* carray ;  
    unsigned int error_number ;  
    char error_name[255] ;  
} ;
```

errno

- UNIX mechanism to pass error information
- including many “standard” errors [“out of memory”, “disk full”, etc.]
- integer that stores the error values
- “errno” program can print the values (for all, or specific ones)


```
$ errno -1
EPERM 1 Operation not permitted
ENOENT 2 No such file or directory
ESRCH 3 No such process
EINTR 4 Interrupted system call
EIO 5 Input/output error
...
```

```
$ errno 28
ENOSPC 28 No space left on device
```


#include <error.h>

- to reference “errno”
- Additional functions

Function	Operation
<code>perror</code>	print an error to console AND error corresponding to “errno”
<code>strerror</code>	returns an error corresponding to “errno”



Memory Ownership

who will “free ()”?

- pointers are often passed to/from functions
- who has the responsibility to call `free ()` ?
- **note:**
 1. C requires that `free ()` be called **explicitly!**
 2. it should be called only **once**
- caller/callee should understand who “**owns**” the resources pointed to by the pointers

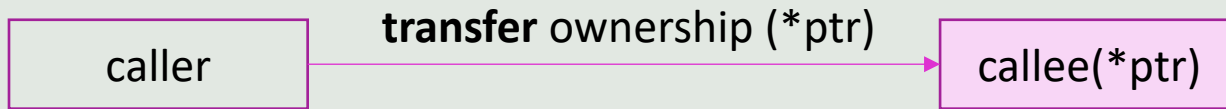
The slide features a light green background with several decorative circles of various colors (pink, purple, blue, orange, cyan) scattered in the corners. The word "Ownership" is written in a large, dark blue font on the left side.

Ownership

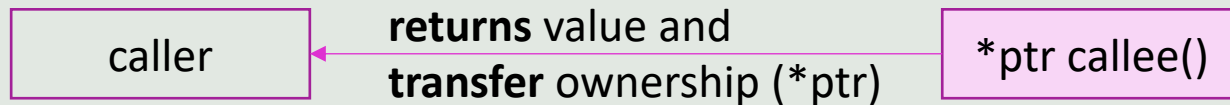
- Owner of a piece of memory is the one that frees it
- OR
- Pass it to another piece of code that then becomes the owner

- A function can **borrow** some memory **but does not free it!**

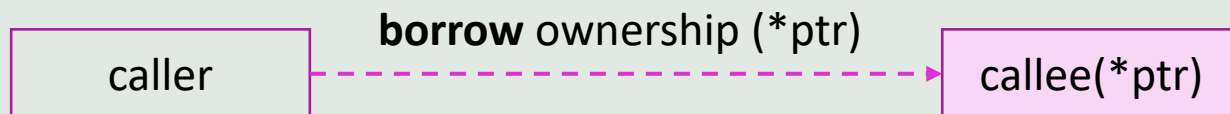
Ownership Patterns



- Example: **enqueue** data into a queue



- Example: **strdup** creates a string and the caller will free it



- Example: **most common case**