CSCI 2410 Systems Programming

Fall 2026

Instructor: Jie Zhou

Department of Computer Science

George Washington University



Slides materials are mostly credited to Gabriel Parmer and Sibin Mohan at GWU.

Application Programming Interface (API)

- Method for computer programs to communicate with each other
- Offering services to other software/code
- Software interfaces, not user interfaces

Function Interfaces

- What functionality does your code provide?
- The way that your code can be used by others
- What goes into your `.h` file?
- Can also include type definitions (e.g., structs) and global variable declarations

Example

```
struct ptrie;

struct ptrie *ptrie_allocate(void);

void ptrie_free(struct ptrie *pt);

int ptrie_add(struct ptrie *pt, const char *str);

char *ptrie_autocomplete(struct ptrie *pt, const char *str);

void ptrie_print(struct ptrie *pt);
```

- We do not need to know what is inside structure `ptrie`.
- That is implementation details.
  - However this `struct` is defined in a `.c` or `./h` file.
- Everyone can implement it differently.

API

```
struct ptrie;

struct ptrie *ptrie_allocate(void);

void ptrie_free(struct ptrie *pt);

int ptrie_add(struct ptrie *pt, const char *str);

char *ptrie_autocomplete(struct ptrie *pt, const char *str);

void ptrie_print(struct ptrie *pt);
```

- These `ptrie`-related APIs only guarantee
  - allocate
  - free
  - add new entries
  - autocomplete functionality
  - print

Specify API Behaviors

```
struct ptrie;

struct ptrie *ptrie_allocate(void);

void ptrie_free(struct ptrie *pt);

int ptrie_add(struct ptrie *pt, const char *str);

char *ptrie_autocomplete(struct ptrie *pt, const char *str);

void ptrie_print(struct ptrie *pt);
```

- What happens if `ptrie_add()` receives NULL?
- Who is responsible for freeing the string returned by `ptrie_autocomplete()`?
- How does `ptrie_add()` indicate failure?
- ……

Return Values

- Actual return,
  - results of computation, e.g., memory allocation, computed values, etc.
- Errors (if any)
- Additional context, such as execution logs
- Further concerns
  - How to interpret return values?
  - Return more than one value?

Return Values in C

- Explicit return a value
  - e.g., `int sum(int a, int b);`
- Implicit return using global variables
  - e.g., `errno` that indicates a specific type of error
- Return through a pointer parameter
  - e.g., `strcpy(char *dst, const char *src);`



`parse_int()` converts a string to an int. Which version is better?

```c
int parse_int(const char *str);

int parse_int(const char *str, int *result);
```

Return Multiple Values at Once

- Use a struct to group all values
  - Return the entire struct
  - Or return a pointer to the struct

```
struct return_type{
    int _a;
    char* _name;
};

struct return_type my_function_that_returns_an_object(…);
struct return_type* my_other_function_that_returns_a_ptr(…);
```

Errors

- Sometimes, errors happen.
  - e.g., when we call `malloc`, system is out of memory
- When errors occur, we want more *context*.
  - What was the exact error?
  - Where did the error occur?
  - What caused the error?
- Programmers can make *informed* choices on how to respond.

Return Values can Indicate Errors

Function Return	Error Indicator	Example
integers	negative values	`printf()`
pointers	NULL	`malloc()`
`struct`	fields in struct	user defined

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

printf()

```
SYNOPSIS
     #include <stdio.h>

     int
     printf(const char * restrict format, ...);
```

```
RETURN VALUES
     These functions return the number of characters printed (not including the trailing '\0' used to end
     output to strings), except for snprintf() and vsnprintf(), which return the number of characters that
     would have been printed if the size were unlimited (again, not including the final '\0').  These
     functions return a negative value if an error occurs.
```

Quiz

 What does this code print?

```
printf("\n%d", printf("world") + printf("hello") - printf ("!"));
```

worldhello!
9

Error Handling Mechanisms in Unix and C

- `errno`: a global integer variable representing common errors
  - e.g., out of memory, disk full, etc.
  - Part of C standard
  - Extended by Unix/Linux with many more errors
  - Defined in `<errno.h>`

Common Memory Management Errors

- Failed allocation: memory not allocated
  - Maybe not enough memory in system
  - or requested too large a piece of memory
  - returns NULL
  - check for the return value before use!

```
int foo (void)
{
    int *a = malloc(sizeof(int));
    if (a == NULL) {
        perror("malloc");
    }
    *a = 1;
    free(a);

    return 0;
}
```

`errno` Command Line Tool

```
$ errno 1
EPERM 1 Operation not permitted
```

```
$ errno -l
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
ENXIO 6 Device not configured
E2BIG 7 Argument list too long
ENOEXEC 8 Exec format error
EBADF 9 Bad file descriptor
ECHILD 10 No child processes
EDEADLK 11 Resource deadlock avoided
ENOMEM 12 Cannot allocate memory
EACCES 13 Permission denied
EFAULT 14 Bad address
ENOTBLK 15 Block device required
EBUSY 16 Resource busy
EEXIST 17 File exists
EXDEV 18 Cross-device link
ENODEV 19 Operation not supported by device
ENOTDIR 20 Not a directory
EISDIR 21 Is a directory
EINVAL 22 Invalid argument
ENFILE 23 Too many open files in system
```

There are over 100 error types!

Additional Helper Functions

- Two additional APIs in libc for dealing with errors:

function	Operation	Defined in
`perror()`	print an error to the console, corresponding to `errno`	`stdio.h`
`stderr()`	return a string corresponding to `errno`	`string.h`