LibFT Project Guide

Core Concepts and Implementation Guide

1. Memory Management

Understanding pointers and memory allocation is crucial. Key functions to implement:

```
ft_memset()  // Fill memory with a constant byte
ft_memcpy()  // Copy memory area
ft_memmove() // Copy memory area, handling overlapping
ft_memchr()  // Search for a byte in memory
ft_memcmp()  // Compare memory areas
ft_calloc()  // Allocate and zero-initialize memory
```

2. String Manipulation

Understanding how strings work in C (null-terminated arrays). Basic string operations:

```
ft_strlen()  // Get string length
ft_strchr()  // Find character in string
ft_strrchr() // Find character from end of string
ft_strlcpy() // Safe string copy
ft_strlcat() // Safe string concatenation
ft_strncmp() // Compare strings up to n bytes
ft_strnstr() // Find substring in string
ft_strdup()  // Duplicate string
```

3. Character Classification

ASCII table understanding. Character type checking:

```
ft_isalpha() // Is alphabetic character
ft_isdigit() // Is numeric character
ft_isalnum() // Is alphanumeric
ft_isascii() // Is in ASCII range
ft_isprint() // Is printable character
ft_toupper() // Convert to uppercase
ft_tolower() // Convert to lowercase
```

4. String to Number Conversion

Understanding ASCII to integer conversion:

```
ft_atoi()    // Convert string to integer
```

5. Advanced String Manipulation (Part 2)

Dynamic memory allocation and string operations requiring memory management:

```
ft_substr()  // Get substring from string
ft_strjoin() // Join two strings
ft_strtrim() // Trim characters from string
ft_split()   // Split string into array by delimiter
ft_itoa()    // Convert integer to string
```

6. Function Pointers (Part 2)

Understanding how to work with function pointers:

```
ft_strmapi()  // Apply function to each char
ft_striteri() // Apply function to each char with index
```

7. File Descriptors (Part 2)

Basic understanding of file descriptors. Output functions:

```
ft_putchar_fd() // Write char to file descriptor
ft_putstr_fd()  // Write string to file descriptor
ft_putendl_fd() // Write string with newline
ft_putnbr_fd()  // Write number to file descriptor
```

8. Linked Lists (Bonus Part)

Understanding linked list data structures. List operations:

```
ft_lstnew()      // Create new list node
ft_lstadd_front()// Add node at beginning
ft_lstadd_back() // Add node at end
ft_lstsize()     // Count list elements
ft_lstlast()     // Get last node
ft_lstdelone()   // Delete single node
ft_lstclear()    // Delete entire list
ft_lstiter()     // Apply function to each node
ft_lstmap()      // Create new list applying function
```

Essential Concepts to Master

1. Pointer Arithmetic

- How to navigate through memory
- Difference between pointers and arrays

2. Memory Safety

- Buffer overflow prevention
- Null termination
- Memory leaks prevention
- Proper memory allocation/deallocation

3. String Concepts

- Null termination
- String length vs buffer size
- String copying vs referencing

4. Error Handling

- Handling null pointers
- Handling edge cases
- Input validation

Study Resources

1. The C Programming Language (K&R) - especially chapters on pointers and strings
2. man pages for original functions
3. Online resources for visualizing memory and pointers

Testing Tips

1. Test edge cases (null pointers, empty strings, etc.)
2. Use tools like valgrind to check for memory leaks
3. Create comprehensive unit tests
4. Compare your output with the original functions

General Tips

- Start with the simpler functions (like ft_strlen, ft_isalpha)
- Use debugger to understand memory behavior
- Write thorough tests for each function
- Pay attention to memory management
- Study the man pages of original functions carefully