Owl Tech Industries

Junior Developer Onboarding Program

Assignment 1: Number Base Conversion Utility

Course:	CS 3503 - Comp Org & Arch
Assignment:	A1 - Data Representation
Language:	C Programming
Topics:	Number bases, bit manipulation
Version:	2.0 (Fixed)

Welcome to Owl Tech!

Welcome to your first C programming assignment!
At Owl Tech Industries, we develop low-level system utilities. Your first project is to build a number base conversion tool - a utility that converts between binary, octal, decimal, and hexadecimal representations. This tool helps engineers debug hardware interfaces and analyze memory patterns.
This assignment will introduce you to C programming while teaching fundamental concepts about how computers represent data.

1 Assignment Overview

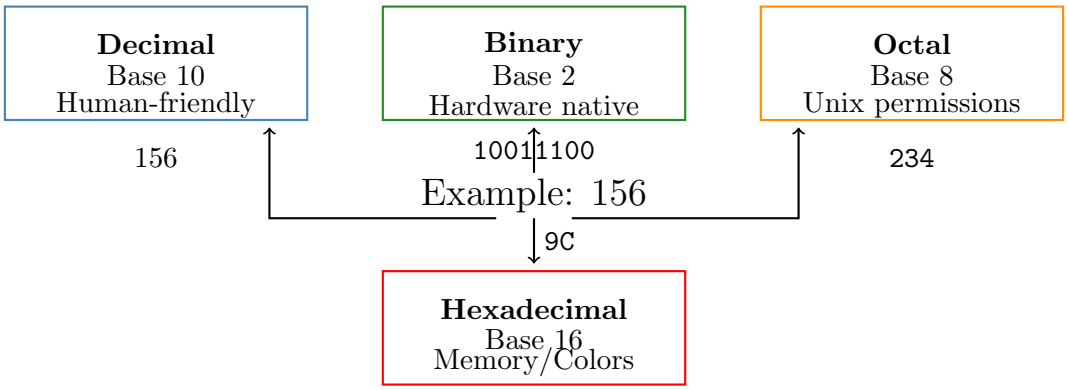
1.1 What You'll Build

You'll create a number base conversion utility with two different conversion algorithms and a bit manipulation demonstration. This project introduces:

- C programming fundamentals
- Number representation in different bases
- Classic conversion algorithms
- Bitwise operations

1.2 Why This Matters

Number Representations in Computer Systems



2 C Programming Crash Course

New to C? Start Here!

If you're coming from Python, Java, or C#, C might feel different at first. Here's what makes C special:

- **Direct hardware control** - You're working close to the metal
- **Manual memory management** - No garbage collector to help
- **Explicit about everything** - C makes you specify exactly what you want

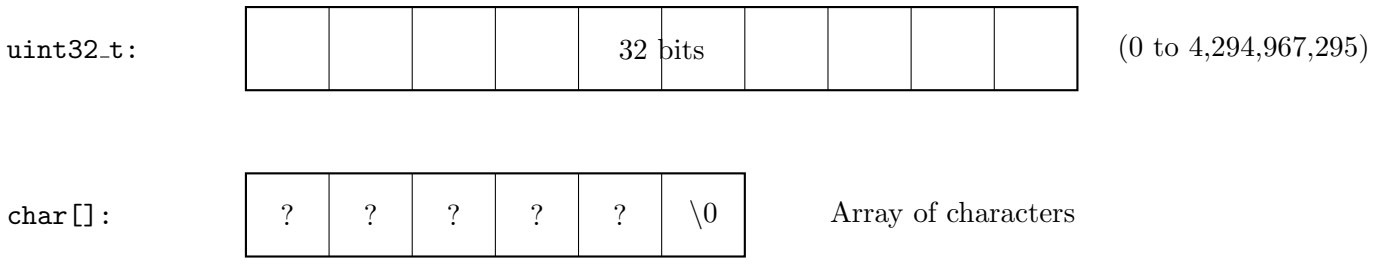
Don't worry - we'll guide you through everything you need!

2.1 Your First C Concepts

2.1.1 Variables and Data Types

In C, you must declare the type and size of every variable:

C Data Types for This Assignment



```c
#include <stdint.h>  // For uint32_t

// Numbers with exact sizes
uint32_t number = 156;        // Unsigned 32-bit integer
char     digit = '5';         // Single character

// Arrays (like lists, but fixed size)
char binary_string[33];       // Space for 32 bits + '\0'
char hex_string[9];           // Space for 8 hex digits + '\0'
```

Listing 1: Essential C Variables

2.1.2 Strings in C

Critical: C Strings

Unlike Python or Java, C strings are just arrays of characters that MUST end with '\0' (null terminator).

```c
char buffer[10];              // Array of 10 characters

// Building a string character by character
buffer[0] = 'H';
buffer[1] = 'i';
buffer[2] = '\0';             // MUST terminate!

// Using string functions
strcpy(buffer, "Hello");      // Copies "Hello\0"
strcat(buffer, " World");     // Appends (careful about size!)
```

Listing 2: Working with C Strings

2.1.3 Functions and Pointers

When you pass arrays to functions in C, you're actually passing a pointer (memory address):

```c
// This function can modify the original array!
void my_function(char *output, uint32_t input) {
    // output points to the caller's array
    sprintf(output, "%u", input);  // Write to caller's array
}

// Usage
char result[20];
my_function(result, 42);  // result now contains "42"
```

Listing 3: Functions with Arrays

3 The Assignment: Number Base Converter

3.1 What You'll Build

You'll create a utility with three main functions:

1. **div_convert** - Uses division algorithm for base conversion
2. **sub_convert** - Uses subtraction algorithm for base conversion
3. **print_tables** - Demonstrates bit manipulation operations

3.2 Function Specifications

3.2.1 Function 1: div_convert

Purpose: Convert a number to any base using the division/remainder algorithm
Prototype: `void div_convert(uint32_t n, int base, char *out)`
Algorithm: Repeatedly divide by base, collecting remainders

Division Algorithm Example: 156 to base 8

$156 \div 8 = 19$ remainder **4**
$19 \div 8 = 2$ remainder **3** $\longrightarrow$ Result: **234**
$2 \div 8 = 0$ remainder **2** (read remainders bottom to top)

3.2.2 Function 2: sub_convert

Purpose: Convert using subtraction of powers
Prototype: `void sub_convert(uint32_t n, int base, char *out)`
Algorithm: Find highest power, subtract multiples

3.2.3 Function 3: print_tables

Purpose: Display bit manipulation results
Prototype: `void print_tables(uint32_t n)`
Operations: Show original, left shift by 3, AND with 0xFF

IMPORTANT - Output Format:
Each operation must output exactly in this format:

```
Original:  Binary=<binary> Octal=<octal> Decimal=<decimal> Hex=<hex>
Left Shift by 3:  Binary=<binary> Octal=<octal> Decimal=<decimal> Hex=<hex>
AND with 0xFF: Binary=<binary> Octal=<octal> Decimal=<decimal> Hex=<hex>
```

Example print_tables Output

For `print_tables(156)`, your output should be exactly:

```
Original: Binary=10011100 Octal=234 Decimal=156 Hex=9C
Left Shift by 3: Binary=110011100000 Octal=4540 Decimal=1248 Hex=4E0
AND with 0xFF: Binary=10011100 Octal=234 Decimal=156 Hex=9C
```

3.3 Implementation Hints

```c
void div_convert(uint32_t n, int base, char *out) {
    char temp[65];  // Temporary buffer
    int pos = 0;

    // Handle zero case
    if (n == 0) {
        strcpy(out, "0");
        return;
    }

    // Extract digits from right to left
    while (n > 0) {
        int remainder = n % base;
        n = n / base;

        // Convert digit to character
        if (remainder < 10)
            temp[pos++] = '0' + remainder;
        else
            temp[pos++] = 'A' + (remainder - 10);
    }

    // Reverse the result
    // TODO: Your code here!
}
```

Listing 4: Division Algorithm Skeleton

```c
void print_tables(uint32_t n) {
    char bin[33], oct[12], hex[9];

    // Original number
    div_convert(n, 2, bin);
    div_convert(n, 8, oct);
    div_convert(n, 16, hex);
    printf("Original: Binary=%s Octal=%s Decimal=%u Hex=%s\n",
           bin, oct, n, hex);

    // Left shift by 3
    uint32_t shifted = n << 3;
    div_convert(shifted, 2, bin);
    div_convert(shifted, 8, oct);
    div_convert(shifted, 16, hex);
    printf("Left Shift by 3: Binary=%s Octal=%s Decimal=%u Hex=%s\n",
           bin, oct, shifted, hex);

    // AND with 0xFF
    uint32_t masked = n & 0xFF;
    // ... similar pattern
}
```

Listing 5: print_tables Implementation Hint

4 Testing Your Code

4.1 Test File Format

You'll receive a test file (`A1_tests.txt`) with test cases. Your `main.c` should:

1. Read and parse the test file
2. Run each test case
3. Compare your output with the expected results
4. Print PASS/FAIL for each test

Important: Test File Corrections

The original test file had some errors that have been fixed:

- The test case for 3405774592 in hex has been corrected from CAFEBABE to CAFFFF00
- The duplicate content after "End of test file" has been removed
- The print_tables output format has been clarified

Please use the updated test file provided with this version of the assignment.

```
Test 1: div_convert(156, 8) -> Expected: "234", Got: "234" [PASS]
Test 2: div_convert(255, 16) -> Expected: "FF", Got: "FF" [PASS]
Test 3: sub_convert(104, 5) -> Expected: "404", Got: "404" [PASS]
Test 4: print_tables(5) -> [FORMATTED OUTPUT CHECK] [PASS]
...
Summary: 145/147 tests passed
```

Listing 6: Example Test Output

5 Deliverables

5.1 What to Submit

NOTE: Your Git repository isn't limited to GitHub; you are welcome to use another source control solution. The idea is to start building out your portfolio or continue adding to it.

GitHub Repository

- convert.c
- main.c
- README.md
- A1_tests.txt (fixed version)

D2L Submission

- convert.c
- main.c
- output.txt
- GitHub URL (in comments)

Both Required for Full Credit!

5.2 README Template

Note on Documentation

This README template is just an example - feel free to use your own format! The important thing is to include build instructions and document your work clearly.

```
# CS 3503 Assignment 1 - Number Base Conversion

## Author
[Your Name]

## Description
My implementation of Owl Tech's number base conversion utility.

## Build Instructions
```bash
gcc -o convert convert.c main.c
./convert
```

Test Results
[X/Y tests passed - paste summary here]

Notes
[Any interesting discoveries or challenges]
```

Listing 7: Example README.md

6 Getting Help

6.1 Owl Tech Resources

Support Available

- **Teams Channel**: Ask questions, share insights
- **Office Hours**: Get help with C concepts
- **GTA**: Check for GTA availability

6.2 Useful References

- C Reference: `https://en.cppreference.com/w/c`
- Bitwise Operations: `https://www.programiz.com/c-programming/bitwise-operators`

A Appendix: C Programming Quick Reference

A.1 A. Common C Patterns You'll Need

```c
FILE *file = fopen("A1_tests.txt", "r");
if (file == NULL) {
    printf("Error: Could not open file\n");
    return 1;
}

char line[256];
while (fgets(line, sizeof(line), file) != NULL) {
    // Process each line
    // Remove newline: line[strcspn(line, "\n")] = '\0';
}
fclose(file);
```

Listing 8: Reading Files in C

```c
// Method 1: Character by character
char result[33];
int pos = 0;
result[pos++] = '1';
result[pos++] = '0';
result[pos] = '\0';  // Always terminate!

// Method 2: Using sprintf
char buffer[100];
sprintf(buffer, "Value: %u in hex: %X", 156, 156);

// Method 3: String concatenation
char output[100] = "";  // Start empty
strcat(output, "Binary: ");
strcat(output, "10011100");
```

Listing 9: String Building Patterns

A.2 B. Bit Operations Reference

Bit Operations Visualized

Original:	00001010 (10)
Left shift by 2:	00101000 (40)
AND with 0x0F:	00001010 (10)
Right shift by 1:	00000101 (5)

A.3 C. Debugging Tips

```c
// Simple debug macro
#define DEBUG(msg, val) printf("[DEBUG] %s = %u\n", msg, val)

// Usage
```

```
uint32_t result = 42;
DEBUG("result", result);  // Prints: [DEBUG] result = 42

// Conditional compilation
#ifdef DEBUG_MODE
    printf("Intermediate value: %u\n", temp);
#endif
```

Listing 10: Debug Printing

A.4 D. Common Mistakes to Avoid

Watch Out For These!

1. **Forgetting null terminator**: Always end strings with '\0'
2. **Buffer overflow**: Make sure arrays are big enough
3. **Uninitialized variables**: C doesn't set variables to 0 automatically
4. **Off-by-one errors**: Remember arrays start at index 0
5. **Test file errors**: Use the updated test file with corrections

Getting Started

Welcome to C programming! This assignment might feel challenging as your first C project, but that's completely normal. Setting up your development environment and learning C's syntax takes time.
Take it step by step:

1. Set up your C compiler (gcc)
2. Start with simple test programs
3. Build one function at a time
4. Test frequently
5. Use the corrected test file provided

By the end of this assignment, you'll have built a real utility that converts between number bases - a fundamental skill in systems programming.
Good luck with your first Owl Tech project!